Glove-based gesture recognition for real-time outdoors robot control

Marc Dupont

UNIVERSITE
BRETAGNE
LOIRE

Université
Bretagne Sud

THESE / UNIVERSITE DE BRETAGNE-SUD
sous le sceau de l'Université Bretagne Loire

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITE DE BRETAGNE-SUD

Mention : Sciences et Technologies de l'Information et de la Communication
Ecole doctorale: SICMA

Présentée par:
Marc Dupont

Préparée à l'Unité Mixte de Recherche CNRS 6074
IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires)

Convention CIFRE N° 2013/9032 avec **Thales Optronique**

Reconnaissance gestuelle par gant de données pour le contrôle temps réel d'un robot mobile

Thèse soutenue le 28 mars 2017
devant le jury composé de :

- Dr Catherine ACHARD (rapporteur),
  Maître de Conférences, Habilitée à Diriger des Recherches
  Université Pierre & Marie Curie, Paris 6
- Prof. Fabien MOUTARDE (rapporteur),
  École Nationale Supérieure des Mines de Paris (ENSMP)
- Prof. Éric ANQUETIL (président),
  INSA de Rennes
- Prof. Pierre-François MARTEAU (directeur de thèse),
  Université Bretagne Sud
- M. Philippe GOSSET (responsable Industriel),
  Thales Optronique, Élancourt

Thèse – Université de Bretagne-Sud

Ecole Doctorale SICMA

Mention STIC

Pour l'obtention du titre de
Docteur de l'Université de Bretagne-Sud

Glove-based gesture recognition for real-time outdoors robot control

Auteur:
Marc Dupont

Encadrement:
Pierre-François Marteau (UBS)
Philippe Gosset (Thales)

Last edited June 13, 2017

Acknowledgements

First of all, I am deeply indebted to my PhD advisor, Pierre-François Marteau, for his incredible patience, advice, and availability during my research; his guidance was the perfect balance between freedom and management. Naturally, I am also extremely grateful to Philippe Gosset, whose leadership was fantastic on the industry side to make such a project happen.

I would like to thank all my colleagues at Thales, and in particular the people of the robotics team: Quentin, Pierre, Nicolas, Sandrine, Christine, Luc, Christelle... I learnt plenty on robotics, electronics, and funny flying things! I also thank Gérard and Thierry for a higher level view of my PhD. Of course, a big thank you goes to Béatrice, Simone, Sandrine, and Pascale. I was also pleased to meet many friendly interns and apprentices, including Felix, Peter, Antoine, Antoine, Idriss, Mélanie, Etienne, Samuel, and many others... A special thanks goes to Clément Letellier, intern during the 2016 summer, who brought me a tremendous help for the hardware design of the data glove presented in this thesis.

On the IRISA side, I would like to thank all permanent researchers, many of whom were the source of exciting discussions, including Sylvie, Pascale, JFK, Salah, Sébastien, Nicolas, Caroline... Many thanks to Mario, Sylviane and Anne! I am also grateful for good times with my fellow PhD student companions, including Maël, Mathieu, Armel, Pamela, Valdemar, Kader, Romain, Adeline, Mathieu, Saeid, Sina, Yanwei, Lei, and many others.

I would like to thank the jury members, Catherine Achard and Fabien Moutarde, for reviewing my thesis manuscript. [To be completed.]

I want to thank my parents, for what seems to be an infinite reservoir of support and kindness since well before I started this PhD. And finally, my sweetest feelings go to Delphine – thank you for having my back no matter what.

Contents

Part I

Presentation

Chapter 1

Introduction

This thesis presents the results of my three years of work as a CIFRE PhD student. In France, CIFRE is a unique scheme to allow a PhD work to take place in an industrial context. Normally, a PhD is a two-party collaboration between the student themselves and an academic institute linked to an university. In a CIFRE PhD, a third party joins the collaboration: a company, interested in the thesis topic, which will employ and fund the PhD student's research.

In my case, thus, I was employed by Thales Group, and more specifically the legal entity Thales Optronique SAS based in Elancourt, near Paris, France; along with academic component University of South Brittany and academic laboratory IRISA.

Thales employs around 65000 collaborators around the world on five different core applications: Aerospace, Space, Transportation, Defence and Security. Thales Optronique is a 1000+ employee company mainly operating in the Defence segment with their military oriented optronics products, incorporating the fusion of groundbreaking research in electronics-augmented optical devices such as numerous camera systems designed for military uses. Because Thales invests massively into research, Thales Optronique is home to some lesser-known departments investigating research-level technology, such as the robotics team which I joined during these three years. The team is further divided in two activities, ground and aerial, respectively investigating ground robots (UGVs, Unmanned Ground Vehicles) and drones (UAVs, Unmanned Aerial Vehicles). I was affected to the ground activities under the guidance of Philippe Gosset, System Architect and lead of the UGV team.

IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires; Insitute of Research in Computer Science and Random Systems) is a 1975-established UMR (Unité Mixte de Recherche, Joint Research Unit) governed by several trustees including UBS (Université de Bretagne Sud; University of South Brittany). Along with joining Thales I also joined IRISA as a PhD student during these three years, under the supervision of Pierre-François Marteau, Professor.

IRISA is a Brittany-based laboratory, located in Vannes, more than 3 hours away from Paris-based Thales Optronique. Despite the distance, the choice of IRISA to solve this thesis

problem is far from absurd, since IRISA is well known for applied machine learning activities regarding real-world applications; in particular, I joined the team Expression, which focuses on human expressivity challenges including gesture recognition and synthesis, speech recognition and synthesis, natural language processing, facial recognition, etc. Expression's track record of exploring gesture-related problems is displayed by works such as sign language automation, hand tracking by cameras, use of gesture for musical purposes, etc.

Due to this long distance between both places of work, I have made many back-and-forth trips between Paris and Vannes during this PhD. I have spent much of my first and third PhD years in Paris while my second year took place in Vannes. I have had the occasion to enjoy the best of both worlds between my familiar, dense and lively home city, Paris, while at times enjoying the astonishing scenery of nature-rich Brittany in the coastal city of Vannes. Assuredly, living in Vannes has changed my perspective on life and I have made outstanding encounters with people I will never forget.

The task proposed initially by Thales Optronique and then accepted by IRISA to initiate this thesis was to research and develop gesture recognition algorithms to control a mobile robot. Named the Z-Trooper, this robot whose development started less than a year before I joined is a ground vehicle weighing around 300 kg; while polyvalent, it was designed from the ground up to assist infantry soldiers and especially carry heavy objects to alleviate back pain on long distances. The gesture recognition demand stems from the desire of the robotics team to propose an innovative means of robotic control on the field. A natural solution would be to use a joystick, but this device is not perfect since it requires being grasped before attempting any operation. A gesture recognition solution would thus allow the user to express commands to the robot without holding a specific object, thereby keeping their hand free. Furthermore, gestures are already well established in the military, and we will show how our gesture control system leverages this existing semantic to provide an intuitive user experience.

It is difficult to find one's place as a PhD student in a CIFRE thesis, since the company and the laboratory have, despite appearences, different expectations. Industrial teams demand engineering results and concrete prototypes; while the academic community wouldn't be satisfied without proof of novelty and state of the art enhancement.

In this regard, I believe to have reasonably well answered to both requirements. On the one hand, my work led to a working prototype for gesture recognition, including not only a custom glove integrating the chosen sensors, but also a software library and front-end GUI to perform acquisition and recognition of gestures in a very resource-efficient fashion. Thanks to this, I have made several demonstrations in different contexts and for different audiences, including HMI research community, industrial engineers, machine learning researchers and even Thales customers during company exhibitions, all of which brought to me very valuable insights each time. On the other hand, to satisfy the academic requirements, it was necessary to choose one topic among the several domains this thesis topic spans (machine learning, human-machine interface, sensors, robotics…) and to explore it deep enough to find innovative solutions and thereby contributing to the academic community. In my case, I have mainly targeted a machine learning specialization: recognition of time series. While it was difficult at first and I ended up reinventing some existing work, I finally found my place thanks to two key contributions: first, a downsampling algorithm to accelerate elastic distance computations (AALTD 2015) and second, a technique for time series detection parameters adjustment and automatic rejection of harmful instances possibly provided by the

user during training (ICPR 2016). As a whole, it is possible that these targeted contributions could make my final gesture recognition system novel in itself. Nevertheless, I have tried to not only pursue high accuracy metrics, but also to keep the human in mind as to provide an intuitive experience in gesture recognition.

This thesis follows a natural unfolding. First, the goal and constraints providing the grounds for all subsequent technical decisions are explained in Chapter 2, followed by a review of prior work and equipment in Chapter 3. Then, we discuss possible gestures in Chapter 4, and move on to sensor choice in Chapter 5 while explaining the design of our custom data glove.

Next, we tackle the system from a software and machine learning point of view; Chapter 6 explains how we formulate the recognition problem as a stream-enabled pipeline; Chapter 7 describes the first important block of this pipeline, namely sensor processing, while Chapter 8 explains how to distinguish gestures when represented as time series and Chapter 9 extends this discussion to a stream-oriented context.

During our journey, experiments will uncover flaws in the training procedures, which we tackle in Chapter 10 by describing how we get rid of low-quality gestural user input and automate hyperparameter selection instead of relying on user tweaking. Facultative speed enhancements are discussed in Chapter 11 when presenting Coarse-DTW and Bubble downsampling.

Finally, Chapter 12 will be the time to step back and verify we have indeed answered both the industrial problem (“it works”) and the academic requirement (“it is novel”).

I hope that by reading this thesis, you will experience the same thrill of scientific experimentation and exploration I have had the chance to be immerged in during these three years.

Chapter 2

Goal and constraints of this thesis

Contents

In this thesis, the principal task we seek to achieve is the recognition of users gestures. But from there, it goes beyond recogntion; indeed, similar systems have been developed in the past and one might rightfully wonder the benefit brought by this thesis' work.

In this doctoral work, we aim at not only recognizing gestures by also – and this is likely the most important part – to do it in a specific use case. As we shall see, at first, this use case might be seen as quite restrictive because it is somewhat peculiar and restricted to a narrow domain. However, the reader will soon understand that the full set of constraints describing our situation actually allows us to develop a system so robust that it can be used in a large variety of real-world scenarii – way beyond the narrow use case we aim for originally.

The specifications of our gesture recognition system can be divided in three parts:

1. main goal,
2. system constraints,
3. additional, but not mandatory, features.

2.1 Main goal

2.1.1 Robot control

Because it is possibly one of the most important part of this thesis, let us highlight the primary objective now:

MAIN GOAL:
Design a gesture recognition system
for accurate robot control
under constraints

Of course, this formulation of the main goal leaves many important questions unanswered: which kind of robot will be controlled? What sort of "control" is needed? What do we precisely mean by "gestures"? We will answer these questions below, keeping the main goal in mind.

2.1.2 Type of commands

The object of this thesis is to control the Z-Trooper, a development robot built by Thales. We briefly reviewed the Z-Tropper's main charateristics in Chapter 1. This robot understands several commands, that we will separate in two categories: high level and low level commands.

2.1.2.1 High level commands

High level commands describe an action involving some nontrivial amount of autonomy, perception, relative localization of the robot itself and other keypoints such as people or manual landmarks.

Example of such commands:

- "come to me": instructs the robot to cease any current automatic exploration and initiate movement towards the current position of the operator.
- "explore": sets the robot in an autonomous mode where it will navigate throughout its environment and create a (possibly partial) map of its surroundings.

Of course, the Z-Trooper can accept other commands, as long as the engineering team provides both the behavior and the interface for such commands. For instance, because the Z-Trooper is still in development, engineers could choose to add a (nonexistent as of today) command "return to home". Therefore it is necessary to take into account the fact that our gesture recognition system will have to understand future commands that we cannot think of, precisely because they don't exist yet.

2.1.2.2 Low level commands

Low level commands do not involve a large degree of autonomy or perception from the robot. As such, these commands can be assimilated as controlling a remote-controlled vehicle.

Example of such comands:

- “go”: the robot starts moving straightahead;
- “left”, “right”: the robot changes direction;
- “faster”, “slower”: the robot increases or decreases its speed;
- “stop”: the robot ceases any motion and stands by.

This kind of control generally asks for more focus from the operator, because it requires coordination during the execution of the task. It is however unnecessary that all autonomy is disabled on the robot side. For instance, one will typically want the autonomous block “anti-collision” to be always enabled, even in manual (low-level) control, in case the operator drives the robot into a wall; anti-collision would take over and halt the motion, even though the “stop” command was not explicitly issued by the operator.

Therefore, in order to communicate the kind of commands as gestures, it is necessary to develop an appropriate set of gestures known as the “gesture dictionary”. It also constitutes a goal of this thesis; once again it will be subject to the constraints we describe thereafter.

2.2 Constraints

In this section, we will dig into the minute details of this gesture recognition application. As the reader shall discover, the set of constraints is quite heavy, and while it seems to be a weakness in that too much constraints makes the task too complex, it is actually a core feature of the whole system because we will focus on designing a very robust and usable application.

Our target user, as we described earlier, is the infantryman working outdoors. Due to this specific end user, its mission, the target robot to be controlled and the environment in which they are immersed, we have identified the following core constraints:

- real-time
- accurate
- mobility tolerant
- compatible with an outdoors environment
- stealth
- robust
- user-friendly

We also identified some additional constraints; those rather belong to the category of engineering and production constraints, and therefore are a secondary focus, that we will keep in mind nonetheless:

- cost-effective
- maintainable and/or replaceable

2.2.1 Real-time

If we want the user to have a true degree of control over the robot, it is necessary for a command issued by the operator to have an immediate effect on the target system. For

example, controlling a robot on a distance planet, such as Curiosity [17], which is only able to receive commands seconds or minutes after it was sent, would not qualify in our situation as real-time [196]. On the other hand, driving a car definitely qualifies as a true real-time control scenario, because the effect of, say, turning the wheel, happens with a latency so small that the user cannot consciously point it out. What does "real-time" mean then? We propose the following definition:

> *A remote control system is said to be "real-time", if the difference of time between expressing an order and the realization of this order itself is not consciously noticed by the user during normal operation.*

It is important to emphasize that most remote control systems actually *do* have some latency anyway, as it is intrinsic to most tasks involving information transfer and especially in the case of electronic and digital systems. Moreover, when asked to focus on detecting and noticing this latency, it is likely that any user will acknowledge some, even though it is not a problem for the remote control task. The question is whether the user can perceive the latency without paying specific attention to it.

Therefore, our system should have a small enough latency that issuing low-level commands is possible without noticing it. In particular, the most critical component of this constraint is the set of commands related to fine-grained mobility, such as "left" or "right", since motion cannot be properly controlled in a high-latency setting. Conversely, we deem the system to be real-time if it can be controlled by a remote operator without difficulty, or at least not related to a latency issue. Hence, this criterion will ultimately be validated if we are able to control the robot's motion via low-level commands.

2.2.2 Accurate

No system is flawless, especially when a human is involved; this is why we should consider the cases when our system will be wrong, and the consequences thereof.

How severe is the cost of accidentally misrecognizing a command, or failing to recognize one? The answer depends on the system being controlled as well as the mission and its context. For example, it doesn't matter if one's TV turns on by itself – apart from a temporary energetic waste and the user complaining to the manufacturer – no harm occurs. It does matter, however, if one's car fails to brake when the corresponding pedal is pressed, or alternatively, if the car brakes without the pedal being pressed; indeed, users are at high risk, putting enormous stress on the manufacturer to build robust equipment, especially regarding software bugs. So, how does our robot fare on this scale of misrecognition, from no harm to possible fatalities?

In terms of mobility, given that the whole vehicle weighs at least 300 kilograms, it is quite problematic if it collides with a human in its course. As such, the cost of mishandling the detection for gesture "stop" could be high if the robot hits a person instead of stopping; in this case, human harm can occur. Of course, safety mechanisms for obstacle avoidance should be always turned on, thereby stopping the robot before colliding with an human; such safety nets should always be enabled during robot operation, especially when humans are involved in the vincinity of the robot. Nevertheless, it is not in the scope of this thesis to discuss obstacle avoidance and we will still consider it a failure of ours when such a critical

gesture as "stop" is not recognized when it should be. Therefore, gesture misrecognitions can carry a potentially high cost; ultimately, we must take utmost care to ensure they happen as unfrequently as possible.

Gesture recognition, as many pattern recognition and machine learning tasks, are seldom 100% perfect. Typically, researchers aim to obtain the best performance of recognition, and this will obviously be our strategy in this thesis while we approach the task as a machine learning problem. There is no doubt that recognition errors will occur during normal operation, and we do not pretend to build a perfect system here. Precautions must be taken so that errors do not have harmful consequences.

Generally in machine learning, and in our gesture recognition application in particular, one can distinguish two kinds of misrecognitions:

- False Positive (FP), or "wrong", occurs when the system accidentally detects a gesture which was not actually performed by the user.
  - Example of a False Positive: Operator is walking; hand swing movement occurs naturally but is not intended to be detected as a command; yet, robot recognizes a gesture such as "go". Consequence: robot starts moving without user intent.
- False Negative (FN), or "miss", occurs when a gesture is being done with a conscious attempt to carry meaning, but is not actually detected.
  - Example of a False Negative: Operator consciously performs gesture "stop" but the system recognizes no gesture. Consequence: the robot keeps moving even though it was expected to stop.

In general, False Positives (wrongs) and False Negatives (misses) are tightly coupled. Indeed, a recognition system which is very tolerant might trigger many detections, resulting in few misses but possibly more wrong detections if the system is too tolerant. Conversely, a system being too strict might generate fewer wrong detections, but also miss many of the rightful ones which were expected to be detected. This balance of tolerance/strictness is very strong and will play a major part for the evaluation of our streaming system. Our gesture recognition problem is a case of a multiclass recognition problem, where we have the dual task of spotting gestures as they arrive and classify them with the correct label. As we shall see later, pure classification does not apply well since there is not one obvious definition of an "instance". Indeed, events occur in a stream where gesture segmentation is ill-defined, and we will discuss this problem in Chapter 9 when tackling the specific challenge of detecting gestures within a stream. Nevertheless, ideas related to this strictness/tolerance tension still remain true by the very nature of this problem, and we will define what FP and FN means in our streaming scenario.

It is difficult to decide if there is a "good choice" between False Positives and False Negatives: too many False Positives equate to spurious detections and ultimately lead to erratic behavior of the controlled system, while too many False Negatives basically make the system unusable and unresponsive. Naturally, we would like to avoid either one if possible. It is therefore important that we seek a system with optimal balance, which is generally characterized by a balanced number of FPs and FNs. But what really matters is the recognizer system being able to reach an optimum leading to *both* a low number of FPs and FNs. On a ROC curve (if this were a binary classification problem), this would amount to the curve

reaching the upper left corner of the diagram. Unsurprisingly, this is often how recognition systems are evaluated in the end, because a low number of false detections is ultimately what matters the most in terms of usability.

We defer to Chapter 9 for a discussion of how a streaming-system such as ours should be evaluated. For now, let us simply highlight that our goal is to create an accurate system, in terms of recognizing as many intended gestures and triggering as few unwanted detections as possible. Indeed, recognition failures can lead to dangerous behavior for the target robotic system.

2.2.3 Mobility tolerant

As we have noted in the previous description of our target users, they are typically meant to work in outdoors settings requiring them to move often, with a high degree of motion. Therefore, our gesture recognition system is expected to be capable of handling gestures performed in what we call a high mobility setting: apart from the classical "standing" position where a user is upright and doesn't move particularly, it also includes more unusual motions (for gesture recognition applications, that is) like *walking* and *running*. Indeed, if the robot cannot be controlled during a light run at a bare minimum, one can argue it will be hardly useful to military operators on the field, since they cannot be expected to stop for the sake of issuing a command to the robot.

This constraint is somehow unusual in gesture recognition applications. To our knowledge, it was not addressed explicitly in state-of-the-art gesture recognition systems (we review them in Chapter 3), at least not with a specific focus on outdoors and mobility. As such, we believe accomodating these constraints is an important step forward for gesture recognition systems, and thus an important contribution of our work.

Moreover, this constraint will be addressed throughout the design of the whole system. We will adopt a bottom-up approach, starting from the sensor choice (which sensors can be used outdoors and work when the operator is running?) up to the algorithms leading to sensor processing (in particular, designing a good correction for the numerical integration of sensor data) and the time series detection and classification engine leading to outputting a detected gesture.

What is so special about mobility that deserves to be addressed specifically? The reason appears most clearly when discussing sensors. First of all, it rules out sensors that require the user to stand in a specific field of view, such as a camera, precisely because that would restrict their motion in a very small area, a constraint that we will never be able to demand of military users. In Chapter 5, we will see how these constraints will lead us into choosing a motion sensor known as an IMU (Inertial Measurement Unit), which is a small integrated circuit outputting key motion information: acceleration and rotation. It can also be used to determine the direction of the gravity and therefore the orientation of the sensor with respect to the world. We will therefore make sure that all subsequent blocks in the recognition pipeline handle properly any motion that could be interpreted as an purposeful gesture, even though it would only be the result of natural running motion; such meaningless gestures are called *parasitic gestures* [21, 37].

Another part of working out the mobility problem is making sure that the gesture dictionary we design in Chapter 4 is consistent with such a mobility context. It would be prob-

lematic if we were to suggest gestures that can't be performed when running, for example! Nevertheless, as we shall see later, the user will be able to teach arbitrary gestures, therefore they have the ability to customize the gesture dictionary.

2.2.4 Outdoors compatible

The outdoor compatibility constraint is a main tenet of this thesis and will drive many early design decisions, in particular regarding the choice of sensors. This is, once again, due to the particularities of our target user, the infantry soldier. Obviously, we shall have a hard time seriously proposing a gesture recognition system for infantry soldiers if it cannot hold typical environment conditions of an outdoors environment as lived by militarymen. The consequence of this constraint on our work will be mostly limited to the choice of hardware technology, because we will choose sensors whose results don't depend on weather, luminosity, etc., and therefore the subsequent blocks of the processing pipeline will not have to care about such environmental details. This will preclude the use of sensors that cannot be installed in an outdoors environment because they cannot be practicaly rendered waterproof, or because using them in a rainy environment will alter the signal significantly enough that they cannot be relied upon anymore.

In our case, outdoors constraint roughly boil down to the system being fully usable in the following conditions:

- weather: sunny, rainy, snowy, ...
- luminosity: day, night, haze, ...
- temperature/climate: cold, continental, warm, hot, desertic, ...

An additional constraint based on the outdoors requirement is the impossibility to use fixed, immobile hardware for the sensors to work. For example, it forbids us to choose a system needing a static, on-scene motion tracker [236]. Obviously, this constraint is also highly imposed by the fact that the user will be next to the robot they control, and this robot is intended to be deployed in unprepared areas where it is clearly impossible, let alone much impractical, to install such static equipment ahead of time.

Practically speaking, the sensors will have to be waterproof and robust with respect to temperature extremes and variations, in order to satisfy military constraints. We would like to clarify the approach taken in this thesis with regards to the weather and temperature constraints. What we did was design the sensor chain (i.e. choose the sensor) so that we can reasonably be sure of the resistance to weather, luminosity and temperature, especially by common sense and by reading datasheets. What we didn't do, though, was a thorough testing of sensors within different simulated and real temperature conditions, nor waterproofness certifications such as IP 68 [209], which would have been a proper scientific demonstration of such claims. We consider such experiments to be engineering work that can be conducted in a subsequent continuation phase of this thesis. The main reason we did not carry out such tests was our focus on the true unknown of this thesis: real-time, accurate recognition, and above all: mobility-tolerant.

Ultimately, by fulfilling requirements to the lowest common denominator in terms of environment constraints, i.e. military, we actually render this system usable for many people,

not just military personnel. Indeed, being agnostic to rain, night and temperature is an excellent capability for the regular user, which implies that our system being outdoors compatible will possibly appeal to many user segments outside of the military.

2.2.5 Stealth

In this subsection, we describe a constraint which applies specifically to military users and furthermore explains the choice of gesture recognition as a means of control. Ground troops are immersed in possibly unknown and even potentially hostile environments, due to the very nature of their work [144]. For this reason, they may encouter the need, at any time, to be undetectable with regards to an enemy presence nearby; it would be a severe liability for them to get detected. Generally though, it is not necessary to be in a mission where stealth is an absolute concern to desire a soundless control system; we would have a hard time picturing infantrymen shouting at the robot to make it move, or otherwise having a conspicuous attitude, and so would they.

In the light of the military context, one can summarize this constraint as *stealth*. There are several, apparently disjoint domains on which stealth arises:

- Acoustic stealth. The command system should not make any noise, nor require the user to make specific sounds, in order to emit a command.
- Visual stealth. The control system should not make an invisible group of infantrymen reveal their location because of issuing a command. However, it is fine to be seen if the group is already detected. For example, a system requiring flashing lights would be rejected, but a gesture system is fine as long as it doesn't require the user to make extensively showy gesturing.
- Infrared stealth. The system should avoid transmitting too much light in the near-visible spectrum, especially in the infrared range, where many sensor (such as RGB-D cameras) work [172, 255].
- Radio stealth. Being radio stealth means not emitting any signal in the radio electromagnetic range, especially regarding wireless communication happening in the microwaves for short-distance, high-speed communications.

In the project leading to this thesis, acoustic stealth was deemed a priority for the system. For this very reason, even though there are quite strong candidates in the domain of speech recognition solutions, they were excluded from the beginning. On the contrary, gesture recognition addresses quite well the acoustic stealth requirement because it intrisically does not require any audio signal or vocal clues to work. As a side note, let us clarify that the decision to consider gesture- instead of speech-based command was not a part of this thesis, but was undertaken before its beginning. Therefore, our work took place in the context of gesture from the beginning, not as an afterchoice in regards to the list of constraints.

How does a gesture-based recognition system fare with respect to those stealth requirements?

First, let us consider audio stealth. Obviously a gesture recognition system does not require speech or sounds, so it matches this requirement quite well. Moreover, performing the gestures should be easily possible without generating additional noise due to specific objects or cloth surface friction.

Second, the visual stealth point is debatable. On the one hand, gestures are obvious visual cues. On the other hand, they are essentially visible if the gesture performer is himself or herself visible in the first place. In addition, the gestures should not lead to significant visual side-effect such as disproportionate lighting or blinking, which would in turn lead to the operator being detected. As such, we consider our gesture recognition system acceptable by this requirement.

Third, the infrared stealth requirement asks whether the system generates significant IR lighting. It depends very much on the chosen hardware. The principal culprit of infrared emission is the line of RGB-D (Red Blue Green Depth) cameras such as the Kinect, which obtain depth information by emitting many infrared rays and observing back the size and reflection of the observed infrared dots that were initially sent. For this reason, such cameras are very "bright" objects when observed through an infrared camera, which makes detection very easy for an enemy. We will later see that it is possible to choose sensors emitting no infrared light.

Last, radio stealth is not as well covered. Most of this requirement does not necessarily involve only the recognition system itself, but also the communication medium through which detected gesture information will be transmitted. Here, we consider it inevitable to have a wireless system at some point on the command chain, because it is inconceivable to deliver a system requiring physical connection between the user and the robot. Given the current technology, the only case for a wired system where the user is not bound would be a camera system; in this case the unconnected part of the chain is between the user and the camera. However, as we shall see later, a camera will turn out to be unsuitable for our constraints. We ultimately will need to choose a system requiring at least some radio communications; we therefore fail to address this problem. The only upside is that militarymen already have always-on radio communication systems, so we would just be another layer on top of an existing situation.

2.2.6 Robustness

Military troops are involved in an outdoors environment requiring them to navigate through, for example, hilly terrains or narrow forests. For this reason, military users need a device that can support shocks, drops, pressure, etc. Moreover, designing a glove for military users entails thinking about their use of the hand for supporting their weight, helping them get up, leaning against a rock, etc. This is reflected in the choice of sensors we will make: a sensor which is too fragile to support such routine use of the hand for on-field operators will not suit our application.

Nevertheless, we do not include, in this requirement, any "extreme" tasks that are part of the military work. For example, our glove will not be designed to be bulletproof, cutproof, or resistant against explosions; however, neither are common military gloves. The rule of thumb here is that we do not account for scenarii where military operators are affected to the point of losing partial of total ability of their hand. In the event of such emergency situations, one can reasonably understand that the primary concern is not the lifeless piece of material that is our glove.

2.2.7 User friendliness

Being easy on the user requires planning beforehand. Most users, military or not, do not like when the equipment is causing more problems that it solves. In this subsection, we distinguish several ways in which the recognition system must be user friendly:

- easy to learn (as a new user)
- easy to train (i.e. show the gestures to the robot)
- easy to setup (when beginning a session)

2.2.7.1 Easiness to learn

Easiness to learn refers to the ability for a new user to get started with the glove, the gesture dictionary, and grasp a basic understanding of how the system works and reacts. We must keep in mind that the user is in no way expected to be an expert in gesture recognition and thus will be using a system that must not require them too much initial competence. Concretely, we wish to present a gesture recognition system that aims for simplicity; users should be able to get a solid grasp on the system fast enough to find it useful. Typically, we estimate that an average user should begin working with the gesture system within minutes, and be proficient in at most one or two house of continuous usage. Nevertheless, mastering the gesture recognition system can be seen as a skill in itself, which means that:

1. some users might naturally be better than others at the beginning
2. prolonged usage will lead to better mastering

All in all, what really matters is that a new user can be up to speed fast enough that it doesn't require weeks of training.

2.2.7.2 Easiness to train

Easiness to train refers to the user performing the "training phase" for the gesture database. Concretely, the training phase mainly involves the user performing asked gestures as demanded by the system. The goal is for the system to gather examples of gestures by the user in order to build its internal detection engine. This is what we call "training" or "teaching" the system. This requirement of being "easy to train" means that a user for which the internal gesture database is not established should not have to spend too much time to build it. We consider the database "built" when the user feels their gestures are well recognized and do not generate too much false detections. Otherwise, it should be possible to continue training the system until the user is satisfied with detection results.

This has a great repercussion on the algorithmic side. Indeed, some machine learning techniques require the use of high-performance hardware to be tractable; for example, Deep Neural Network techniques need hours or days of specialized hardware (GPU clusters) that we have neither the time to train nor the ability to embark on the robot; this will drive us towards simpler, more efficient techniques.

This concept should also be understood in the context of Human-Computer Interaction, and especially in terms of the Graphical User Interface (GUI) making it possible to train the system with gestures. Hence, we will also have a secondary focus on how the user is expected to teach gestures to the system.

2.2.7.3 Easiness to setup

Easiness to setup mostly refers to how much operation is expected by the user before being able to use the system. We focus on the every day operation, not the one-time setup. More concretely, internal feedback from infantry soldiers report an increase in technologic burden, highlighted by the number of equipments to wear, switches to turn on, cables to plug, and batteries to charge constantly. The equipment must not be a weight on the user and therefore it should be contained to a bare minimum. We shall see later that our chosen solution, a glove, reduces this load to a single cable plugging into the radio-enabled robot control device. In contrast, during this thesis many ideas for motion tracking have appeared, such as the installation of multiple sensors on the arm, the back, the hip, etc. leading to ultimately more measurements and thus more ability to distinguish movements. Alas, the reality is that putting on several kits of sensors on the body is painful for the operator which we would be distracting from their core mission; therefore we had to resist against the temptation of sensor accumulation and reject most of these ideas.

In summary, while the initial goal (producing a gesture recognition system to control a robot) was already approached by prior works, the very fact that we design a system catering to such hard and unusual restrictions makes it all the more novel and challenging. By itself, answering to such constraints, especially mobility, is to the best of our knowledge a contribution in gesture recognition systems; additionally, this thesis includes several other contributions, more targeted in the real of time series recognition techniques (downsampling, instance selection, hyperparameter estimation).

By appealing to military users and their specific constraints, we get the side benefit of building a system that is so robust that it can be also used by users having a lower bar of requirements. For example, an outdoors gesture control system can please other families of operators, especially in the domain of professionals in mission, such as firemen, policemen, or industrial applications such as railway inspection, factories, warehouses, etc. Outside of the industry, private individuals can also find a use for such a gesture recognition system. Nonetheless, it is sometimes believed that camera-based contactless systems might overtake glove-based ones [166, 179]; indeed the gaming industry seems to follow this trend [182, 255].

2.3 Additional goals

The set of constraints developed in the previous section refers to core constraints of the project, based on our typical military operator as the main user. Nonetheless, there are some more subtleties that can transform this work from “just military” to “easily applicable in other systems”. Indeed, making our work easily transferable to other domains potentially increases its usefulness. It is to be kept in mind as long as it does not impede on our primary goal, that is, real-time robotic control in mobility for military.

In such light, we have identified two additional goals fulfilling these criteria:

- do not restrict the target system to just a robot
- allow custom gesture dictionaries

2.3.1 Interface-driven gesture recognition

There are other systems sharing characteristics with a robot that can be controlled, and might especially benefit from gesture-based control. They may not necessarily be mobile systems, and they furthermore do not need to possess any form of autonomy to qualify for such gesture control capabilities.

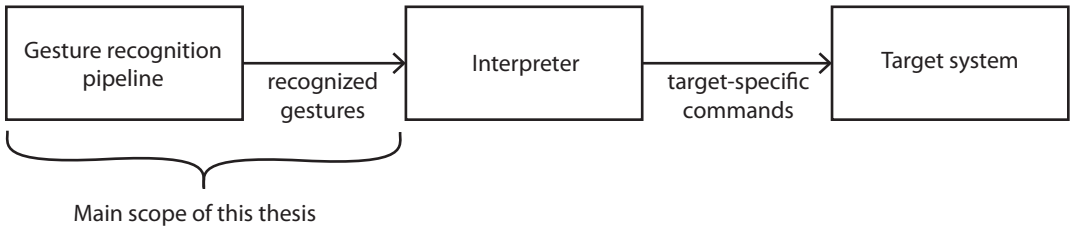


Figure 2.1: This thesis aims to offer a reusable interface to control any system.

Here are a few examples of possible technologies one can think of gesture controlling:

- a pan-tilt camera: two axes of direction can be controlled with 2x2 gestures, for example the "pan" axis can be controlled with gestures "left" and "right" and the "tilt" axis with gestures "up" and "down". Some cameras also feature a depth axis that we all know as "zoom", and it can be therefore controlled with two additional gestures such as "zoom in" and "zoom out". Other features can also qualify for gesture control, such as toggling between day and night vision.
- a slideshow application: although the military constraints are a bit excessive in this case, it is still totally possible to drive a slideshow presentation software with two simple gestures such as "next slide" or "previous slide".
- a computer game: in many aspects, controlling the motion of a robot is akin to controlling a player in a video game. Indeed, history shows that the gaming industry does not lack of application of gaming gloves [45, 164, 217, 232]. A possible extended application could also take place in an augmented reality setting, where the outdoor capability of the glove can be leveraged in a gaming application.

In order to do this, we have chosen to define a clear interface between the gesture recognition pipeline and the controlled system. Therefore, we do not have to talk explicitly to a robot, but to a black-box system which only has to know how to interpret our gesture recognition output.

Of course, gestures being simple labels, it is necessary to somehow interpret them into low-level commands. To that end, it will be necessary to develop a specific piece of software known as a bridge or an interpreter, translating high-level gestures into low-level control as suitable to the target system. This architecture makes it possible to drive any kind of controllable system, achieving cleaner modularity and reusability at no expense on the primary goal of robotic control, since it fits perfectly this architecture.

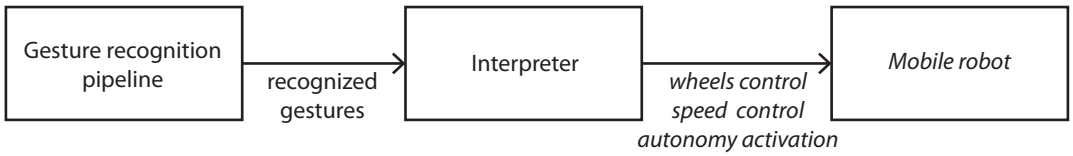


Figure 2.2: In particular, we will attempt to control a mobile robot and focus on this use case.

2.3.2 Custom gesture dictionary

One way to approach the problem of gesture recognition for robot control is to design a pre-configured gesture dictionary [2, 8, 244, 248]. Here, we effectively consider which gestures can be the most useful for the application; yet, giving users the ability to teach gestures outside of our robot control scope soon appeared to be very important. In particular, consider if one of the gestures proposed later (see Chapter 4) is actually unusable in a real-world scenario, has an inappropriate cultural meaning [115], is ergnonomically inacceptable, uncomfortable, difficult, etc. In this case the user should be able to change the gestures.

Therefore, we put users at the forefront of their own gesture control scheme by letting them completely teach their own gesture dictionary to the system, thereby satisfying the demands of different users with a single approach. For this reason, we have chosen to design our system so that a new user gives instances of their own gestures and therefore can teach new ones. The system will not care whether it corresponds to the gesture dictionary we propose. This makes it possible to not only override the proposed gesture dictionary for robot control, but also record gestures better suited to other systems, such as the pan-tilt camera we discussed before.

2.4 Conclusion

The set of constraints we described in this chapter appears to be quite unique, and it is due to the specific military-first application we target. We have also seen that in order to increase broad application of this work, we should think of it as a multi-purpose system and design it around high user configurability.

The next chapter discusses prior art in the different areas spanning this thesis, including existing gesture recognition systems, possible sensors and applied machine learning algorithms.

Chapter 3

Previous work

Contents

In the past decades, previous systems related to hand gesture recognition, data glove, robot control devices have been set up. In this section, we present succinctly a few prior works in these areas, especially previous glove systems; then we move on to their internals: the flexion sensors, for retrieving hand posture, and motion sensor, to obtain the movements described by the user in the context of a gesture.

3.1 Description of the hand's anatomy

This thesis mostly focuses on the computer science part of gesture control, but in order to discuss about the sensors and use some basic terminology, we should stop for a moment and consider the only non-electronical device in the whole system: the hand! It seems like an obvious step, but we ought to describe this biological tool in some details.

In humans, the hand is a prehensile organ which serves multiple purposes: grasping and manipulate objects, sensing textures, hardness, temperatures, etc. In this work, of course, we will focus on one of the hand's task that interests us the most: gesturing. In order to talk with precision about the different postures and motions of the hand, we start by describing its anatomy.

3.1.1 Bones

We are interested in the mobile parts of the hand and how they move together. As such, we will mostly talk about the hand's skeleton as well as its kinematic properties. It can be approximated by ideal joints connected by segments, which reduces the complexity of the hand down to a simple physical system. Joints have special names for the different bones they link. The forearm and hand are composed of these major bones [163]:

- radius and ulna: the two major bones of the forearm;
- carpus: set of bones (carpal bones) at the bottom of the hand, which form the wrist;
- metacarpus: set of bones (metacarpal bones) forming the inner part of the hand, where the "palm" is located;
- phalanges: elementary bones of the fingers

All those bones, with the exception of forearm bones radius and ulna, make up the solid parts of the skeleton system that we call the hand. Now, those bones are not of very much use if they cannot move relative to each other: thankfully for our prehensile capabilities, bones are linked with "joints", which allow them to move more or less freely, depending on the degree of freedom offered by each joints as wel as hard limits imposed by the placement of bones and tendons.

3.1.2 Joints

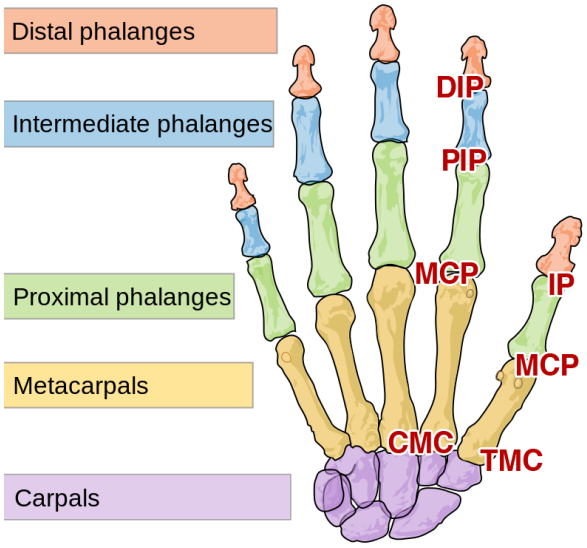


Figure 3.1: Bones and joints of the hand.

3.1.2.1 Wrist joint

The wrist joint (or radiocarpal joint) [163] separates the carpal bones from the radius and ulna, i.e. the base of the hand with the arm. This joint has two degrees of freedom: - in

the plane of the arm: *radial deviation* (*abduction*) described the hand moving towards the thumb; *ulnar deviation* (*adduction*) happens when the hand is moved towards the little finger. Formally it is said that these two opposite rotations take place around the *dorsopalmar axis*, that is, the direction orthogonal to the plane made up by the metacarpus. - in the plane of the hand: *palmar flexion* happens when the hand is flexed towards the palm; one may recall Adam's hand in the Michelangelo painting. Conversely, *palmar extension* is the opposite motion, when the flexion happens towards the back of the hand. The axis of rotation is defined by the *capitate bone*, one of the carpal bones. Technically, only these two degrees of freedom are available for the wrist. However, one can move their hand, not along two, but *three* axes of freedom. Where does this third degree of freedom come from? Not from the wrist joint; actually, it is the consequence of an elbow joint, making the two main bones of the forearm (radius and ulna) rotate around each other. This third axis gives the two following opposite motions: *supination* is the rotation of wrist outwards, like the hand position of a juggler, while *pronation* is the inwards rotation [102, 163], best illustrated by a basketball player.

3.1.2.2 MCP: Metacarpophalangeal joints

A human hand has five MCP joints, one for each finger [49, 162]. They separate the inner part of the hand, that is, the metacarpus, and the base of the fingers, more precisely the first phalanx known as proximal phalanx. MCP joints have two axes of freedom. The first axis is the most obvious: flexion and extension make the biggest rotation in terms of range of motion, and they provide quite visible signals in terms of gestures, as they help tremendously in telling apart an open and a closed finger. Along the second axis, *adduction* and *abduction* of the fingers are respectively the motions of spreading fingers or bringing them back together. It is very difficult to perform abduction (spreading the fingers apart) while one's hand is fully closed [162].

3.1.2.3 PIP: Proximal Interphalangeal joints

A human hand has four PIP joints [49, 162]; one for each of the non-thumb fingers. All PIP joints are placed after their corresponding MCP joint, at the other end of the proximal phalanx. A PIP joint connects the proximal and intermediate phalanges, or the first and second phalanges if numbered from the metacarpus. The PIP joint has a single degree of freedom where both phalanges rotate relative to one another. The inwards movement is known as flexion, while the outwards motion is called extension.

3.1.2.4 DIP: Distal Interphalangeal joints

After the four PIP joints, one finds the DIP joints [49, 162]. They separate the intermediate and distal phalanges, that is, the second and third ones if we follow the numbering scheme of the above paragraph. They are very similar to the PIP joints; indeed, they also have a single degree of freedom, and their axis of rotation is parallel to the one of the PIP joints. We also talk about flexion and extension in much the same way as we do for the PIP joints. Besides, we note a very interesting property; when one slowly close one's own index, one notices that both joints between phalanges (PIP and DIP) move in agreement. There seems

to be a strong correlation; and it seems very difficult to leave one joint open while the other is closed. Such a correlation has indeed been identified by anatomy researchers, and it turns out that both angles follow a linear relationship as follows [133, 162, 166]:

$$\theta_{\text{DIP}} = \frac{2}{3}\theta_{\text{PIP}} \tag{3.1}$$

The key takeaway here is not so much the 2/3 constant in the linear equality, but rather the very fact that those two angles are correlated. In other words, due to this relationship, the kinematic system of one finger loses one degree of freedom on the DIP joint, which had already a single one degree of freedom. Consequently, we can forget about measuring the DIP joints altogether without losing information about the hand; only the PIP joint will suffice.

3.1.2.5 CMC: Carpometacarpal joint

There are five CMC joints, one for each finger[27, 86]. However, the thumb CMC has a special feature: unlike the four other ones, it is able to rotate. So, even if the four other fingers have CMCs, those just strengthen the link and do not bring any degree of freedom. Consequently, we are only interested in the thumb CMC, because it is the sole CMC bringing a possibility of movement. Additionally to the two visible phalanges, the thumb has a third phalanx, hidden in the palm and joining the wrist. It is formally known as the thumb metacarpal bone. The thumb CMC is also called the TMC, or trapeziometacarpal joint, because of its connection to the trapezium and the metacarpus; however, in the following we will just write "CMC" for "thumb CMC", because it is the only useful CMC in terms of movement. Just as with the MCP joints, the thumb has two degrees of freedom, or more precisely two rotations: adduction/abduction refers to the rotation taking place in the same plane as the thumbnail, while flexion/extension happens in a perpendicular plane. Flexion occurs when grasping an object or closing the hand, for example.

3.1.2.6 IP: Interphalangeal joint

The IP [86] designates the only inner joint of the thumb, since other fingers have two distinct joints: DIP and PIP. Regardless, this joint has the same function in the thumb than PIP and DIP have on other fingers. It posesses a single degree of freedom in the form of *flexion* and *extension*. One might note that most people have an augmented ability to perform the *extension* movement – curving the thumb outwards – compared to other fingers, resulting in a prominent curvature on the thumb during extension.

3.1.3 Degrees of freedom

Overall, these 5 types of joints (wrist, CMC, MCP, PIP and DIP / thumb's IP) represent a grand total of 20 joint per hand. They totalize 18 degrees of freedom, including the wrist's supination/pronation degree of freedom which is technically part of the elbow, and excluding the five direct correlations between PIP/DIP or IP/MCP developed earlier.

Theoretically, it would be advantageous to obtain those 18 degrees of freedom as input data to our recognition pipeline, because more information often leads to better recognition

Joint		DOFs
Wrist	1	1×3 *
Index → pinky	CMC × 4	4×0
	MCP × 4	4×2
	PIP × 4	4×1
	DIP × 4	~~4×1~~ **
Thumb	CMC × 1	1×2
	MCP × 1	1×1
	IP × 1	~~1×1~~ ***
Total	20 joints	18 DOFs

Table 3.1: The hand's joints and degrees of freedom. * If including pronation/supination, which is in fact an arm joint with 2 DOFs. ** Removed because correlated with PIP. *** Removed because correlated with MCP.

rates by machine learning algorithms, Although we acknowledge the "curse of dimensionality" problem [18], here it seems reasonable to consider that no joint will overwhelm the gesture recognition system with useless input data.

Therefore, the question arises: of all these degrees of freedom, which are to be prioritized, and which can be safely discarded? By answering this question, we will have a better understanding of those sensors that are truly required for our hand recognition application to work. We will also be able to diminish the cost of the glove by reducing the number of sensors, and finally, not surcharge the operator with too many of them.

Unsurprisingly, it appears that most of those degrees of freedom are not used totally independently, but rather in a highly correlated fashion [141]. When flexing a finger, we humans generally tend to bend it entirely and we manoeuver MCP and PIP joints at the same time; fingers in position such as ASL (American Sign Language) sign for letter X (Figure 3.2), where MCPs are open but PIP and DIP are bent, do not happen frequently.

Figure 3.2: ASL letter X demonstrates an unlikely position where MCP is straight but PIP and DIP are bent.

In other words, we can detect most interesting positions – and conversely, express much information with those positions – with a single sensor on each finger. That is because "closing a finger" makes sense whether the closing happens at MCP or PIP, and most of the time it happens at the same time [141]. A further consideration is the one of abduction and adduction of fingers, i.e. spreading them apart and gathering them back: while it serves to differentiate some gestures, for example ASL letters U and V (Figure 3.3), it turns out to be

difficult to sense on a glove (the Cyber Glove being the only glove to do this).

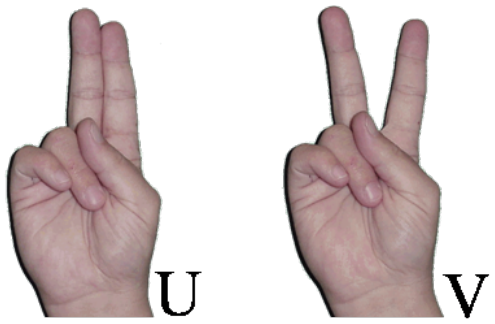


Figure 3.3: ASL letters U and V. A glove cannot distinguish them without a sensor dedicated to abduction/adduction.

Consequently, we will focus on the specific five degrees of freedom regarding finger flexion, and we will not care whether the flexion comes from MCP or PIP/DIP since they are almost always done together.

3.2 Sensors

The main task of this thesis being the design of a gesture recognition system from the ground up, including proper choice of hardware, it seems necessary to discuss the sensors we can choose from. This is a good thing: by controlling the hardware, we can build the hand tracking device the most suited to our needs. This means we have more freedom among the complete toolchain. Discussion of which sensors we include in the glove are delayed to Chapter 5, after we have examined what kind of gestures are the best for our application inChapter 4. Only then, we will be able to choose the best intersection between appropriate gestures and available sensors.

3.2.1 Cameras

One of the most common ways to recognize gestures nowadays happens with camera devices [37]. Thanks to recent advances in image processing, it has become commonplace to track a user's skeleton, including if desired the hand's skeleton. We distinguish here two main kinds of camera systems: classic RGB cameras and depth sensors.

3.2.1.1 Common Cameras

An RGB (Red Green Blue) camera is a very common image capture device. It represents images as matrices of pixels containing 3 channel color information: red, green and blue, three primary colors from which any color can be rendered on a computer screen or mobile device. Sometimes, a computer vision processing system will be only given access to the greyscale version of the image, thereby reducing the amount of information to a single channel averaged from all three original ones. Because of their versatility and their mimicking of human eyes in their ability to see gestures, previous research has naturally tackled the problem of gesture detection using regular cameras.

A common issue resides in the difficulty to go from a matrix of pixels to a representation of the hand that can be further processed. The first step requires segmentation of hand by image processing techniques, for which various techniques to find the hand in camera images have been developed, such as motion descriptors [103], skin detection [40, 228], contour analysis and tracking [177] or motion energy [248]. Hand segmentation is further discussed by Zabulis et al. [252] and Rautaray et al. [179]. It is possible to use colorful fabrics to enhance detection by a computer camera [210]. Other research also tackles the problem of whole-body gesture detection through cameras [23, 70].

A common camera brings the main benefit of being readily available and easy to integrate in most scenarii, at least indoors. They are cheap and omnipresent, all the more since the event of mobile devices which are virtually all equipped with one or multiple cameras. Notwithstanding our specific application, passing from the matrix of pixels representation of a picture/video to a suitable representation of the hand may be tricky, especially in situations such as ours, where training examples can be very sparse. Lastly, we will finish by considering outdoors requirements from our application: a camera is subject to light changes, but we wish our system be usable at night; in particular, all approaches described above for for hand segmentation, including motion- and color-based ones, are too fragile to work with high acccuracy in military environments (forests, desert) and in the dark. Finally, camera placement around the robot is a tricky subject that could be seen as unsolvable: how many cameras? Where should they be placed to avoid occlusion?

Most of these issues will be applicable for depth sensors as well, which we discuss below.

3.2.1.2 Depth sensors

A recent addition to the plain RGB camera is fueled by the desire to obtain third-axis information about the image. Depth sensors, also called depth cameras or RBGD cameras, will capture how far each pixel is from the camera, leading to a 3D representation of the picture. Here, we mention two main depth sensors are mentioned because of their popularity, especially in a gesture recogntion context [37].

The first is the Microsoft Kinect [255], which acts as a regular RGB camera also returning depth information, furthermore, a powerful API makes it possible to obtain proessed images containing skeleton information, for example. This is made possible thanks to a combination of depth data and machine learning techniques [116] to spot skeleton keypoints and locate them in the 3D space of the camera. Though the Kinect was designed for video game purposes, is has since widened to encompass other fields, one of which is gesture recognition [37]. Depth data is obtained by emitting infrared rays in all directions in the field of view of the camera.

The biggest advantage of the Kinect is the ready-made API to obtain skeleton information. Though not always perfect, it is often sufficient to create innovating applications on top of it [7, 64, 167, 183, 202, 218, 244]. On the other hand, drawbacks not ony include most of the burdens brought by the common camera (see above), especially regarding the questionable placement of camera(s) around the robot for hand tracking in our precise application. Another downside lies in the impossibility to track IR dots outdoors, because the sun emits IR on its own [229]. All in all, the Kinect was designed for a living room environment, not so much for outdoors applications.

It is worth mentioning a second very popular depth sensor primarily used for gesture recognition: the Leap Motion [119]. This device is designed for desktop use; the goal is primarily accurate detection of the hand posture (finger state) and orientation. In this case, the user just places their hand above a small box containing the remote sensors. Since it is specialized in hand tracking, and also because of a much better image than the Kinect (mainly due to close proximity of the hand and less background noise), it is able to recover the hand's skeleton in detail and is capable of detecting precise hand movements [240]. The API offers the user a fine-grained control over applications such as virtual aircraft flying or manipulation of 3D objects [172]. Many applications are built on top of the Leap Motion, including sign language recognition [171, 172] and virtual manipulation [180].

Finally, Microsoft used a depth sensor in a rather unusual way, by placing it below the wrist in Project Digits [113]. The sensor applies inverse kinematics to recover hand posture from input data indicating the distance to each finger's intersection with a sensor-projected plane. Therefore, it can recovers the full 3D pose of the user's hand without requiring any external sensing infrastructure or covering the hand itself. The system targets mobile settings and is specifically designed to be low-power and easily reproducible using only off-the-shelf hardware [142].

3.2.2 Finger sensors

As we have seen above during the presentation of previous data gloves, there exists several kinds of sensors that can report information on the state of fingers. The most useful information is flexion, followed by abduction/adduction (spreading fingers or gathering them back). In the following, we will mainly focus on flexion, because not many sensors have the ability to recover the abduction/adduction degrees of freedom.

3.2.2.1 Resistive ink flexion sensors

Resistive ink sensors are thin pieces of plastic whose resistance change when bent. They are very affordable [71] and multiple brands are available, including Spectra Symbol [207] and Tactilus [195]. Their interest in the context of gesture recognition stems from their size (in general, finger-sized sensors are widely manufactured) and their ability to go through a important flexions for a high number of cycles; they are designed to be bent up to at least 180° or more [206]. In general, they will undergo serious bending, however it is best if the flexion is somewhat uniformly distributed throughout the sensor.

The technology leverages the behaviour of sprayed conductive ink on top of a rigid but bendable piece of material, upon which microscopic stress fractures are purposedly introduced between conductive zones: during flexion, ink micro-pigments get further apart from one another, leading to less current conduction, and therefore increased resistance [185, 195].

They have two main weak points: the first is a small part the base of the sensor, which is not bendable (it breaks upon torsion) and needs to be protected by adjunction of a solid part to avoid rupture of the electronics; the second is that a very localized bending (i.e. decreasing a point's curvature radius too much) might lead to permanent damage and altered reading of the sensor. Nevertheless, it is easy to avoid both problems by adequately protecting the sensor, and also not aggressively folding the sensor onto itself, which is unlikely not occur

during regular usage as finger flexion are well within the realm of a reasonable curvature radius.

The change in resistance is very noticeable: a typical sensor can go from 10 kΩ when straight to 20 kΩ when bent. This can be read thanks to a simple voltage divider circuit [208], converting the resistance into voltage information which can be then received through a common electrical interface known as ADC (Analog to Digital Converter). The input will be digitized, and depending on the values of the reference resistor in the voltage divider, the range of sensor resistance values may span less than the whole ADC range (it will be contained in a suboptimal window of the ADC range). In practice, though, this is hardly an issue, and the combination of voltage divider + ADC works very well for our purposes. Finally, it is worth noting that most resistive ink flex sensor have an electrical drawback known as "overshoot", a phenomenon during which rapid flexion of the sensor makes a small non-lasting peak in resistance value [199]. We will further discuss overshoot in Chapter 7.

3.2.2.2 Stretch sensors

There are several families of stretch sensors, distinguished by their electrical category: resistive and capacitive. The former reports finger flexion thanks to resistance change. The latter undergoes changes in capacity, which requires a dedicated measurement circuit for measurement; it is more complicated to set up from scratch and especially for low capacity values which can interfere with common circuits. In any case, both operate on the premise that the circumference around the finger stretches significantly while it is bent.

3.2.2.2.1 Resistive stretch sensors

A simple, resistance-based stretch sensor from Images SI [91] benefits from a simplicity of electronic implementation (plugging the sensors into an ADC through a voltage divider, like resistive ink flex sensors described above) but is somewhat awkward to fix on the glove. Moreover, the stretching of the device can be felt by the user; and last, because of its thinness, it bears the risk of slipping on the side of the finger if not properly fixed.

3.2.2.2.2 Capacitive stretch sensor: StretchSense

A more advanced kind of stretch sensor is build by StretchSense [215]. This sensor is a capacity stretch sensor; itt takes the form of an elastic stripe which can be easily placed on top of a finger. We evaluated one such sensor with the evaluation kit.

During testing, the sensor reported quite accurate measurements, and we noticed no perceivable sign of overshoot. For these reasons, it is a great contender against resistive ink flexion sensors described above, during the early phase of this thesis when we decided upon which flex sensor to use. One main drawback to this sensor is electronic integration (requires a dedicated circuit for capacity measurement) and price (around $700 per sensor [214], around 50 times pricier than resistive ink sensors. Lastly, we thought durability would be better for this sensor, as the the enclosure seemed able to endure rough shocks due to its soft material; unfortunately, after the elastic outer material degraded and a small dent appeared on the side. This quickly led to complete breakage of the sensor afterwards: it got

cut in the middle at the postion of the bent. However, it seems like other fabric options are available and could mitigate the problem [214].

3.2.2.3 Optical fibers

Optical fibers are devices that transmit light (electromagnetic waves in the visible spectrum) through a piece of bendable material. Use of optical fibers for fingers can take place in two flavours. The first one, rather brittle, leverages reduced ability for light to go through the fiber under bending, and is used in the Sayre Glove [49, 217], one of the earliest data gloves and also used for human-robot interaction [66]. Unfortunately, it quickly leads to sensor fatigue and low reproducibility, therefore is best avoided altogether.

A second family of techniques is more complex and uses properties of high frequency electromagnetic waves in optical material in order to obtain stretch measurements. These devices, called Fiber Optic Sensors, can be divided in three categories [124]: local fiber optic sensors, which detect optical phase change induced in the light along the fiber; FBGs (Fiber Bragg Grating) which are altered in the middle and whose strain can be detected through wavelength changes via an optical interrogator; and distributed fiber optic sensors, which undergo light intensity variation inside the fiber.

They are mostly used in civil engineering for reporting microscopic stretching of heavy structures such as bridges or to monitor traffic [124]. Unfortunately, the required equipment to interrogate the sensor is too cumbersome to be integrated in a glove. Furthermore, typical strains are orders of magnitude smaller than typical finger stretch While we decided to look for miniaturized options, it quickly became apparent that such optical fiber setup would be an excessive effort, offering no real advantage over resistive ink sensors while incurring costs and questioning feasability at all. Such sensors seem to fulfill a real need for stretch measurements in their specific domain, but the intersection with ours seems too small.

3.2.2.4 EMG sensors

In the same way the well-known EEG (Electroencephalogram) medical device captures electrical signals within the brain, its close cousin EMG (Electromyogram) aims to capture electrical signal. In this case, the obtained signal provides data to recover finger flexion. This is due to the fact that most of the muscular activity leading to finger flexion and extension takes place in the forearm. In medical settings, precise EMGs can be performed by introducing needles in the skin, but often there is enough information by simply placing electrodes on top of the skin, a technique known as *surface EMG*.

It has been shown it is possible to run machine learning techniques for finger extension and flexion detection, based solely on surface EMG signals [134], and EMGs have indeed been used in gesture recognition contexts [32, 33, 126, 221].

The most popular commercially available device for surface EMG gesture recognition is the Myo [222], a polished bracelet designed to be worn on the forearm. It comes preprogrammed with a bank of gestures [223]. The main advantage of such a setup is how the hand is truly left free, because no glove has to be worn; only the forearm bracelet; although, our findings indicate some users might experience slight discomfort during long sessions.

Drawbacks include, in our experience, the difficulty to make sense of the finger patterns from the raw EMG data. Indeed, the Myo low-level API (not the SDK, which limits which

gestures can be learnt) offers 8 electrical signals, which refer to 8 electrical signals of the forearm and do not directly to the 5 fingers. Moreover, muscular activity reports muscle tension as well, which disrupts hand posture readings depending on whether the user is flexing fingers with or without strength.

3.2.3 Motion sensors

While previous subsections discussed sensors to obtain the hand posture, we have not discussed ways to obtain data regarding orientation on the motion of the hand. As we shall discuss later (Chapter 4), hand posture and motion consist of the most important ways to convey information. In this subsection, we not delve into the topic of motion sensing systems requiring static installments such as motion trackers [236], camera-based pose-recovery algorithms [9, 37] or marker-based camera systems [61] since they are not applicable to the outdoors nature of our work.

Rather, we will primarily discuss one technology of motion sensing: the IMU (Inertial Measurement Unit), which is able to give us numerical measurements on the motion and the orientation of the sensor.

3.2.3.1 IMU

IMUs, or Inertial Measurement Units, are small chips comprised of several independent motion sensors. They make it possible to obtain different kinds of data through three main sensors:

- accelerometers: report gravity and linear acceleration
- gyroscopes: report speed of rotation
- magnetometers: report the North direction

They come in different packages and are used throughout different applications like robotics, UAVs, mobile phones or gesture recognition applications [8, 19, 20, 28, 80, 126, 138, 139, 176, 183, 211, 226, 230, 238]. In our case, we settled on a miniature IMU well suited to be placed on the back of the hand: the MPU-6000 / MPU-6050 family [92]. They use the now-common MEMS technology (Micro Electro Mechanical Sensors) in order to make mechanical systems small enough to fit in a tiny chip. Another IMU manufacturer, Xsens, created IMUs that were used in robotics and gesture recognition [138]; earlier products had a bigger form factor than the based MPU-6000, which is problematic in terms of real-world glove integration, but now there are also small size IMUs using MEMS technology, as long as one can design and manufacture an electronic board to collect IMU data. Consumer-ready electronic board for MPU-6000 processing are readily available on the market [205].

In most gloves with motion sensing, there is one IMU placed on the back of the hand, because this place is a good notion of what we think as the hand's reference frame (independent of finger flexion, in particular) only while at the same time not impeding movements; in general an IMU on the back of the hand is barely felt if at all by the user.

The most obvious asset of an IMU is the possibility to obtain many motion measurements, in the first place, which is a tremendous information in the context of gesture recognition. Additional advantages include the small size factor (4mm x 4mm x 0.9mm for the

chip alone) [92], small electrical consumption [92], easy electronic integration (numerous ready-made available boards [205] including microprocessor programming [204]). The main drawbacks are centered around sensor inaccuracies: gyroscopes and accelerometers suffer from biases and noise in particular; also, thorough processing of those sensors to obtain additional features (orientation, linear "pure" acceleration) is not an easy task as it requires correction schemes without which divergences appear in matter of seconds; this also explains why an IMU cannot report first-order physical quantities such as position, a topic we discuss in Chapter 7. Nevertheless an IMU is an excellent tool for our job and well-suited to all the constraints of outdoors requirements described in Chapter 2.

3.3 Data gloves

A "data glove", or "wired glove", "electronic glove", etc. is a device that is supposed to be controlled by the hand, by means of wearing a complete piece of cloth over it. Most of the time, a data glove will contain some electronics. The goal of a data glove is to retrieve enough biomechanical variables in order to describe the position of the hand. This is usually done by placing different sensors on top of, or inside the glove. Those sensors will generally relate hand posture, motion, etc. into physical properties, such as resistance, capacitance. Ultimately, those variables are encoded digitally in order to be computer-friendly and helpful for later stages of the recognition pipeline.

3.3.1 Sayre Glove

One of the earliest realizations which qualifies as data glove is the Sayre Glove, developed in 1977 at the University of Illinois at Chicago [45]. The device used a technology that seems rudimentary nowadays: light travelling from fixed sources into flexible tubes are received at the other ends of the tube with an attenuation positively correlated with the finger flexion. In other words: a straight tube will let pass all light from the source, but as the tube is flexed, lesser light will reach the receiving end. No position or motion capture system was integrated. They did not use their invention to make a full-fledged gesture recognition device, but only as a multiple-slider control, one for each finger. [49, 217]

3.3.2 Power Glove

In 1989, Nintendo officially licensed a device which would allow game console players to interact in novel ways with their environment: the Power Glove. [217] It was not actually built by Nintendo, but by Mattel instead, a famous toy manufacturing company. The glove was mostly used for some specially tailored games on Nintendo's then-leading platform NES.

The Power Glove features conductive ink coated flex sensors for all fingers except the little one, which already gives a good hand posture information, albeit incomplete. The little finger was likely the wiser to exclude, as it often follows the ring finger position, except for some very specific positions of the hand where the little finger has a special meaning, such as "fist closed except the little finger". Position in space is found out via a rather clever system: two speakers are positioned on each side of the TV monitor and emit ultrasonic sounds in order to be transparent to the human ear. The glove measures the time taken for sounds

to travel from the receiver, and uses triangulation to determinate position and two angular variables. A more recent device, the Wiimote (2006) operates on a similar principle, except it uses a cheap camera to detect two sources of infrared light [64]. Anyway, the philosophy behind the Power Glove was to have an inexpensive device that could be easily manufactured and affordable by customers, while providing an original gaming experience. In this regard, one could assert it did well on those fronts, in spite of a limited commercial success: only two NES games including first-class use of the Power Glove were ever released.

3.3.3 Cyber Glove

Company Cyber Glove Systems designed a wired glove whose goal is to capture the hand's posture with a high degree of accuracy. The older version, the Cyber Glove II, contains up to 22 flex sensors: 3 flexion sensors per finger, 4 inter-finger sensors, 2 sensors for wrist orientation and a final sensor on the back of the hand [234]; these are thin foil strain gauges sewn into the fabric. Regarding communication, the glove operates on a wireless protocol, and it supports plugging in an additional device to compute hand orientation, such as an IMU board. We own a CyberGlove at IRISA, acquired for previous experiments and projects not related to this thesis [104, 105]; after discussion with authors, the main feedback was that the glove was comfortable and returned reliable sensor measurements, but was a bit fragile and not resistant to prolonged usage. In fact, our glove has a couple of sensors that ended up being broken during operation. On the flip side, since the Cyber Glove incorporates several sensors per finger, there is enough redundancy between hand joints [89] to reconstruct the signal for some of the broken sensors.

The Cyber Glove III was developed to improve over the previous version, but the areas of improvements are not of much interest to our specific study; while keeping an accurate measure of the hand posture, it also features ways to interact easily with a bigger motion capture system, as well as battery enhancements, more storage options, etc [43]. This follows a general trend in the area of human tracking: frequently, users such as movie studios, dancers or live performance artists wish to track not only the hand, but the whole body [23, 70]. Consequently, they require a full-blown body capture system, of which hand tracking is only a part. Finally, we should note that the Cyber Gloves are very expensive [199] compared to similar gloves.

3.3.4 P5 Glove

As much as a joystick can be used as a game controller, a wired glove can also be used for gaming purposes. The P5 glove [143], introduced in the beginning of the 2000s, was claimed to be at the time an innovative device able to control a player's virtual environment, such as grasping a sword or hitting with one's fist. It is not exactly a glove, since there is no piece of cloth; rather, it is a simple piece of plastic sitting on top of the hand, and maintains five flex sensors aligned with the finger.

The glove supports hand tracking in all three linear and all three angular axes, as the Cyber Glove II, which is achieved thanks to an optical tracking technology. Also, because this glove is targeted at desktop PC users, it also supports a mouse compatibilty mode, in which the user becomes able to control the on-screen cursor.

3.3.5 5th Glove

Manufactured by 5th Digit Technologies, also known as 5DT, the 5th Glove [60] is an interesting piece of wearable technology. It comes in two flavours: the 5-sensor glove measures only one flexion per sensor, whereas the 14-sensor version measures two flexions per finger, as well as the opening angle between fingers known as abduction.

It is possible to use the glove with a wire using USB or possibly a serial protocol, but more interestingly, there is an additional option to use it wirelessly. Consequently, users gain the ability to move almost freely in their environment, provided they stay within adequate wireless range. Furthermore, like many modern wired gloves, the 5DT is equipped with a lycra fabric known for its one-size-fits-all property and sufficient comfort.

Price is quite affordable for such a glove, as the 5-finger version is below a thousand US dollars. It is sure expensive for consumers, but a reasonable industrial application such as ours should be in line with such a budget, considering the glove is expected to be a one-time cost, which is hopefully amortized over time.

Early gloves by 5DT have long been used in gesture recognition research, such as work by Van Vaerenbergh et al. to recognize finger-thumb opposition gestures [232], recognition of forged signatures by Tolba et al. [225] or Huang et al. for gesture clustering [88].

3.3.6 Peregrine Glove

We stay in the entertainment industry with a modern gaming device: the Peregrine Glove [94]. The main target is desktop game players; visually, it displays a nice high-tech look, while on the inside it incoporates multiple sensors allowing the user to interact in novel ways with a computer. On top of the glove, several thin metallic wires are sewn among the fabric. It is designed so the user can touch one finger with another, say, thumb touching index for instance; with a simple electrical signal detection, the glove is able to tell that two fingers are touching. This is then mapped to an arbitrary keystroke on the computer, making it possible to control a game environment with hand positions. On the whole hand, one finds as many as 38 zones known as "touch points" [93], in order to create a substantial number of hand posture combinations including several zones per finger.

While the Peregrine can be an appealing device to desktop users, a few design choices make it problematic for outdoor use, which will be our main use case. First, the glove exposes metallic fabric, which are not resistant to varying weather constraints like those we find in military settings – or even everyday outdoor scenarii, for that matter. For example, if it starts to rain, even slightly, the fully exposed metallic wires create an obvious risk of short-circuit between parts of the glove, rendering its use completely unreliable and possibly even hazardous. This is definitely not acceptable for our use case. The second issue is the lack of a motion tracking system such as an IMU, which is important for our gesture recognition application. Nevertheless, it provides a good inspiration, as it is one of the rare gloves incorporating the idea of sensing contact between fingers.

3.3.7 AcceleGlove

Instead of using common finger flexions, Anthrotronix's AcceleGlove [4] bets everything on IMUs. Indeed, not only one 3-axis accelerometer is present on the back of the hand, but also

five identical accelerometers are placed on the end of each finger, near the fingernail [3]. This way, by detecting the orientation of each finger given the accelerometer, one should be able to compute the flexion of each finger. This clever technique makes it possible to know finger flexion, without resorting to flex sensors, which span the whole top area of the finger.

We own an AcceleGlove since 2013 at Thales robotics lab. It served for very early testing regarding gesture recognition technology at the beginning of this thesis. During tests, we realized that while accelerometers reporting gravity lead to good posture recognition when static, they will unfortunatey rule out certain finger positions. Indeed, an accelerometer alone cannot report a complete description of its orientation; when static, it will only report the "up" vector, or the "gravity" vector – both are equivalent depending on the chosen sign convention. The problem can be described as follows: imagine someone closing their hand as if they were very angry and banged their fist on the table; let us call it "Posture A". Now, consider them keeping their hand in the same position against the table, but opening fingers, so that their hand is just a plane orthogonal to the table; let us call this "Posture B". Because the gravity is seen the same from the accelerometers in both positions, one will be unable to tell the difference between Posture A and Posture B. For this reason, nowadays complete IMUs integrate magnetic and rotational sensors to indicate a second vector of reference and provide more inertial data; if an hypothetical AcceleGlove v2 integrated such additional sensors as well, the problem should disappear.

In this thesis, the results of our experience with the AcceleGlove led to the following insights. First, an IMU is a good idea for knowing the orientation on the back of the hand, because it can give information about the relative orientation of the hand, but should be completed at least with gyroscopes to complete this information. On the other hand, we discovered that accelerometers-only are not reliable for telling the hand posture, especially finger flexion in some positions, as highlighted by the previous paragraph. Finger flex sensors seem to be a more robust and less complex alternative.

3.3.8 VHand

Manufactured by DGTech, the VHand 3.0 is an instrumented glove intended for research and motion capture purposes, containing classic sensors to describe the hand posture and motion. On the outside, a black one-side-fits all lycra glove covers the whole hand. Then, five flex sensors spread on each finger detect whether they are flexed or open. Motion is sensed thanks to a low-cost IMU, installed on a small box on the back of the hand. An embedded CPU captures all this data and is able to pre-process some of the data as well as transmit it to a receiving computer. The communication takes place either in a wired fashion over USB, or over the air using a compliant WiFi chip. Additionally, the glove can be fully unplugged, since it includes a small Li-Po battery, giving a few hours of free movement use without having to wire the glove for operation. The IMU reports 9 degrees of freedom, that is, one 3-axis accelerometer, one 3-axis gyrometer and one 3-axis magnetometer. Last but not least, the price is not very expensive for a glove of this type: around $1000 for the WiFi version. Of course, it is definitely expensive to a customer, but it is perfectly acceptable for research purposes such as ours.

3.3.9 Goniometric gloves

Data gloves and flex sensors have also been studied in the medical field. In this setting, the goal is not so much proper gesture recognition but rather obtention of precise angular measurements of hand joints. Such a glove is called in the literature a goniometric glove, from greek gonia meaning angle. In 1990 Wise et al. used an early data glove, aptly named the DataGlove, to investigate fiber optic sensors in the context of semi-automated joint measurement [241]. In 2003, Dipietro et al. used a HumanGlove to monitor effictiveness of rehabilitative treatments [50]. Simone and Kamper continued on such ideas to measure bend flexion in impaired patients [199] and monitored 25 hours of continuous hand daily life activity to evaluate compliance of patients with physical therapy and home rehabilitation instructions [200, 201]. A recurring thematic is the question regarding repeatability of angular joint measurements, which is evaluated thanks to a protocol devised by Wise et al. in which cycles of holding a mold and flattening the hand is repeated several times [241]. Reported typical overall errors range below 6° [71, 200], a very encouraging result if flex sensors are to be used in a recognition setting such as ours. Thorough analyses of flex sensors themselves were also carried out by Saggio [184, 186] and Orengo [161].

3.4 Machine learning techniques

It is possible to use a myriad of machine learning algorithms in gesture recognition. In general, the best algorithm depends on several factors, such as the desired accuracy, but also whether the system should work in an isolated or continuous fashion, which sensors are used, the kind of input data, the size of the training database as well as the amount of computing power available. In this section, we describe the possible algorithms for gesture recognition along with applications in which they were used.

3.4.1 HMM

An HMM (Hidden Markov Model) is a statistical model describing the behaviour of a system operating as a Markov Model [175]; unlike the output of the model which is observable, states are hidden. They have been first discovered by Stratonovitch et al. in 1960 [213] and then further described by Baum et al. in the second half of the same decade [14]. HMMs work on a finite set of states, which means a quantization technique is needed in order to map the continuous values into discrete states, such as k-means or SAX [72].

HMMs map well to temporal processes due to the natural correlation between one output and the following. Therefore, they have been used in multiple domains such as speech recognition [11, 99] and bioinformatics, especially for DNA sequence analysis [22, 56]. They have numerous applications in gesture recognition too. In 1994, Yang et al. [245] classified 9 gesture classes with positional data (X/Y) in both isolated and continuous experiments. In 2008, Elzemain et al. [57] used around 50 video sequences to detect and classify gestures in a continuous fashion from stereo camera images; for segmentation, they used a simple blank label technique (“zero codeword”) represented by the static motion. While this would not map well in our high-mobility scenario, it is an interesting starting point. HMMs can also be used to connect the dots in situations where a grammar is required, such as the usage in 2010, of a neural network by Pizzolato et al. [170] to recognize Brazilian sign language

LIBRAS letter spellings, the output of which was combined with an HMM to combine letter into words. An humanoid robot was motion-controlled in 2012 by Xu et al. [244] by continuous classification of a 6-word gesture dictionary, while Zhu et al. [256] used HMMs inside an interactive gesture recognizing platform in which the user was given feedback during learning. A Kinect was used in 2013 by Wu et al. [243] to detect a 20-word sign language dictionary amont 5 signers. Beh et al. [16] proposed in 2014 an automated process of segmenting gesture trajectories based on thresholding kinematic features.

3.4.2 Random forests

By combining tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest, one obtains the Random Forest technique [26]. It has been used in real-time gesture recognition by Pugeault et al. in 2011 for ASL (American Sign Language) fingerspelling recognition [174] using Kinect data. Keskin et al. [112] used randomized classification forests in 2012 to create pixel-based classifiers combined by voting, on ASL and ChaLearn [78] dataset. In 2014, Kuznetsova et al. classified feature vectors of ASL gestures obtained by Kinect data using multi-layered random forests. In 2015, random forests were used to classify finger digits acquired by a time-of-flight sensor [98].

3.4.3 CRF

Conditional Random Fields are another kind of statistical modeling technique. With CRFs, classification is not made alone since the model takes into account context and neighboring samples. Like HMMs, CRFs map well to chain-like data and are popular in sequence labeling tasks such as natural language processing or handwriting detection applications. Gestures applications include the following. In 2010, Liu et al. coupled a CRF with a deep neural network to detect eye gaze or head nodding [129] ; Bhuyan et al. used CRFs in 2014 to spot gestures in a stream based on a simple threshold model [21].

3.4.4 k-NN

Nearest neighbor classifiers (k-NN), based on metric properties of the input data space, make it possible to classify in cases when the input is sparse. The classification rule is very simple and behaves well when instances are well separated. The nearest-neighbor rule, 1-NN (also called template matching) is applied in works such as Mokhber et al. [147] in 2009 for everyday action classification with Mahalanobis distance, Yang et al. [246] in 2009 for video hand segmentation, Liu et al. for static posture recognition [131] or Parvini et al. [164] who describe a multi-layered approach ending with a template matching classifier.

3.4.4.1 DTW

A nearest neighbor classification rule needs the existence of a distance function, or dissimilarity function, which does not necessarily need to satisfy metric axioms, unlike methods such as SVM which need a true inner product function to work properly. In time series classification, the most popular method is to use a dynamic programming approach described

in the 1970's by Sakoe and Chiba [189] and then Itakura [96] for speech processing, known nowadays under the name DTW (Dynamic Time Warping). DTW is designed to take into account time elasticity differences between signals, which makes it more appealing than plain Euclidean distance since it finds an optimal time warping match instead of blindly try to align the time series point-by-point.

DTW has been long used outside the speech processing community. Here we describe a few gesture applications of DTW. In 2008, Lichtenauer et al. [127] applied 1-NN / SDTW (a statistical modification of DTW) to Dutch Sign Language recognition and outperformed HMM-based system thanks to more resilience to time elasticity. In 2010, the work of Hartmann et al. [80] used a DTW-based thresholding method after time series prototyping (optimizing the number of instances in a class and possibly finding an optimal representant) thanks to class separability metrics, a technique which was evaluated in an accelerometer-based 9 gestures dictionary among 7 participants. In 2011, Doliotis et al. [51] classified Kinect-obtained hand trajectory time series, and Reyes et al. used with a technique to weight features for online spotting and classification of Kinect gestures [181]. DTW can also be used for clustering, as in the 2011 work of Keskin et al. [112] who generated clusters from isolated gesture trajectories representing digits to enhance recognition by statistical models (HMM, HCRF). Gillian [72] applied DTW to multidimensional time series with a templating method (ND-DTW) for musical gesture time series recognition based on accelerometers data in 2011. Cheng et al. [35] proposed in 2013 to divide the curve of a Kinect hand gesture into "fingerlets", to be either learned or set manually to capture inter-class variations, for further classification with DTW; Kinect data is also used by Arici et al. [6] to optimize weights of the hand joints for DTW-based classification of body gestures. Cheng et al. proposed an online extension of DTW in 2014, WDTW (Windowed DTW), which tackles the stream recognition problem by considering a flexible search window inside the DTW online matrix [36]. In the same vein, ideas by Sakurai, Faloutsos et al. [190] (although not directly gesture-related) led to a very efficient DTW optimization to match a stream against mutiple reference patterns, by leveraging DTW's min operator not only on separated time series but all of them at the same time.

3.4.5 SVM

SVMs (Support Vector Machines) are statistical techniques designed to separate or classify data using linear or non-linear boundaries between multiple classes of points. While the theory is based on inner products, efficient calculation can be obtained thanks to kernel functions without losing convergence unicity properties. Video gesture applications include action recognition [239], classification of video motion descriptors [254], ASL fingerspelling [174], hand orientation recovery [69, 130, 219, 237] and analysis of driving scenes [159].

SVMs are well suited to classify points in metric spaces thanks to typical kernels (linear, gaussian, polynomial, for example), but they are difficult to apply on time series due to the non-positiveness of most distance functions, including DTW. To remedy such issues, specific kernels for time series have been designed such as the GA kernel [42] and KDTW [136]. KDTW has succesfully been applied to isolated gesture recognition applications on Kinect and Vicon sensors [137].

3.4.6 Neural networks

Biology-inspired neural networks are composed of several layers which include multiple units each acting as a logistic regressor. Before the trend of deep learning by the machine learning community, neural networks had already been used by Erenshteyn et al. for ASL classification [58], Van Vaerenbergh et al. [232] for gesture classification, Maurer et al. [138] for recognition of human gestures (using Hopfield networks) or Pizzolato et al. [170] for sign language recognition.

More recently though, hardware acceleration offered by GPUs (Graphics Processing Units) opened new possibilities in neural network architectures, which led to deep learning techniques. Since deep architectures with convolutional layers map well to locally correlated data such as video frames, they have been naturally applied to gesture recognition with camera or depth sensors [12, 13, 100, 128, 148, 152, 155, 169, 242]. Some architectures suit best temporal data such as RNNs (Recurrent Neural Networks) [154] and in particular using a special memory unit pattern known as the LSTM (Long Short-Term Memory), able to remember long-past events thanks to gates whose temporal opening can be learnt by the network [122, 227].

3.4.7 Others

Another temporal-based technique is the FSMs (Finite State Machines), which are used for example to classify temporal signatures based on motion energy [248].

Fuzzy logic [253] is an extension of Boolean logic which make continuous truth values possible. Their use in gesture recognition include health-related applications [15, 187] and human-computer interaction [150]; they are also at the basis of a robot gesture control technique under a specific formalism [2].

Ensemble methods, which consist of aggregating the result of different classifiers, were used to merge clusters for classification of 5DT gestures [88].

Bag of words techniques use in gesture recognition are generally confined to camera data [82, 156], but it appears they can also be used in time series processing [10].

3.5 Applications of gesture recognition

So far, we have mainly described the hardware sensors and software techniques for gesture recognition. Although sensors and machine learning are fascinating, in this subsection we aim to step back and describe the end rather than the means, that is, the applications of such gesture recognition works.

3.5.1 Daily assistance

Generally speaking, an assistive robot is one that gives aid or support to a human user; the encompassing field of assistive robotics largely refers to robots who assist people with physical disabilities through physical interaction. Some assistive robots also assist through non-contact interaction, such as those that interact with convalescent patients in a hospital or senior citizens in a nursing home, and include rehabilitation robots wheelchair, companion,

or educational robots, as well as manipulator arms for disabled people, for schools, hospitals, and homes [59].

Usage of gesture recognition for daily assitance includes helping handwashing mentally ill people [85] and recognzing activities for senior home monitoring [38] and event summarization [140]. Robotics use include teleoperation of robot for assistance of daily tasks [13, 52].

3.5.2 Human-robot interaction

Human–Robot Interaction (HRI) refers to the study of understanding, designing, and evaluating robotic systems for use by or with humans [73]. In particular, gesture control can be seen as a keystone in enabling more proximity between humans and robot and further integrate them in our daily life by decreasing human-machine incomprehension. Naturally then, gesture controlling robots has been the subject of much research at the intersection of computer science and human-computer interaction.

Gestures have been used to control service robots [235, 251] being possibly humanoids [250], via pointing gestures {nickel_visual_2007} or general gestures using a Kinect [247]. Navigating robots also benefit from gesture control [52, 66]. Gesture recognition for daily life assisting robot was also studied [13]. The *Hobbit* robot [62] was created to prevent and detect fall of elderly people by detecting gestures of daily activity. In the automotive industry, assembly robots have been controlled by gesture from a top-view camera [41].

Teleoperation is not a new topic [31]: non-gesture controlled systems include a teleoperation system for humanoid whole body motion with a single joystick [198], a networked virtual reality telerobotic system [117], and optimal motion control for teleoperated surgical robots [67]. It is thus understandable to develop gesture-based robot control applications similar to ours, including that of a humanoid in real-time [244], gesture-based interaction with an autonomous mobile robot [40], robot motion control with restricted motion gestures dictionaries [2, 87, 248], and innovative ideas such as the use of a haptic interface for increasing the user's perception of a mobile robot's workspace [48] and desktop-based teleoperation at a high-level from a supervisory telerobotic control [228].

3.5.3 Virtual manipulation

Virtual manipulation refers to interaction with a simulated environment such as a game of a screen displayed 3D space. For example, it is possible to use gesture control to browse a picture gallery [29], to grasp virtual objects [121] and to manipulate virtual objects [84]. Medical virtual manipulation is an interesting challenge [158] and sees applications such as manipulation of medical imaging from within the sterile operating field [216].

3.5.4 Sign languages

Since they use gesture as a primary medium, sign languages have been the subject of much research in gesture recognition, such as ASL (American Sign Language) [58, 81, 118, 164], LSF (Langue des Signes Française; French Sign Language) [2, 25], GSL (Greek Sign Language) [228], PSL (Portuguese Sign Language) [226], DSL (Dutch Sign Language) [127], JSL

(Japanese Sign Language) [218] and Kana [220], ISL (Italian Sign Language) [169], VSL (Vietnamese Sign Language) [28], or the Brazilian sign language LIBRAS [170, 171].

3.5.5 Other

Other applications include gaming, including a gesture control system to play games in public settings instead of home environments [182] and remote driving with stereo camera based "GestureDriver" [63].

Part II

Obtaining gestures

Chapter 4

Gesture dictionary

Contents

4.1 Why a gesture dictionary?

To maximize the chance of a robot understanding the action we describe through gestures, it is necessary that we devise specific ones made explicitly to be understood. It would be a different challenge altogether to try to make the robot understand gestures without instructing it what those gestures are and what their signification is. In terms of machine learning, it means that we are going to do classification of gestures, that is, in a supervised fashion: some examples will be labelled beforehand, so that the system knows what they look like and what to expect from the gesture stream during operation. This is the most common approach; while unsupervised learning is possible [88] we cannot map a command to the robot if classes are not known beforehand.

Therefore, it is natural for this thesis' work to describe a set of gestures whose purpose is to be mapped directly to robot commands. Since we design a Human-Machine Interface, the need to create gestures that map well to a user's intention is primordial: intuitiveness and ease of performance of such gestures must be taken into account.

The gesture dictionary will consist of the set of gestures that our robot is meant to understand. In this case, we will distinguish between two similar but different sets:

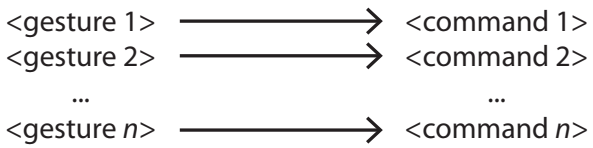


Figure 4.1: A gesture dictionary describes which gestures should trigger a certain command.

- the set of "conceptual" gestures describing ideal, abstract gestures
- the set of "actual" gestures describing real-world performances of said gestures

Analogously, one can say that the letter "A" belongs (only once) to a set of "conceptual letters"; on the other hand, if one writes "A" a hundred times on a piece of paper, those 100 letters belong to "actual letters"; they are all different, but they still correspond to the unique, ideal letter "A".

In machine learning, instead of "conceptual" or "actual" gestures, we would rather adopt the adequate terminology:

- an unique, ideal gesture is said to be a "class"
- a realization of such a gesture is an "example", "instance", or "reference" of this class.

To continue illustrating with our example of letters, "A" would represent a single class over the 26 classes of uppercase letters; and all 100 written "A"s are instances of class "A". Perhaps more practically speaking, an instance would not be the proper trace of ink on the paper, but rather, a computer representation of this letter, such as an image. For example, in the MNIST data set, a very famous machine learning data set, there are ten classes for numbers 0, 1, 2, ... up to 9, and several thousands grayscale 28x28 pixel images representing examples for such classes. This allows the computer to train on different real-world realization of such ideal digits. In our case, gestures will have their own computerized representation as multidimensional time series, as we will discuss later. For the time being, the following discussion of our gesture dictionary will mainly consider gesture classes (the ideal versions) and not so much actual realization of those gestures.

The term "dictionary" is well-suited to the current description. Indeed, in a word dictionary, each word is mapped to a specific meaning (we will ignore the property of human languages which makes it possible for words to be assigned multiple meanings). In a similar fashion, a gesture dictionary is needed in order to map a gesture to a specific meaning – in this case, a command to be interpreted by the robot. The gesture dictionary serves as a reference to the user when they will teach the robot with actual instances of those gesture classes.

A fundamental question in this case is: should we decide the gestures in advance for the user, or should we let the user decide which gestures they want to perform for any given command? To this question, we have considered the following points:

- on the one hand, new users of this system will probably want to know which gestures work well for a given command, so we should provide them with a dictionary;
- on the other hand, after some time users will probably want to expand on this initial dictionary and invent their own gestures to drive the system.

So there is a need for a gesture dictionary, but it must be extendable by the user. Therefore, we have chosen the following approach. At first, the system is provided with an initial gesture dictionary, which is specifically designed for the task at hand (robot motion control) and known to work well thanks to extensive testing during this thesis. Then, new users of the gesture control system will familiarize with those techniques, and later will be able to tweak the gesture dictionary with new classes.

4.2 Previous gesture dictionaries

In the past, many applications have benefited from gestures to communicate intents, orders, commands or other thoughts. Usually, this is done in scenarii where the use of words is either impossible or impractical.

Because our current society focuses much of human communication through oral language and not so much through gestures, our natural habit is to use gestures either as a supplement to spoken language or occasionally to convey very simple ideas such as “hello” or “come here”. A notable exception, of course, is the use of sign languages, which are rightly considered as full-fledged languages in that they succesfully convey ideas as complex as spoken and written languages.

Aside from sign languages, other gesture dictionaries chiefly make use of simple and intuitive gestures with a low degree of grammatical construction, and are used in specific contexts for specialized applications, as we shall see below. Such previously created gesture dictionaries will serve us as inspiration for both the capabilities and expected complexity of such schemes as well as the gestures themselves. In our application, we obviously want to design a gesture dictionary that has a smooth learning curve and conveys a meaning as clear as possible through the simplest and most detectable gestures.

4.2.1 Military

Let us begin with the kind of dictionary the most adapted to our application: military gesture dictionaries. In the ground military, gestures serve to communicate intent from team members to other team members. In general, such gestures will be very simple by nature, since the main purpose is to convey orders in the fastest and most intuitive way. As we shall see, most gestures have a meaning clear enough that each gesture can be learnt in virtually no time, rendering the military forces very apt to use those gestures on the field.

Infantry soldiers on the field have guidelines for specific soundless, gesture-guided communication. Those gestures are very simple and convey much meaning in a minimal form. Interestingly, they are most often performed from a single leader to a small group of subordinates waiting for orders.

The most common gestures in French Military Guidelines **??** are:

- change direction
- push (“appuyer”, i.e. walk close to a wall or range of trees)
- silence
- go forward
- halt/stop

- accelerate
- slow down
- look
- stop firing
- come back to me (“ralliement”)

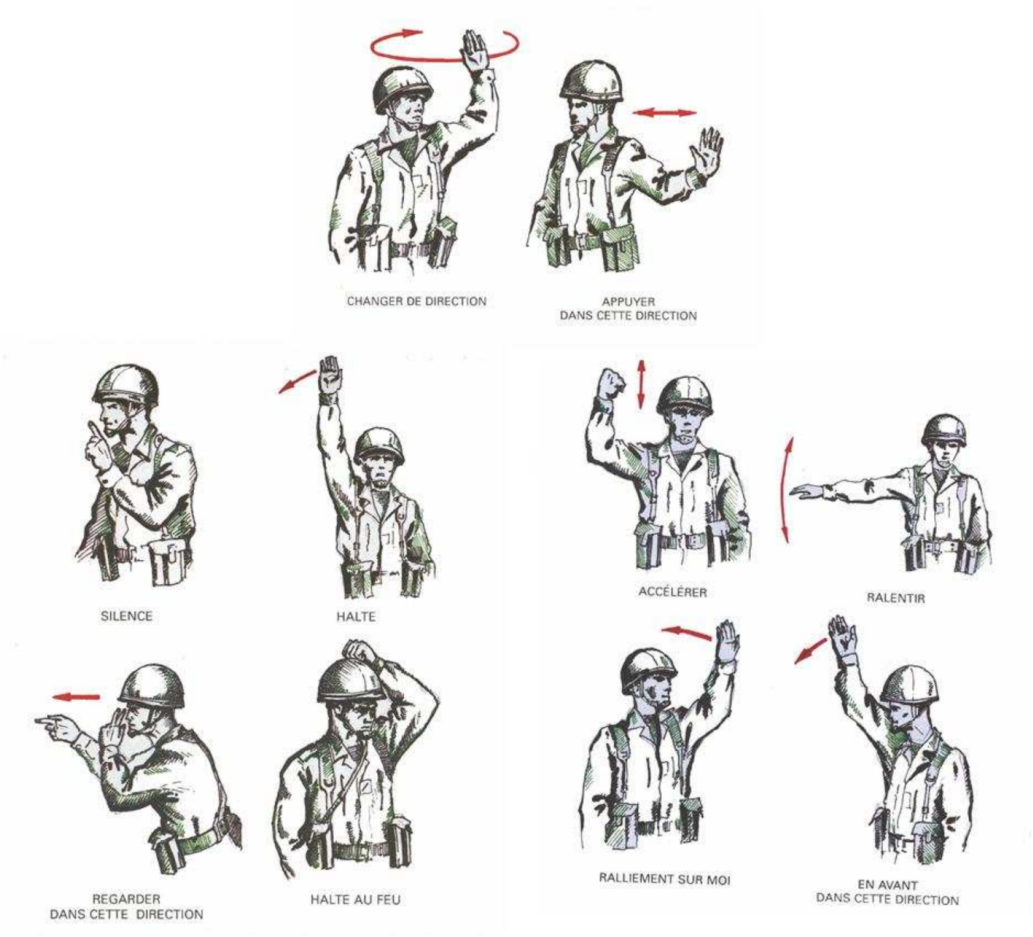


Figure 4.2: Gestures defined by TTA 150 edition 2008 [145].

In an older edition of TTA 150 [144], one also finds additional gestures:

- at attention (“garde-à-vous”)
- repos
- listen
- start firing

Those gestures are an exceptionally good source for our own military-inspired gesture dictionary, since they provide assurance that such gestures are:

1. usable on the battleground
2. known and understood by infantrymen

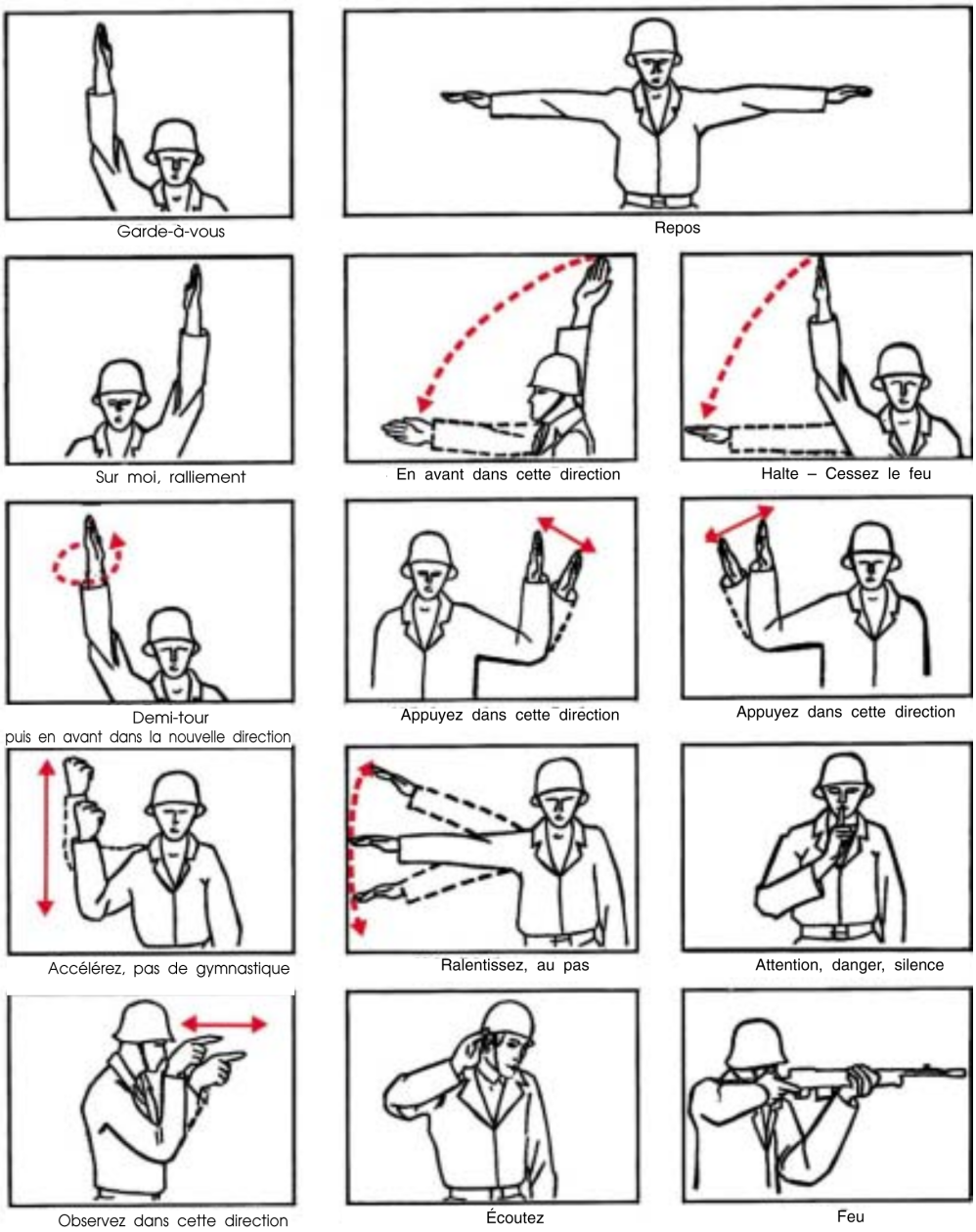


Figure 4.3: Gestures defined by TTA 150 edition 2001 [144].

3. highly specialized, mission-specific commands

Moreover, our interest in motion commands such as starting, stopping, adjusting speed or steering is satisified as most of those commands are already existing in the military vocabulary.

Note that those gestures carry important semantics through:

- finger positions
- dynamics (acceleration, rotation)
- orientation of the hand

This suggests that our gesture recognition sensors and algorithms should be able to detect those semantic carriers through adequate signals. This is an important point, since not being able to detect one of those semantic carriers risks to severely impede recognition.

While designing our gesture dictionary during this thesis, we had to slightly change the meaning of some gestures in order to provide a better experience when it comes to good recognition by the sensors. The reason behind such a change is ultimately linked to the fact that some of those military gestures are not totally adapted to what our sensors will be able to detect. In Chapter 8, it will appear that military-inspired gesture "stop" with an open hand conflicts with "go"; indeed, "go" starts with a raised open hand, so the system struggles to decide if it should detect "stop" or "go". It is therefore our role to adapt such gestures so they get the best accuracy (high detection, low error rates) without changing the form of the gesture too much. Our choice for these changed gestures must stay intuitive for the operator when they are learning our gesture vocabulary.

Special units such as SWAT (Special Weapons And Tactics) also use similar gestures. Usage of soundless communications is fundamental in case the scenarii of immersion requires extreme stealth during operation.

Finally, one might want to note the existence of additional military dictionaries, such as collections of military gestures designed for aircraft control from the ground [231]. Those gestures are less of an inspiration source for us mostly because the context is somewhat different; for example, "hover your aircaft" does not make sense in the context of ground robots. Also, most of the gestures require two arms and are designed for maximum visibility from above the ground rather than intuitive and easy gestures. It provides nonetheless an interesting perspective on the use of gestures in a military context.

4.2.2 Diving

Let us draw inspiration from the civil world for a moment. Here we examine the case of diving, where one generally needs to communicate with other divers for coordination. While underwater, this is achieved by two primary media: voice communication thanks to specific devices, and gesture signalling. Here we will only focus on the latter. Most are easy gestures carrying simple meanings in order to maximize the communication flow underwater.

This dictionary [194] includes topics such as motion commands (stop, come, go down...), air- and health-related discussion, meta-discussion (ok, not ok, repeat, question...) and behaviour suggestions (look, hold hands...). Gesture are, once again, highly intuitive and simple to remember for a newly-taught diver.

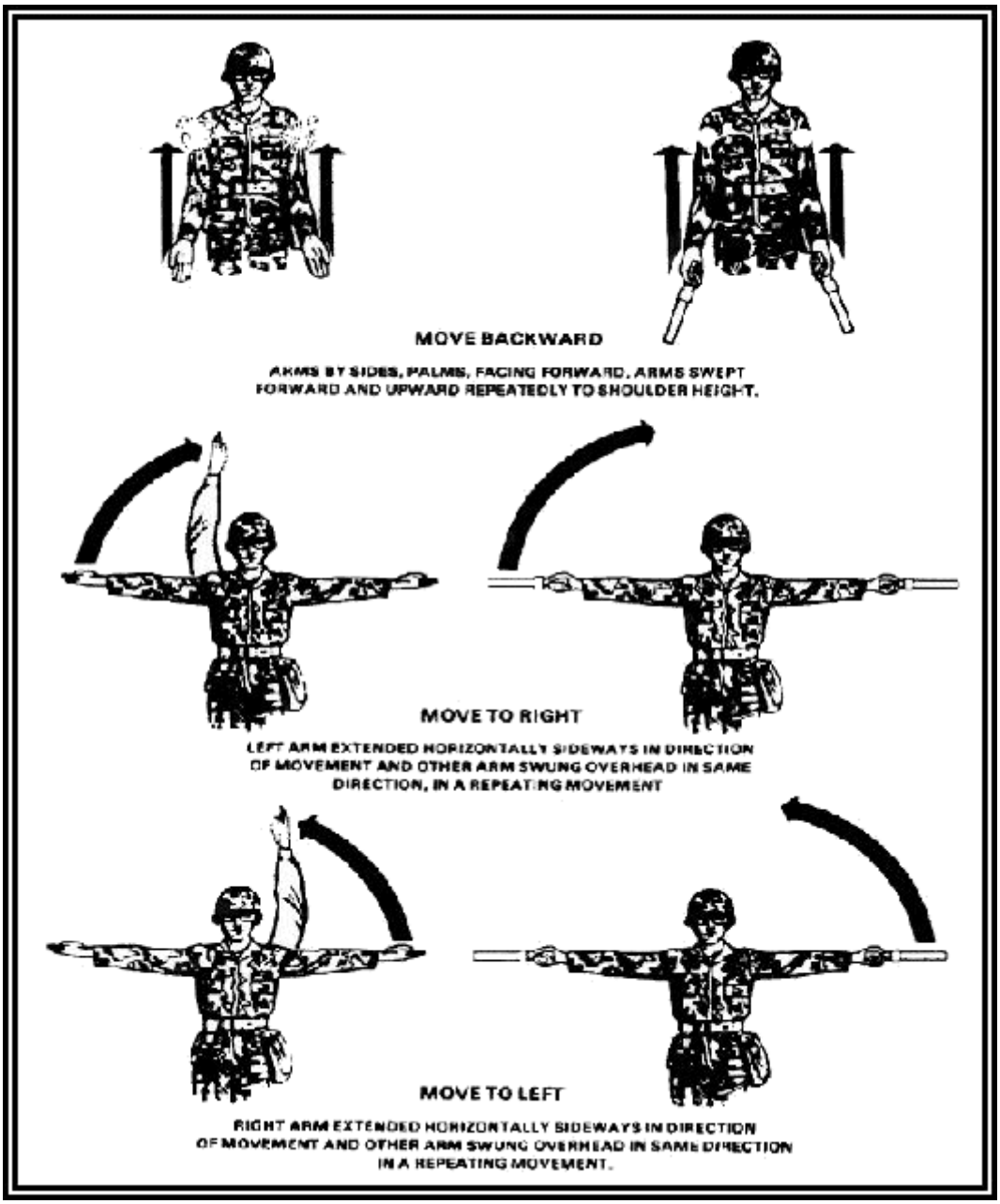


Figure 4.4: An excerpt of Field Manual 3-04.513 Battlefield Recovery and Evacuation of Aircraft [231].

4.2.3 Motorcycling

Use of gestures also takes place amongst motorcycle users. While the most common gesture, a simple greeting carried out with V-opened index and middle fingers, is not very relevant for our discussion, a handful of other gestures designed for group riding shares some similarities with dictionaries discussed above. In any case, those gestures are performed in order to convey meaning from one person to one or several people. In addition, due to the need to hold the handlbars by at least one hand, most of those gestures have to be performed with a single hand – a point that is of much interest to us because our goal to design a single-handed gesture dictionary.

The most interesting hand signals for our case are steering commands (left, right; mostly used as a replacement for non-existent or defective turn signals nowadays); speed directions (speed up, slow down), basic orders (start, stop, follow me).

4.2.4 Cranes

Another example of the need for gesture communication happens in the construction industry, more specifically in the subdomain of crane and crane operators.

The crane controller is the person on top of the crane, generally in a small cabin. They have control over the crane, but it is possible that an engineer or worker down on the ground can have a better view of the task to realize. Therefore, it is necessary for them to given orders to the crane controllers. In general, voice communication is preferred thanks to walkie talkies. However, it may not always be the most appropriated tool for the task of conveying orders from the ground to the crane controller; which is why an operator on the ground can also use gestures to communicate commands up to the crane controllers. It is a way of remote controlling the crane even without being at the real control, thanks to a human intermediary interpreting gesture orders and thereby piloting the crane. Because of the widespread use of hand gestures in crane remote control, a French norm [157] specifies which gestures are to be used on the field; on a global scale, the ISO comittee issued a similar standard [95] to normalize their usage among worldwide workers.

As the main goals of cranes is the displacement of heavy loads, classic orders emitted by a crane operator on the ground will stay in the realm of possible useful movements of the crane, especially: going up and down, rotating around the main crane axis, or translating laterally, as well as meta-discussion (ok, not ok) or emergency situation advisory.

Even if this is not related to gesturing, we can mention that the crane controller on top of the crane can reply back to the ground operator via the use of horn signals. This reinforces the idea that our gesture-based control (the one which is the topic of this PhD, not this crane example) can benefit of any kind of reply from the controlled system such as a sound, a visual clue, or the actual requested action itself if it happens fast enough. Main answers from the crane controller are as follows:

- ok, acknowledge
- please repeat
- danger / distress / SOS
- crane in action, be careful

Even though the design of a reply scheme is not in the scope of this thesis, these replies can still serve as a reference to a future implementer or anyone interested in the "two-way" aspect of a human-robot communication system.

Computerized remote gesture control systems have been developed in the past, mainly using camera systems [149, 168] or touch screens [90]. However unlike such prior works, our system provides an additional confidence in terms of usability for a real crane situation in the building industry. This comes from our initial military constraints, rendering our system well transferable on a building site, where workers:

- have a main task (moving heavy objects) for which the gesture control is just a means, not an end;
- are subject to outdoors conditions;
- need a robust device;
- will not deploy a Kinect or a camera-based system;
- probably already wear gloves anyway, easy to replace their normal glove with a data glove.

4.2.5 Sign languages

One of the most elaborate form of gesture communication, sign languages are distinct from the other categories described before. Their purpose is not to describe a small number of precise, domain specific actions, but rather a largely successful attempt at comprehensive human communications in the same vein than so-called oral languages [212].

Not only one sign language is too complicated to describe here in a few paragraph, let alone the existence of several sign languages for different regions of the world: American Sign Language, French Sign Language, and many others. We will succinctly outline the key takeaways of sign languages for our gesture analysis, with a special focus on the most semantic conveying aspects of these languages.

Most sign languages need a proper way to describe letters. For this purpose, each letter is represented by a specific position of the hand, which for some of them can be slighlty similar or reminiscent of the actual written shape of the said letter. Though letters are not the principal means of communication in sign languages, and rather serves as exceptions when a specific concept lacks a dedicated sign, letters can still be expressed fast enough that spelling an unknown name can be done almost as fast as spelling it orally.

From the way letters are shaped on the hand, we can discuss several interesting facts:

- letters are represented by an unique shape of the hand
- orientation matters: letters should be performed with adequate positioning especially with respect to the "up"/"down" axis
- most letters are performed statically (posture only) but some of them require motion (e.g. a three-stroke movement for letter "z" or single rotations for "p" and "j")

Words and sentences follow a specific grammar, which generally narrates the story being told. Sign languages use far less auxiliary words serving only grammar purposes in oral languages such as articles, auxiliary verbs, etc. As an example, in LSF (Langue des Signes Française, French Sign Language), sentences often follow logical structures such as "agent

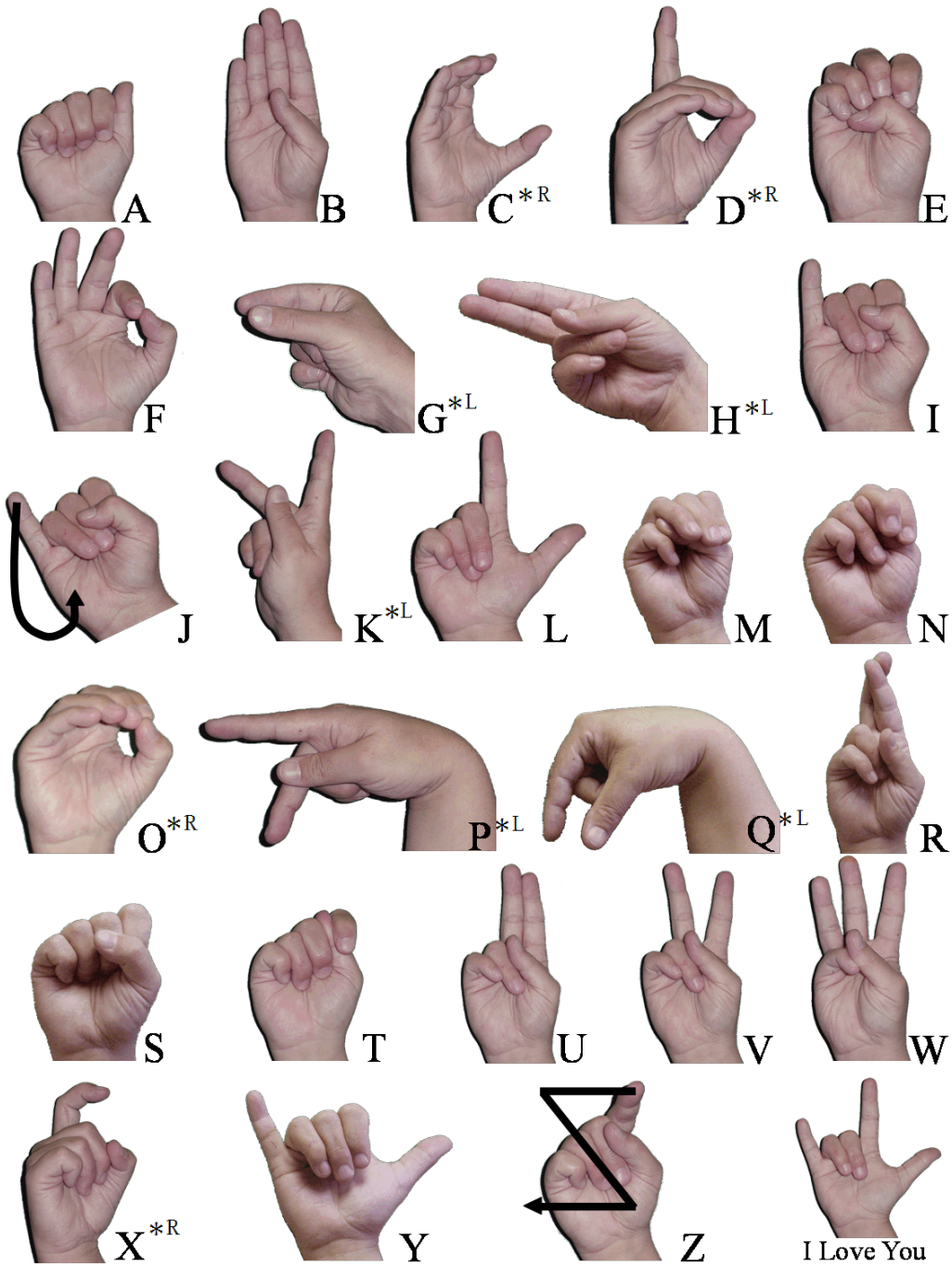


Figure 4.5: ASL letters.

+ element/location + action (verb)" [77]. The core semantic carrier of a successful discourse includes:

- facial expressions
- hand configuration
- location on body or in space
- hand orientation
- motion
- time line (behind body = before, in front of body = after)
- transfers of size, shape

Once again, we find the three main components of a gesture scheme that we identified in earlier gesture dictionaries for systems control: hand configuration (posture), motion (dynamics) and orientation. This encourages us into keeping these semantic carrier and try to detect them with the best accuracy.

At the same time, this list contains many interesting aspects that we could include if we were to design a true sign language recognition system; for example, inclusion of the time line as the behind-front axis could turn out to be a real semantic vector, but it is not strictly necessary to convey such meaning in our application, notwithstanding the challenge posed by detecting the position of the hand relative to the body. It is important to remember that we aim for real-time control of a robotic system with a high degree of precision, and not human-to-human complex communication.

4.3 Designing our gesture dictionary for robot control

After reviewing some of the most used gesture schemes, we can draw conclusions before heading to the design of our own gesture dictionary. Gestures discussed here take place in the context of a human-to-human communication; they are mainly designed to efficiently convey meaning. Similarities through the different gestures dicussed above, even between distinct communities, suggest that they appeal to human senses of motion, pointing, etc. to carry meaning in an universal and efficient manner. Of course, it is not the purpose of this thesis to investigate whether such gestures are truly universal of if they refer to a particular cultural setting. We will simply note that gestures describing similar intents seem to relate between different contexts, such as "slow down" being the same in diving or motorcycling. We shall take advantage of this universality when designing our own efficient dictionary for robot control by reusing such well-understood gestures.

As we have noted throughout our above study of previous gesture dictionaries, some patterns emerge regarding the ways gestures are used and understood by humans. The core pattern is what we have called so far the semantic carriers, that is, the information about the hand which makes it possible for a human to convey meaning. We retain the following semantic carriers:

- posture of the hand, i.e. individual finger flexions
- dynamics of the hand, i.e. motion
- orientation of the hand

In this thesis, we will follow the hypothesis that such descriptors convey enough meaning to build a successful communication of simple orders between two humans, or in our case, between a human and a robot. Therefore we will focus our effort on detecting only those semantic carrier, no more, no less. Not only this choice of descriptors does not come out of the blue, as was shown by the above analysis of gesture dictionaries, but we will see (Chapter 8) that they convey sufficient infomation to be accurately detected and furthermore drive a robotic system in real-time, validating that our choice was good. For the time being, let us not jump to conclusions; we begin by explaning the design of our gesture dictionary below.

4.3.1 Possibilities

4.3.1.1 Static and dynamic gestures

In the previous gesture dictionaries, we have seen that many gestures are mostly well performed and understood when performed without any motion. Such a gesture is said to be *static*, while a gesture performed with motion is called *dynamic* [164]. It turns out that absence of motion carries semantic in itself [57]; one could say that "no motion" is just any other kind of motion, just degenerated into a constant state. As an analogy, in mathematics, one can identify a scalar x with the constant series $(x, x, \ldots)$. This analogy is not as far-fetched as it sounds: later, when we introduce the time series based representation of gestures, it will turn out that static ones are essentially collided into an almost-constant time series. Consequently, we can work with gestures identically regardless of whether they are static or dynamic, thereby killing two birds in one stone by reusing the same time series based recognition technique.

4.3.1.2 Short or long gestures

While designing the gesture dictionary and the recognition algorithm, we would like to avoid forcing the user into fixed-length gestures. Indeed, it is possible that some gestures will take place in a short time, especially static gestures which can be accurately described by a few data points; while other gestures will be performed with some more time. Nevertheless, a gesture dictionary designed for real time control will be likely poor if it proposes a gesture whose total duration is too long, for example above 2 seconds. As a rough estimate, it should be possible to perform the typical gestures as proposed in the previous dictionaries in a fraction of seconds, so our own dictionary should aim for the same.

Hence, we approach the problem as follows. On the one hand, design a gesture dictionary in which gestures are mostly short and to-the-point, not unlike the previous dictionaries discussed above; on the other hand, design our software recognition algorithm so that it is able to detect gestures of any length. This will serve the purpose of not limiting the gesture input space, in case the user wants to create newer gestures, which is an important goal in this thesis.

4.3.1.3 Sustained gestures

Some gestures can either convey meaning on a one-shot basis, with no specified duration, whereas others could specify a sustained intent for as long as they are active. For example, a

gesture "go" indicating the robot to start moving is a one-shot gesture, unlike a gesture indicating "turn left" for which we want sustaining until the intent is over. A low-angle left turn would be realized with a short instance of this gesture while a high-angle turn, or a halfway turn, could be realized by sustaining this gesture until the operator is visually satisfied with the current angle of rotation; then they stop performing the gesture (an operation we call "releasing" the gesture) indicating to the robot the order of stopping rotation.

In order to address this problem, we suggest two possibilities, both of which are not incompatible with each other in the recognition algorithm:

- sustained position: the gesture to detect is static (no motion), and the operator stays in this static posture until they want to stop the action;
- repetitive motion: the gesture is dynamic, and the repetition of such a gesture will indicate to the robot that it should continue executing the corresponding order until the end of that command.

For example, among gestures presented in Section 4.4, there are several sustained gestures such as "left" (Figure 4.8) and "right" (Figure 4.9); for examples of repetitive gestures, see "faster" (Figure 4.11) or "slower" (Figure 4.10).

4.3.2 Constraints

In this subsection, we will tackle the important problem of linking how sensors and gestures are linked. The main goal is to describe the incompatibilities of the sensors with respects to possible gestures, or more precisely, make it clear that given a choice of sensors, some gestures are impossible to detect. In such a case, this does not mean that the gesture needs to be abandoned, but maybe it can be tweaked in order to be well detectable by our sensors.

4.3.2.1 Do not overlap gestures

When designing the gesture dictionary, we must keep in mind that it is undesirable to provide two gestures for which one is included in the other. For example, consider the gesture "go" discussed earlier (4.2.1), and consider another gesture "open hand" which is the starting position of "go". In this case, it is difficult to describe what we expect the system to recognize, because while proceeding to "go", we have performed "open hand" inevitably. Therefore it is unclear which gesture the system should output. For this reason, we will make sure that no gestures is included within another, especially as a starting position.

4.3.2.2 Intuitiveness and ease of realization

Of course, the description of prior gesture dictionaries should have taught us that the most used gestures are highly intuitive and easy to perform. The reason is that complicated gestures tend to be hard to remember, and unlike sign language speakers, military infantry soldiers do not necessarily have the background nor the time to learn complex gesture schemes.

All in all, the most sound reason for making a simple and intuitive gesture dictionary is *because we can*. There is no need to complicate the task for the end user. Some simple gestures are already part of our daily life, so it should be a matter of extending this vocabulary by a small margin in order to comply with having enough gestures for robotic control.

4.3.2.3 Avoid neutral gestures

Let us remember that this gesture recognition setup is meant to be used in an always-on fashion. In other words, the glove will be enabled most of the time, but it is not asked to report a gesture at any time; rather, it should stay silent most of the time and report gesture only when it sees one. If the system reports a gesture when there was no intent from the operator, it counts as a false positive.

Now, it is also our task to choose gestures so that they avoiding such false positives. For example, a bad idea for a gesture in our dictionary would be "hand down, fingers half open"; for it is exactly the rest position of the hand while standing or walking. Therefore, spurious recognitions will happen all the time when the operator is walking! This illustrates the fact that we should definitely avoid using rest positions and common meaningless hand postures (i.e. where the hand does not convey a proper meaning but is rather at rest or used as a tool to grasp, itch, rub, etc.). Other related but less obvious gestures to avoid are open hand gestures with no motion. An open hand is too easy to perform inconciously during normal operation, for example when pushing against a surface. If designing an open hand gesture, it should be combined with a distinctive motion so that it does not happen by accident.

4.4 Our choice of gestures

Given our choice of sensors and the associated constraints, we propose to adopt the following gestures (indications are given for the left hand):

- go: open hand, palm facing left, starting from shoulder height and energetically unfolding the elbow.
- stop: closed hand, pointing upwards.
- left: "L" shape, index pointing front, thumb pointing left (thus back pointing downwards).
- right: "L" shape, index pointing front, thumb pointing right (thus back pointing upwards).
- back: "thumbs up" shape, thumbs pointing backwards.
- slower: open hand, palm facing downwards, moderate up-down oscillations.
- faster: open hand, palm facing upwards, moderate up-down oscillations.
- come: open hand, palm facing upwards, then closing the hand while keeping the positions; possibly repeated several times.

Several of those gestures are directly borrowed from the military gesture dictionary, such as "go", "slower" and "come", and need no further explanation; they are also found in other gesture dictionaries, such as "slower" which is also used by motorcyclists or "come" by divers.

Gesture "stop" was chosen as a closed hand instead of gesture "stop" with an open hand, because, as we discussed above, an open hand gesture with no motion is at risk of generating too many false positives. "Stop" is a closed hand gesture, which is also known as "freeze" in the SWAT dictionary (Section 4.2.1) but now, it conflicts with the military gesture "accelerate" (arm raised, closed hand pushing up and down). So we changed this "accelerate" gesture into another gesture: "faster", which can be described as "open hand, palm facing

Figure 4.6: Gesture "**go**".

Figure 4.7: Gesture "**stop**".

Figure 4.8: Gesture "**left**".

Figure 4.9: Gesture "**right**".

Figure 4.10: Gesture "**slower**".

Figure 4.11: Gesture "**faster**".

Figure 4.12: Gesture "**back**".

Figure 4.13: Gesture "**come**".

upwards, oscillating up and down". Gesture "faster" is easy to learn, since it is the same as "slower" only with the palm rotated 180°. See Figure 4.11.

Figure 4.14: Gesture "**stop**" with an open hand as in the military. We will not use it since it conflicts with "go". (More details in Chapter 8.)

4.5 Unlock gesture?

We will end our discussion of this gesture dictionary by considering a possible enhancement in terms of accuracy and avoiding false detections. A valid concern is the possibility that even if the recognition is tuned to avoid false positives (FP or wrong) or false negatives (FN or miss), it is still not impossible that the system will detect a few undesired gestures.

Figure 4.15: Gesture "**unlock**" originally designed to declare an intent to control the robot.

Figure 4.16: Gesture "**lock**" would then disable the robot control to protect against recognition mistakes.

For this purpose of boosting accuracy, we have designed a very simple and way to explicitly indicate that we talk to the robot (or the target system). This is what we call an unlock gesture. With an unlock gestures, the system is by default "locked", more precisely, not listening to any command. It will only respond after having been "unlocked", which is done when the user performs the unlock gesture. Then the robot is ready to execute subsequent commands.

In early designs of the gesture dictionary, we believed having an unlock gesture was critical:

- unlock: all fingers closed except pinky; in this position, sharp outwards rotation of the wrist.

However as the thesis went on, informal tests unveiled the possibility to actually design a system with sufficiently few false detections *without an unlock gesture*. We therefore removed it from the main dictionary, and the subsequent experiments and results have been done without it. The results (Chapter 8) will show that the detections score are already promising without an unlock gesture. However, it would still be very easy to integrate in the pipeline, as just another gesture which can be normally trained through the graphical interface. For a future implementer, we recommend the unlock behaviour be integrated in the "bridge/interpreter" block of the control chain, therefore making it easy to exactly describe how the system should act when locked or unlocked. One can furthermore implement either a re-lock timer after a few seconds, or consider adding an opposite "lock gesture" to switch the system back into its locked position. Also, the unlock gesture designed above had ergonomics problems since it we have observed finger pain after some time.

When the custom glove was designed, we also incorporated a magnetic Hall effect sensor to create a touchless button. The goal is the following: being able to report when the user "slides" the index finger's side with their thumb. This gesture is very easy to perform but could not be adequately sensed with our initial sensors (flex + IMU), notably because one resistive sensor is not able to retrieve sufficient information about the thumb position, which we have seen is quite complex (see Section 3.1) and also because finger contact is not reported with the flex sensors.

The Hall effect sensor has been found to work well during informal experiments. However, so far we have not fully integrated it into the recognition pipeline for lack of time. We

strongly believe it could be a very powerful way to increase accuracy of the control system by making it easy for the user to switch between locked and unlocked states.

Chapter 5

Building a data glove

Contents

In the previous chapter, we discussed possible gestures for robot control, while in Chapter 3 we described existing data gloves and sensor technology. In this chapter, we find ourselves at the natural convergence of both discussions since we consider building our own, custom data glove for recognition of our robot control gesture dictionary. From the beginning, building our glove was a first-class goal of this thesis: Thales had expressed the demand that we make a custom glove from raw sensors. This provides several advantages:

- the glove can be manufactured in a custom manner to suit our needs;
- we have control over where computations take place, and in particular, we can choose to offload computations to the glove;
- we can choose exactly what types of sensors will be used for gesture recognition;
- we can obtain a glove with appropriate robustness for showcasing to military users.

The last of these reasons is by far the most important and less subject to discussion. Indeed, commercially available gloves discussed in Chapter 3 are generally designed with indoor usage in mind, for example, in a laboratory or in a living room setting. None of them provides a true degree of robustness for outdoors usage.

Consequently, during this thesis we have followed the approach below:

1. as early as possible, investigate the sensors to decide on the most suitable gesture recognition hardware technology for our use case;

2. then, buy a commercially available glove with equivalent sensors, even if it is not robust enough for final use, in order to investigate the sensor pipeline and algorithmic difficulties and validate the gesture recognition system before having our own custom glove;
3. in parallel, design specifications for our custom glove and launch production of a first prototype;
4. validate that the software pipeline and gesture recognition algorithms, built with the commercially available glove, work with our custom one.

Below, we describe the sensors we have chosen to use in our gesture recognition application. Then, we explain why we chose the V-Hand as an acceptable commercial equivalent of our custom glove. Finally, we discuss the design of this glove as a hardware device and show the end result obtained after manufacturing.

5.1 Choice of sensors

5.1.1 Flex and IMU

Given the gesture dictionaries described in Chapter 4, and especially the three important semantic carriers (finger posture, motion, orientation), we propose the following choice of sensors:

- Resistive ink finger flexion sensors, one on each finger. Report a continuous range of openness (0-100%) for each finger;
- IMU (Inertial Measurement Unit), one on the back of the hand. Collects both the motion and the orientation.

Indeed, previous data gloves indicate that resistive ink sensors work well for gesture recognition applications. Resistive ink sensors are well tested by the academic community [71, 161, 184, 186, 199] and readily available. Also, the cost of breaking a sensor and replacing it is small since they have a low price tag. We chose to use $12.95 Spectra Symbol flex sensors, since despite a small overshoot they led to very repeatable sensor readings (see Chapter 7) and they fit well on fingers.

Initially, the IMU provides gyroscope readings (angular momentum in degrees par seconds), and accelerometers readings (including both the gravity direction and the pure acceleration). Through some advanced preprocessing, which we describe in Chapter 7, we can know the orientation of the sensor at any time, even though it is not initially included in the raw measurements. Note that we are not interested in the absolute orientation on the compass plane (north-east-south-west), so it does not matter if we do not have a magnetometer, which would be the appropriate correction sensor for this measurement.

All in all, our three semantic carriers (posture, motion, orientation) are collected thanks to those two sensors.

Regarding the motion channel, it is worthwhile to explain that we do not collect true absolute positioning in coordinates (x, y, z), but the acceleration, which is the second derivative of the position. Explained shortly, the main problem is that position is not returned in

the raw measurement and must then be calculated via integration of the second derivative. Because we do not possess any way to correct this integration, the error (difference between true position and computed position) grows quadratically with time, which translates experimentally to the position being several meters off in a few seconds [20]. Therefore we must settle for no position and take that into account within our algorithms. We describe this problem visually in Chapter 7.

5.1.2 What we can get out of the sensors

In terms of high-level gestures, let us examine the repercussion of sensing posture, motion (acceleration + rotation) and orientation.

Our choice of posture sensing with five finger flexions implies that we can accurately sense many postures with open or closed fingers. For example, most finger postures in the LSF alphabet are detectable with those five fingers sensors, which is an excellent point. The exception would be the detection of finger adduction/abduction (spreading fingers apart and closing them), which would make the sensors mix up letters U (two fingers open but touching length-wise) and V (two fingers open and spread apart).

The military gesture dictionary, i.e. the one closest to our application, is basically a subset of the LSF dictionary when it comes to finger flexion (we are not talking about motion or orientation here), and no complicated posture is part of this gesture dictionary, for the simple reason that those gestures must be easy to perform and visible from afar. Therefore most of them are in line with our choice of sensors for the posture.

The IMU will, after preprocessing, return acceleration, rotation, and two axes of orientation (palm/back facing upwards; palm/back facing ahead). Regarding orientation and static gestures, this provides us with the ability to detect all letters in the LSF alphabet; when it comes to dynamic gestures, acceleration and rotation will provide sufficient information to distinguish at least in which direction the gesture is going. This is more than enough for all domain specific gesture dictionaries above. However, it would not suffice as is for the recognition of sign language, where absolute position plays an important part. As we have seen, sign languages use several axes around the signer's body, which the sensors cannot accurately report.

5.1.3 Limitations

The choice of sensors described above necessarily comes with some trade-offs. Indeed there are several data that our combination of sensors, flex + IMU, chosen for robustness for outdoors use, cannot obtain.

First, we lack accurate position reporting. Therefore we have no indication of where the hand lies, neither around the operator's body, nor absolutely or relatively in the Earth frame of reference. This is due to the double integration problem described before.

Because it is based on glove sensors only, this system suffers from the inability to track other body parts, especially the arm. For example, if we were to detect a gesture where the arm is raised straight, we could not actually know if the arm is raised. The only thing we can do is specify a combination of posture, such as, "all fingers open", along with a specific orientation such as "hand raised". But if the user does the same posture and the

same orientation without the arm raised, say: forearm raised, elbow bent, hand at shoulder height; the sensors would report the same values. Simply put, our sensors do not distinguish arm position.

In the same vein, this sensor combination implies there is no way to detect any kind of contact, whether finger to finger (such as index-thumb loop), finger to palm, palm to palm, palm to body, etc. Consider a gesture involving clapping hands; while it is not impossible to detect it, it would be mixed up with the same gesture involving a single hand clapping against a wall. In this case the motion could help discriminating the gesture nonetheless, in the hope that clapping against a wall has a distinctive enough motion signature regarding acceleration.

As we shall see later, these limitations do not impair gesture recognition as long as the dictionary is well chosen. The rationale is that most gestures we have seen in prior dictionaries can be reported well enough with the chosen sensors, not only in theory, but also in preliminary experiments carried out early in this thesis, which allowed us to validate the hypothesis that those sensors were good enough to recognize gestures, even though the complete gesture recognition pipeline was not completely built.

5.2 The V-Hand as a prototype glove for gesture algorithms

Given our choice of flex sensors and IMU, it was easy to find a prototype to simulate our data glove before it was available. We refer to Chapter 3 for existing data gloves. One answered perfectly to the requirements, except regarding robustness: the V-Hand, manufactured by DGTech, includes a 9-DOF IMU (3D accelerometer, 3D gyroscope, 3D magnetometer) and resistive ink sensors, one on top of each finger. Flex sensors are integrated in a lycra one-size-fits-all fabric; on the other hand, a small box containing the IMU and other electronics such as a wireless chip, a USB connection, and a small-factory LiPo battery, was placed on the back of the hand. While it reported good measurements, we felt the system was a bit clunky mainly due to this electronics box which kept moving during hand motion. Nevertheless, the V-Hand allowed us to test the data processing pipeline and the recognition algorithms.

Based on the V-Hand's datasheet [47], we designed our own communication library to interact with it. A Windows SDK is provided but we wanted full low-level control of the glove under Unix systems. Our communication tool is composed of a single executable passing V-Hand compatible commands to the serial port via simple command-line flags. It implements the whole V-Hand 3.0 protocol as described in the datasheet. Two actions are key: modify WiFi settings, such as SSID, password and IP configuration; and obtain real-time hand data, which is after all why we got the glove in the first place. Motion and flex data are updated 100 times per second. When starting streaming, one must choose between five possible packet types:

1. Quaternion + Fingers
2. Quaternion
3. IMU + Fingers
4. IMU
5. Fingers

As a reminder, IMU means the whole 9-DOF package (gyroscope, accelerometer and magnetometer). Fingers means the five flexion data. Quaternion refers to the orientation data, represented by a unit quaternion indicating the rotation between the IMU frame of reference and the Earth, allowing to distinguish between different orientations.

The perfect mode we would have wanted is "IMU + Quaternion + Fingers". This way, we could have combined the raw accelerometer readings with the quaternion to remove gravity and obtain the pure acceleration. Alas, it was not available, so we decided to fall back on type 3 "IMU + Fingers" and design our own motion processing pipeline to gather quaternion values on our own. This is done in Chapter 7. However, as we shall see below, when designing our custom glove we will obtain quaternion values directly from the IMU's DMP (Digital Motion Processor) instead of computing it. Anyway, implementing the V-Hand protocol was source of some frustration because the datasheet [47] contains several errors, inaccuracies and leftover details such as the size of some fields in the low level packet definition. Nevertheless, after some trial and error we ended up with a reliable framework to process V-Hand data.

The "IMU + Finger" packet detail is described below:

- HEADER: start-of-packet constant marker
- CMD: indicating this packet is a motion packet
- PKGLEN: length until end of packet
- PKGTYPE: indicates this is a type 3 "IMU + Fingers" packet
- ID: glove ID (not used)
- CLK: timestamp with 10ms precision
- Gx, Gy, Gz: three 2-byte numbers representing gyroscope readings
- Mx, My, Mz: three 2-byte numbers representing magnetometer readings
- Ax, Ay, Az: three 2-byte numbers representing accelerometer readings
- F1, F2, F3, F4, F5: five 2-byte 0-1000 number for each finger flexion
- STATUS: report a glove problem (not used)
- BCC: checksum
- ENDCAR: end-of-packet constant marker

We found this style of packet rather good for transmitting hand information. As a result, we were largely inspired by it while designing our custom packet.

Many experiments took place with the V-Hand during this thesis; our custom glove was available only during the last months. Therefore this glove was crucial in making this gesture system possible before we got our first custom glove prototype.

5.3 Development of our data glove

In this section we describe some details behind the construction of our custom glove. For this step, we delegated some work to then-intern Electrical Engineer Clément Letellier, who joined us during 6 months in early 2016, and without whom this custom glove would probably have not existed, or at least not within reasonable delays. Clément provided substantial insights for electronics design considerations and this section includes some of the work we conducted together.

Figure 5.1: The custom glove we describe in this chapter.

5.3.1 Design considerations

Before building the glove, we had to settle a few questions regarding its construction, electronics to include and interface with both the user and the receiving computer.

5.3.1.1 Manufacturer

At least, the clearest point was that we wanted a rugged, solid glove that would truly fit the needs of military soldiers. From the beginning we aimed for a real-world device, not just an experimental one. After discussing with multiple manufacturers, we chose a company specialized in the domains of motorcycle, bicycle and winter sports gloves. It turns out that

motorcycles gloves meet high resistance requirements due to necessity of protecting from abrasion, and as such, answer surprisingly well common military requirements.

Figure 5.2: Our custom glove and the V-Hand, for comparison. One V-Hand flex sensor was put out for visibility. Despite integrating similar sensors, our glove arguably suits outdoors conditions better.

5.3.1.2 Cable or wireless

Which is the best between a wireless glove and a wired one? If the glove is wired, it will function only when physically connected to the receiving computer. On the other hand, a wireless transmission makes it possible to move freely in the environment. While it seems intuitive to choose a wireless glove at a first glance, by examining the question more closely we discover the following problems:

- it requires on-hand power (battery) along with charge-related constraints (time, charging cable, bulkiness of a multiple part system)
- it requires a wireless protocol and use of the radio spectrum (and consider military will probably not use a classic consumer interface such as IEEE 802.11);
- it requires additional hardware to support the communication and the charge;
- it requires a way to set up wireless communication parameters (frequency band, encryption, etc.) and to debug the cause of possible problems when they happen.

Military sources inside Thales indicated that they would rather be comfortable with a wired glove rather than a wireless one, especially given the wireless and battery constraints.

Figure 5.3: The glove with a simple USB extension cord. This gives enough length for connexion into a device in the operator's pocket or near their belt, making the extension cord pass in their sleeve. One could also imagine a smart vest in which a receiving USB port is directly made available at the end of the sleeve, bypassing the extension cord altogether.

Furthermore, if the glove is wired, we can easily justify of having a cable passing inside the arm sleeve in the user's jacket, in order to ultimately have the glove plugged into a receving mini-computer taking care of robot communication. Development of the Z-Trooper, in any case, will include a small wireless control panel carried by the user, so we figured it would be wiser to defer communication to this control panel and not take care of wireless communications ourselves. But most prominently, the desire to obtain a prototype as fast as possible encouraged us to choose the easiest options. In this case a wired glove was just simpler to build.

5.3.1.3 CPU or microcontroller

The question of whether we should use a microprocessor (CPU) or a microcontroller was raised. In short, a CPU would allow more processing power and thus enable part or all of the recognition algorithms to run on the glove, since the motion processing and gesture recognition pipeline requires a single CPU to work during operation mode (Chapter 6).

The final say in our decision was that existing algorithms were already programmed to work on a laptop for this thesis scope. For a proof of concept such as ours, porting the algorithms to an embedded CPU would have been a big time sink for little benefit. Additionally, embedded CPU programming is a specific skill that neither of us had in the robotics lab, and even though it could have been learnt, it is unfortunately time-consuming. Microcontrollers seemed less daunting to setup especially thanks to ready-made platforms such as Arduino which saved us a great deal of time.

Therefore, for complexity reasons, we chose to use a microcontroller of the ATmega family, for compatibility with the chosen IMU, an MPU-6050 thanks to available open-source

Arduino libraries.

5.3.1.4 Button or switch

On top of existing sensors, we wondered if a button should be added for a future use case of interaction from the user to the glove without resorting to gestures themselves. A plain pushbutton was considered, however we preferred to test a more innovative idea: we settled on a magnetic (Hall effect) sensor for thumb-index trigger by sliding. The user can trigger the switch by sliding their thumb along the index, a very easy gesture that does not rely on our gesture recognition pipeline and is thus detected directly at a lower level. A series of two alternating magnets ($+$ then $-$) are installed against the thumb's top phalanx, and the Hall effect sensor is placed on the side of the index's first phalanx. We should note that this switch could serve the purpose of a reliable "unlock gesture" as discussed in Chapter 4.

Figure 5.4: Sliding is sensed thanks to a Hall effect sensor and two alternating magnets. Given the consecutive signs of magnets, the glove reports the sliding direction.

5.3.1.5 Data protocol

At the time of deciding on these design considerations, the V-Hand had demonstrated good ability to recognize gestures at the rate of 100Hz with chosen hand information. Therefore we decided to keep a similar frame format; this time, we adapted it to include the IMU's DMP (Digital Motion Processor) quaternion output which is arguably more precise than our custom technique to derive the rotation quaternion described in Chapter 7. Of course, we also included gyroscope, magnetometer and accelerometer readings as well as finger flexion sensors. There is also a a timestamp and a CRC-8 checksum for packet correctness. We kept

transmission simple by using a serial link at 115200 bauds on top of a USB-serial converter (FTDI Chipset) so that the output cable of the glove is a USB plug.

5.3.1.6 Vibrating motor

Since the glove's main action is to control the robot, it is quite easy to have a direct return of the glove's recognition result by looking at the robot's behaviour. However, we wondered if it could be beneficial to add a haptic return for the user. Since it was easy to integrate and it could provide meaningful insight for a prototype device, we chose to include one vibrating motor inside the glove. Therefore, our data glove qualifies as a haptic device.

The manufacturer's experience with hand sensitivity indicated that the best position for the vibrating motor (in terms of being best felt by an user as a haptic return) was under the wrist, below the palm of the hand. Consequently, they integrated the vibrating motor at this location.

Figure 5.5: The vibrating motor (small circular piece below the strap) is put against the wrist for best sensitivity by the operator.

5.3.1.7 Waterproofness

Since we touted this project's reliability in any weather conditions in Chapter 2, we should be providing a waterproof glove. Unfortunately it is not our main priority so far and it would be problematic if one of our two current prototypes breaks. Therefore we did not conduct waterproof testing. We did, however, cover the sensors and electronics board with a protective waterproof gel. This should provide a reasonable level of protection for this proof of concept.

5.3.2 Overall design

Having now answered all these design-related questions, we show here the final architecture of the glove. First, we chose a board containing both the microcontroller and the IMU: the ArduIMU [204]. The IMU is connected to the ATmega via an internal SPI bus, and the board provides usual Arduino connectors including input/output pins (some of them being ADCs for analog readings) and serial output.

The main interest of the ArduIMU resides in the fact that it becomes the only necessary board in our glove, since it is at the intersection of the IMU, the flex sensors (through ADCs), the Hall effect sensor (through an input pin), the vibrating motor (through an output pin) and the USB communication (through serial pins behind the FTDI adapter). The FTDI adapter is tiny enough to be integrated in the enclosing of the USB plug itself, meaning that no supplementary room is needed on the glove for an adapter circuit. The five flex sensors are wired to the board and read by a typical voltage divider circuit.

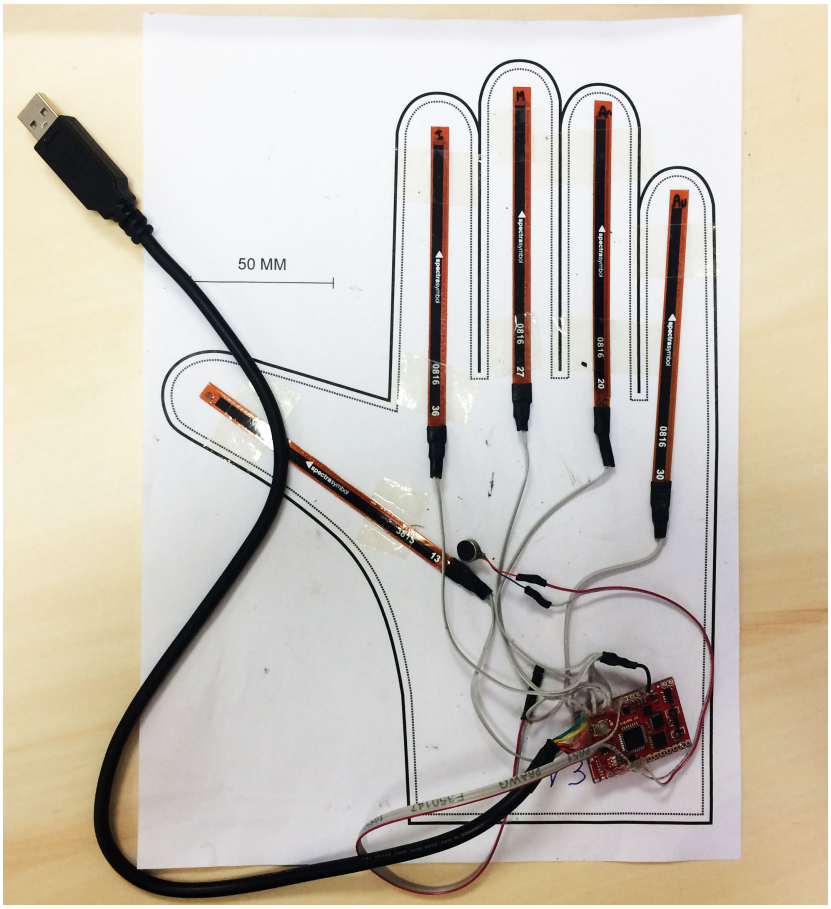


Figure 5.6: The glove electronics before sending to manufacturer. One sees flex sensors, the ArduIMU, the vibrating motor (small circular piece next to the thumb flex sensor) and the Hall effect sensor (small, black stripe below all cables).

Regarding alternating magnets, that is, the $+$ and $-$ thumb-located magnets designed to be read in succession by the index-based Hall effect sensor during thumb sliding, we had a small accident in a first prototype. Indeed, both magnets collapsed one onto the other due to magnetic attraction, rendering the sliding system unusable. For a second prototype, the

manufacturer had the insightful idea to integrate two small pockets at the extremity of the thumb, one for each magnet, in order to prevent them from collapsing.

The ArduIMU board is placed on the back of the hand, between two layers of glove clothing; this is an improvement from the V-Hand since it is well fixed to the hand, unlike the awkward electronics box of the V-Hand which had the tendency to move around during high motion periods. Flex sensors are, as for the V-Hand, enclosed within the glove. Unlike the V-Hand, in our glove nothing is removable because integration was made during construction of the glove itself, not as an afterthought. No maintainance is mandatory regarding the sensors which explains why it is not designed to be opened.

Figure 5.7: The ArduIMU board is very light and is not felt by the user. It is protected between two glove layers.

5.4 Conclusions

Building our custom data glove was a very satisfying experience, since unlike what one could obtain with software-only projects, it is really fulfilling to hold the physical result of one's work. To conclude this chapter, we shall note that two glove prototypes were built (a third being in preparation at the time of writing) and have stood very well to daily use since they were constructed several months ago. The readings are accurate and most importantly we were able to reuse the recognition algorithms that we developed for use with the V-Hand without specific adjustments.

A minor issue concerns the orientation of the IMU board: it has to be placed with the same orientation for each new glove. Otherwise, two gloves whose IMU's orientation differ will lead to incompatible motion time series and therefore a proper training phase will have to be re-done. This is not a very big problem since it only affects the training phase, though;

Figure 5.8: The flex sensors are fully integrated between the glove's inner and outer layers.

moreover the issue is easily mitigated by deciding on a specific orientation once and for all before manufacturing several gloves.

Perspectives include:

- integrate a wireless communication (for possible use in non-military contexts);
- decide of a specific orientation and stick to it, for forward compatibility between gloves;
- certify the glove as waterproof;
- possibly investigate the consequences of scaling manufacturing of the glove for mass-production.

By concluding this chapter, we thereby end most discussion regarding hardware. Starting with the next chapter, we will turn to software-related challenges, including motion processing and proper application of machine learning techniques for recognition.

Part III

Processing gestures

Chapter 6

Software recognition pipeline

Contents

Previous chapters mostly focused on the sensor choice and integration so that we can have a reliable data glove for gesture recognition. We have not discussed yet how the software should handle the sensor data in order to give recognition decisions. In this chapter, we describe the setup of the software recognition system for a real-time setting.

The goal is to start from sensor input and return labeled results of gesture recognition. To be truly real time, we make it possible for each new data point coming from the sensors to go through the whole pipeline and lead to gesture detection as soon as possible by maximizing all processing that can be done on this data point.

6.1 Pipeline description

We will distinguish two modes of operations: training mode and operation mode. While the operation mode is designed to be used most of the time, the training mode is the necessary phase during which a user will perform examples to populate the gesture database used in operation.

6.1.1 Operation mode

The operation mode corresponds to the phase when recognition actually occurs. It refers to the system's behavior when deployed; most of the time, this should be the enabled mode, since recognizing unknown gestures is the most useful state of the pipeline.

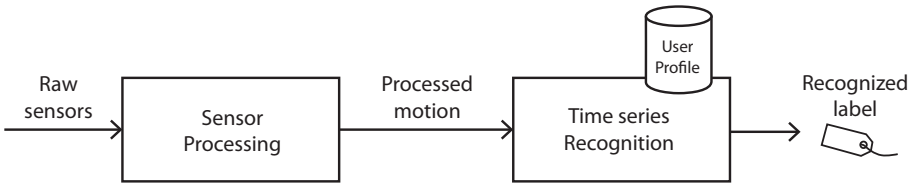


Figure 6.1: During operation, the system analyzes the user's motion and emits gesture labels upon detection.

The operation mode is designed as follows:

Each input point coming from the sensors is processed (see Chapter 7) to be used as a stream data point for subsequent time series recognition (see Chapter 9). The pipeline operates at the data rate of the sensor input stream. The test glove we purchased, the V-Hand, operates at 100 data points per second. Our custom glove operates a little bit faster at around 150 points per second.

Let us see in detail what blocks compose our pipeline as well as intermediate data:

- *Raw sensor data* represents the data given by the glove over USB. Both with the V-Hand and our custom glove, each frame represents an update for all sensors, composed of unscaled values (unit-aware data for physics-based processing):

  – 3x impure acceleration
  – 3x gyroscope
  – 4x quaternion data if available
  – 5x finger flexion

  The quaternion data (1 quaternion composed of 4 scalars) is not available in the case of the V-Hand (Chapter 5), so we compute them in our motion processing block. However, in our custom glove, we made sure to retrieve quaternion data directly from the IMU's Digital Motion Processor.

- *Motion processing* refers to the operation during which all sensor data are analyzed and transformed. Main operations are: motion feature engineering (sensor fusion) and scaling. This step is described in Chapter 7.

- *Processed motion* is the result of the motion processing step, which is suitable for time series analysis in the next stage. It contains scaled values (roughly in [-1, 1]) representing:

  – 3x pure acceleration
  – 3x gyroscope
  – 2x orientation
  – 5x finger flexion

- *Time series recognition* refers to the stage where the incoming processing motion stream is compared (*template matching*) with known instances of gesture time series stored in the user profile. For this step to be possible, the user profile must be populated during the training stage. As we shall see in Chapter 9, the main component of this time

series recognition step is a Threshold Recognizer, in which elastic distances between the stream and stored time series are computed and then compared with a threshold value to decide if a label should be triggered. A user profile is made of the following data:

  - the *gesture database* containing labeled time series. An element is a pair (ℓ, r) where ℓ is a gesture class label and r is a sequence of successive motion data points;
  - *threshold values* T_ℓ under which a class ℓ should be triggered.

- *Recognition output* refers to the system's decision regarding which gesture it believes is made by the user. It consists of simple BEGIN and END markers along with the label name. When the system thinks a gesture ℓ is performed, it outputs (BEGIN, ℓ), and when finished, (END, ℓ). It is not allowed to emit a (BEGIN, ℓ_2) marker if a previous (BEGIN, ℓ_1) marker was not ended by an (END, ℓ_1) before.

By design, this pipeline matches perfectly the schematic interface described in Chapter 2, because it outputs gesture labels without particular knowledge of the system being controlled afterwards. Since it is agnostic to the commanded system, it can be used to control a robot, but also other kinds of devices.

6.1.2 Training mode

As we described above, the operation mode requires a user profile including example of gestures and threshold values. Consider the two following facts:

1. Gestures are performed differently by distinct people [203]. We discuss this in Chapter 8.
2. We want to reach best recognition scores on a single operator and not necessarily cross-operators. In other words it is acceptable to expect a user to teach their own gestures before operating the deployed recognition system.

Therefore it appears that the best recognition results will be obtained by building a training set (i.e. a database of labeled gestures) for each operator who wishes to use the system in operation. Therefore, we will consider a second architecture which is mostly similar but whose goal differ: instead of generating recognition results, it will acquire ground truth data and learn from the user's labeled gestures. This setup is known as the training mode.

6.1.2.1 Architecture

In training mode, the pipeline operates as displayed on Figure 6.2.

Since the main goal of the training stage is to populate the user profile, it does not by itself return a label input. Rather, it needs to have a "ground truth" label information to learn which portions of the stream (time series) represent gestures. Therefore, it accepts BEGIN and END markers from the user in order to perform time series extraction. This label information makes our pipeline a fully supervised learning stage.

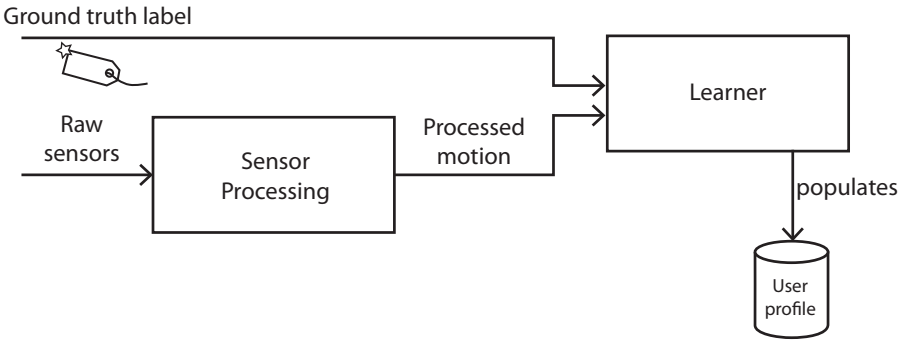


Figure 6.2: By providing ground truth labels along with the hand's motion, the system can learn the gestures it must recognize.

6.1.2.2 Learner Modes

In this thesis we have investigated two possible ways for the system to populate the user profile:

- simple extraction: extraction of the time series from the stream without further analysis;
- advanced training: extraction of the time series and intelligent analysis on the stream history to assess their quality and determine class thresholds.

6.1.2.2.1 Simple extraction.

In this technique, time series are extracted and stored, along with the class label, in the user profile. This is done without further analysis of the extracted time series' quality. Furthermore, some hyperparameters have to be provided manually by the user. This procedure is the simplest to set up because it only involves time series extraction and storage in the gesture database for real-time template matching. It is also instantaneous. However, we will see in Chapter 10 that if the user gives low-quality gestures during learning, it can harm recognition results. This is why an advanced procedure was developed.

6.1.2.2.2 Advanced training

In this technique, the learner extracts time series as previously. However, it will also analyze the quality of provided gestures by comparing each time series of the database against the history stream. A high quality gesture will match well other instances of the same gesture, while a low quality one will fail against similar gestures of the same class, or will be too close of another gesture class. Detecting problematic gestures and deleting them allows a user to make a few mistakes during training without impeding recognition results.

Earlier experiments of this thesis, shown in Chapter 8, have been realized with the simple extraction mode. This will be the occasion to discover problems with the simple training mode and to introduce Chapter 10 in which we describe how to assess quality of extracted time series. In practical terms, the advanced procedure requires less expertise from the user, since it does not need manual hyperparameter tuning and it is resilient to occasional mistakes during training.

6.1.3 Coupling the training and operating pipelines for real-time feedback

In order to give real-time feedback while the user is performing training, it is possible to make a coupling between both the training pipeline and the operation pipeline. Since the training stage only operates on the user profile, it is possible to continue with the operation stage continuously. When the learner indicates succesful computation of a new user profile, the former profile is discarded and replaced by the new one. This is depicted in Figure 6.3. The recognition results are sent back to the GUI in order to be seen by the user in real time.

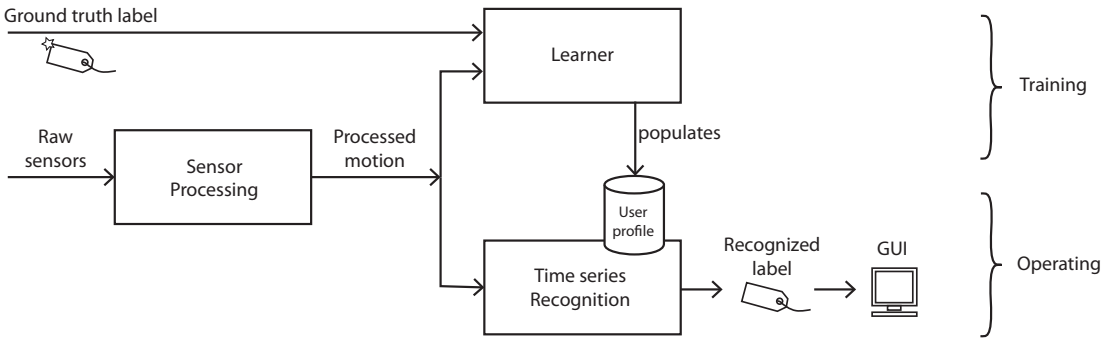


Figure 6.3: By coupling the training and operating pipelines, the GUI can display real-time recognition feedback during training. This allows the user to better understand their influence on the recognition system.

6.2 GUI

In this section we describe how to obtain gesture data from the user, and display motion information and gesture recognition results for real-time feedback. During this thesis, we have experimented with two main version of our recognition GUI. Each of these versions corresponded to a technique for training gestures. In both cases, the GUI was always a succesful demonstration that the technique works in real time. Template matching uses always a single CPU and performs without stuttering on 100 Hz or 150 Hz data.

6.2.1 First prototype: simple training, instantaneous visualization

During this thesis, it was a real issue to design a user-friendly way to label gestures in a supervised manner, with limited input from the user. Over time, we have built two versions of a Graphical User interface (GUI) corresponding to two modes for obtention of gesture information.

The first prototype was built on the premise that our user would push-and-release a button whenever they want to input a new gesture. In this setup, the system is almost always in operation phase, except during the push-and-release interval where it also activates recording of the current gesture. As soon as the pushbutton is released, the system extracts the recorded time series and places it in the user profile for subsequent template matching by the recognition block. Therefore it corresponds to the simple extraction mode.

The processed motion input was shown to the user by indicating real-time values of each 13 dimension including accleration, gyroscopes, orientation and finger flexion. Most

importantly, gesture recognition output was prominently displayed to the user in bold, red text, in order to give real time feedback of the pipeline's results.

While the push-and-release technique was efficient, it required the user's two hands since one hand must perform the gesture with the glove whereas the other presses and releases the pushbutton. This was a good technique for standing captures but impractical for captures in motion (walking, running).

6.2.2 Second prototype: advanced training, time visualization by color signatures

In the second GUI prototype, we wanted to provide the following enhancements: - more intuitive motion visualization - automated ground truth labeling (avoid requiring awkward simultaneous user action on the GUI) - automated computation of class thresholds (avoid requiring user tweaking of non-friendly numeric values)

In order to palliate each of these issues, we proposed the following. First, instead of reporting instantaneous values on the dashboard, we proposed to make the user more aware of the "stream" and "time series" aspect of the recognition. An obvious answer would have been to use classic time plots, i.e. curves with horizontal time axis representing the processed motion data, but it would have required showing 13 scalar time plots to the user, which is an important load of information and consequently not very user-friendly.

Rather, we decided to choose a simple color-based visualization. This is shown in Figure 6.4. Since gyroscopes and pure acceleration are 3D vectors, and orientation is a 2D vector (interpreted as a 3D vector with a zero component), we decided to map 3D vectors values to an RGB cube. As we will see in 7, input data is scaled to fit between -1 and 1. Here are a few examples of such a mapping:

- 3D vector $[0, 0, 0] \rightarrow$ RGB $[128, 128, 128]$ (gray)
- 3D vector $[1, -1, 0.5] \rightarrow$ RGB $[255, 0, 192]$ (purple)

As for flexion data, it was displayed as five black independent black and white values.

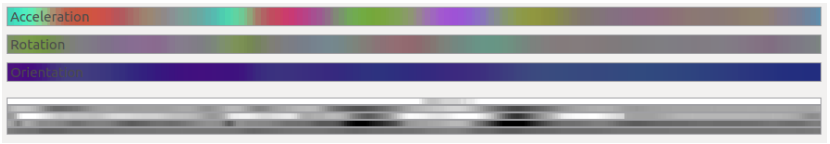


Figure 6.4: The RGB stripes are easier to read than 3 classical plot curves.

Thanks to this simple, colorful representation, we were able to display acceleration, momentum and orientation by three color-based stripes with an horizontal time axis. New values are added to the right and old values are shifted left, which at 100 or 150 Hz renders as a smooth and natural scrolling visualization of the real-time motion performed by the user. Using this representation a user quickly sees that doing the same movement twice results in two similar color signatures, which is designed to help them understand what goes into the "brain" of the otherwise opaque machine learning stage. It is easier to explain them that the system sees time series of color data and successful recognition is triggered when the system detects a known color stripe. Upon successful recognition, not only the GUI displays

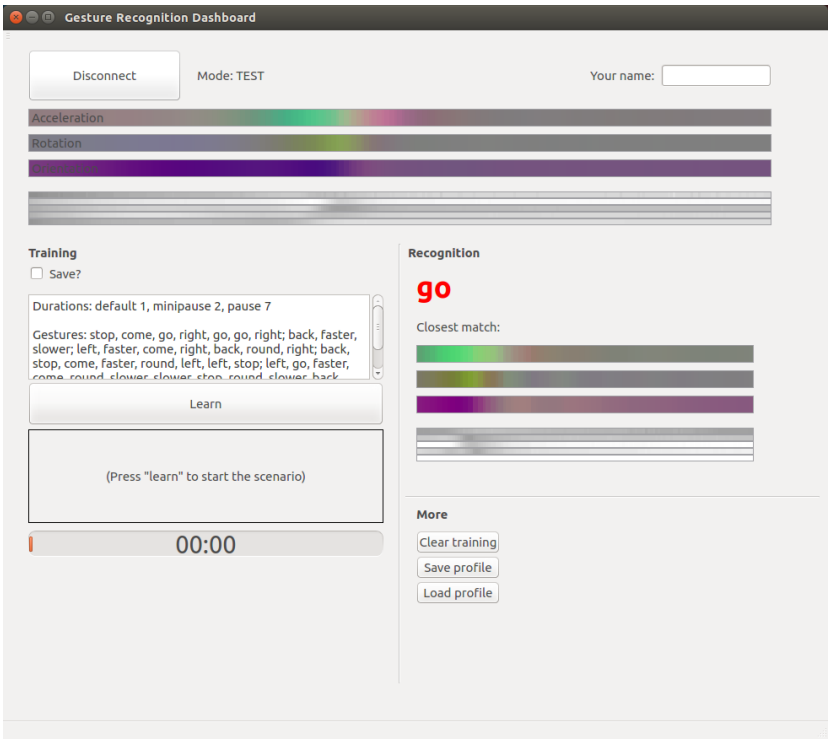


Figure 6.5: GUI in operation stage. The recognized gesture is displayed in bold, red font.

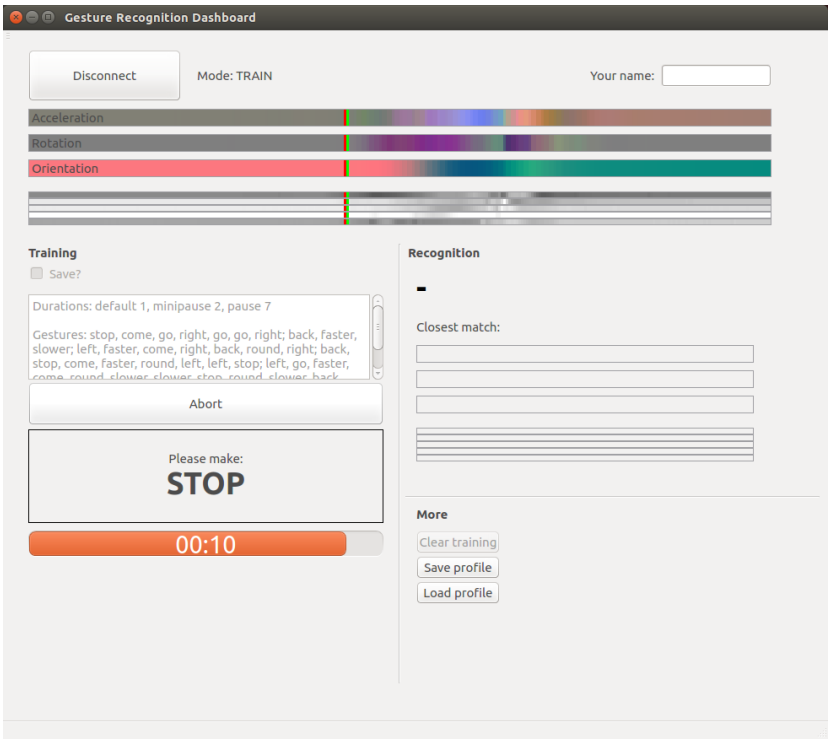


Figure 6.6: GUI in training stage. The left-hand side contains training settings, which includes the "scenario" describing the gestures to be asked for one capture stream. A visual timer indicates when the user is expected to perform the gesture. The bright vertical bar over the RGB stripes (top of the window) indicates the BEGIN boundary of the gesture currently recorded.

prominently the recognized label, but also shows which time series is currently matching the stream, thanks to the color representation.

Automated ground truth labeling was a challenge for which we propose the following solution. First, the user writes a "scenario", which is simply a sequence of gestures the system will learn. (For user-friendliness, a default scenario is automatically entered at startup, so a user normally does not even have to modify the scenario in the first place.) When ready, the user presses the "Train" button only once, at which point the system switches into training mode and start accumulating data. The interface asks each gesture, in order, as written in the supplied scenario. A visual timer, implemented as a progress bar, makes the BEGIN and END time bounds known to the user so they know when they are expected to provide gesture data. No action on the GUI is required until the scenario is over, which takes around 3 minutes for the default scenario.

Under the hood, the training system performs the advanced learning procedure in parallel. Each time a new labeled time series is available, the advanced training procedure starts on a second thread and begins to analyze the known time series data set against the whole labeled stream. It detects which set of gestures are best and reports associated thresholds for each class. This procedure is described in Chapter 10. In the meantime, the user continues to proceed with recording the gestures of the scenario.

In the following chapters, we will conduct an in-depth description of the main components of the pipeline. In Chapter 7, we will deal with the Motion Processing block; then, Chapter 8 will discuss possible techniques for isolated recognition, which although not a block of this pipeline, will yet serve as a basis for Chapter 9 in which the Gesture Recognition block is described. Problems with the simple procedure are discussed and the advanced training procedure is then provided in Chapter 10.

Chapter 7

Motion Processing

Contents

7.1 Introduction

Before being able to use the hand data for proper recognition, it is necessary to clean the data and prepare it. Of course, this work must be done in streaming. We consider, in order, the following steps:

- cleaning / denoising
- feature engineering
- scaling

The cleaning step refers to the possible operations aiming to remove discrepancies in the data and enhance its quality. For instance, a typical data analysis cleaning step would involve looking at the data and making sure outliers, such as oddly-elevated values, are dealt with. Sources for outliers can be due to bad sensor measurement but also to transmission errors, which can lead to unexpected highly significant bits being flipped or missed.

In the case of the V-Hand, the rare incorrect data frames do not make their way to the motion processing system due to being rejected by a checksum control. In the case of our custom glove, however, some incorrect values have made their way despite a CRC check; fortunately, erroneous frames were always completely incorrect and we could reject them via

simply checking that a frame timestamp's is consistent with the previous one (i.e. increasing, not too far).

The feature engineering step wil be described below. In short, the flex sensors need no further feature engineering, but it is essential that we process the IMU data in order to obtain "hidden" information such as orientation and "pure" acceleration.

Regarding scaling, it is a necessary operation so that differences between finger flexions are as signifcant as difference between acceleration, rotation or orientation. If no scaling is carried out, one signal might vary between, say 0 and 1, white another varies between -1000 and 1000, which will lead the DTW processing step to mostly ignore the differences in the former signal's values in favour of the latter one.

In our case, the most sensible way to perform scaling is to multiply every signal by a well-chosen value so that its typical range falls between -1 and 1. We shall choose identical constants for scalars coming from the same sensor, so that for example are three gyroscope scalars share the same scaling constant. It is not, however, strictly required to clamp the scaled values if they occasionally exceed those bounds; this will not cause issues with the time series recognition stages as long as all signals have roughly the same variation.

7.2 Flexion sensors

In Chapter 5, we decided to use resistive ink flex sensors. The raw reading of those sensors is already good; the range between straight and bent positions describes well the different intermediary positions. In the V-Hand, the flex sensor readings range from 0 to 1000 due to intermediary preprocessing by the glove firmware. During tests, it appeared that such a resolution was well sufficient to describe finger flexion in our recognition application. Experiments (see Chapter 8) also show that the V-Hand flexion data copes well with DTW and DTW* for recognition.

7.2.1 Overshoot

Resistive ink flex sensors exhibit a phenomenon known as overshoot: the signal peaks for a fraction of seconds, when the sensor is bent fast, which occurs quite frequently as fingers can go through open-close cycles multiple times per second [68].

In Figure 7.1, we can observe a typical overshoot pattern. The problem is well known [161, 199], and although overshoot can be mitigated by custom fabrication techniques (Gentner and Classen [71] propose to add a 0.2mm PVC foil on top of the sensor) it was worth trying recognition despite the overshoot phenomenon, since it would allow us (or any future implementer of this gesture recognition system) to choose mostly any kind of flex sensor in the market. Building our own sensor is difficult and somewhat does not fit well the scope of this thesis in recognition.

As we shall see in Chapter 8, it turns out the overshoot problem is not an obstacle to the obtention of good time series for the gesture recognition. We thus decided to leave the overshoot unprocessed. The main reasons are as follows:

- our purpose is not accurate measurement of angular bending of the fingers (such as goniometric gloves [50, 200, 241]), but rather obtaining a repeatable signal for the same

gesture [71];
- experimental results (Experiments of Chapter 8) confirm recognition works well despite overshoot.

In order to assess reproducibility of this overshoot, we devised a custom bench for the repeating bending of a flex sensor. The bench was simply composed of a servomotor physically linked to the flexion sensor, so that turning the servo one way or another would respectively bend the sensor or release it. After leaving the bench running for 45 minutes, we obtained the trace in Figure 7.2.

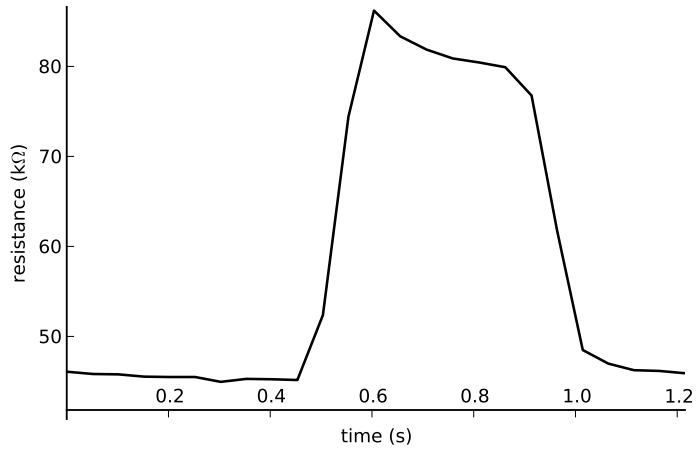


Figure 7.1: The overshoot phenomenon is the peak in resistance values at the beginning of the flexion phase.

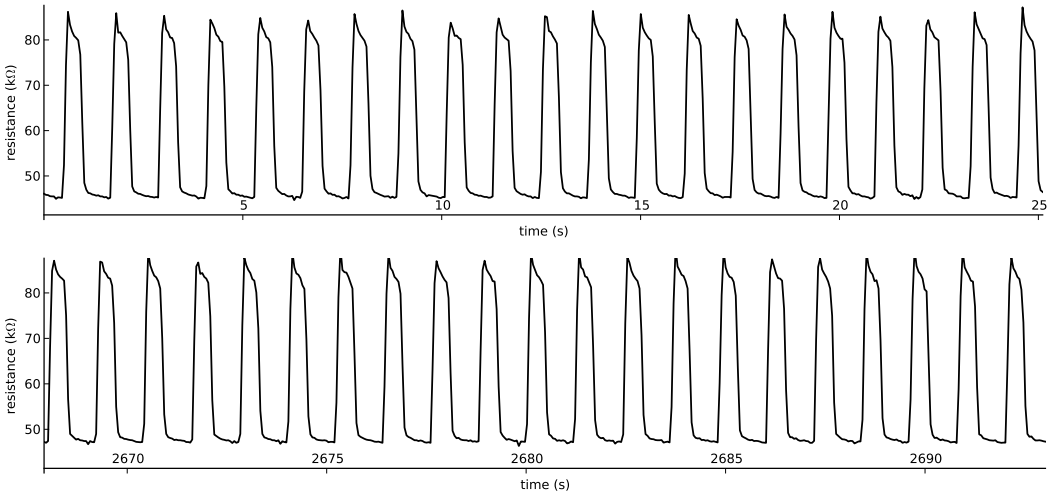


Figure 7.2: The sensor was subject to repeated flexion during 45 minutes (top: beginning, bottom: end). The sensor signal appears to be quite stable throughout time.

Figure 7.2 suggests that the overshoot behaviour is well repeatable and does not vary much for the same flexion. We can thus be reasonably confident about reproducibility of the finger flexion trace for a gesture repeated multiple times.

7.2.2 Noise

A second aspect of the finger flexion sensor could possibly be a problem. Indeed, as with any electronic (especially analogic) sensors, the signal might vary with respect to the true value. In our case, it could be either due to the natural variation of the resistance value, by the voltage divider steps and potential swings in the reference voltage, or by the ADC stage of the capture board.

Looking at Figure 7.1, we can see that the noise is not significant with respect to the signal; also, because we intend to use the finger flexion sensor with DTW-based distances, it might be worth noting that such distances show a good resilience to noise. For instance, see dataset "CBF" of the UCR time series dataset, which exhibits high noise patterns yet leads to good recognition results (99.7% accuracy) [34].

Lastly, it is important to consider the drawbacks of denoising: the main disadvantage is the incurred latency, which is unavoidable for any stream-enabled noise correction algorithm.

In short, we decided not to remove noise from the flex sensor data, mainly because there is virtually none (indicating good sensor quality), but also because it would cause latency in the signal, which is undesirable if denoising is not strictly necessary.

7.2.3 Feature engineering

We did not identify any fusion of data that could be worthwhile to engineer, on top of the five raw flexion data. Of course, we could have designed such features as the mean of a group of fingers, such as index, middle, ring, pinky into one signal, but we did not consider it as an informative feature since it would add no real information on top of the initial sensors unlike the IMU feature engineering described below, which does add some interesting information.

Therefore, the output of the finger flexion stage is simply the five finger signals.

7.2.4 Scaling

As we described above, the values should be placed between -1 and 1 for good comparison with other values. However, we found out finger data are very strong and it is enough to move them into the range between 0 and 1. As the V-Hand returns values between 0 and 1000 it is a simple matter of dividing each data point by 1000. In our custom glove, we repeated this 0-1000 scheme in the glove's firmware, so that the same scaling can be applied.

In short: flexion data are simply multiplied by a constant scalar so as to fall in the range $[0, 1]$.

7.3 IMU

7.3.1 Introduction

A typical IMU is in general composed of the three following systems: accelerometers, gryroscopes, and magnetometers. They each serve a different purpose and their signals can

be merged together to obtain additional information. Accelerometers are based on an internal, microscopic mass inside the sensor, which is sensitive to both the IMU's motion and the gravity. Further processing is needed to distinguish between both, which we will describe below. Gyroscopes return information about the IMU's momentum, also known as rotation speed. It can be equivalently described as the first-order derivative of the angular position (Euler angles). The third kind of sensor, magnetometers, serve mainly for correction purposes; they return the direction and strength of the magnetic field around the IMU. In general, it is designed to indicate the north, but it can also be subject to magnetic perturbations if there are magnets, ferro-magnetic material, or magnetic field generators such as DC motors nearby.

The IMU is placed on the back of the hand in order to describe the motion of the hand regardless of finger flexion. In the following, we will always follow the coordinate system as shown in Figure 7.3. The vector $\vec{z}$ indicating the upwards direction, goes outwards from the back of the hand, orthogonal to the plane formed by the hand itself; the vector $\vec{y}$ goes in the direction of the middle finger (when straight); vector $\vec{x}$ is placed so that $(\vec{x}, \vec{y}, \vec{z})$ is an orthonormal basis, and therefore goes in the direction of the thumb finger provided it is well perpendicular to all four other fingers. As a consequence, the plane of the hand should be described by vectors $\vec{x}$ and $\vec{y}$. This notation implies the glove is a left glove; if a right glove, the $\vec{x}$ axis should still go to the right (an orthonormal basis in our 3D space is the same regardless of which hand wears the glove) and therefore it points to the right thumb's opposite direction.

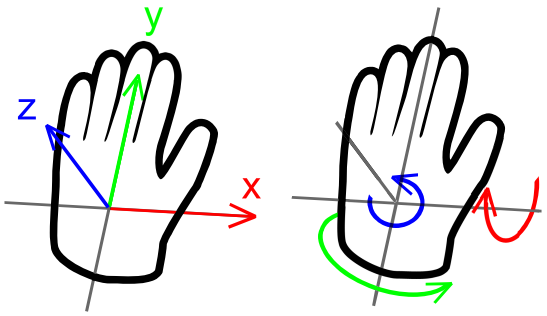


Figure 7.3: The orthonormal basis convention we use throughout this thesis. In all 3D plots of this document, RGB colors refers to XYZ axes.

7.3.2 Magnetometers

Magnetometers return a simple reading, that of a 3D vector indicating the magnetic field in the vincinity of the sensor. In general, the sensor is expected to capture the North direction, which is the strongest field if no other magnetic field is generated nearby by a magnetic object or an electrical current.

We chose not to use magnetometers, and here we explain the reasons of our choice. The first reason is a constraint regarding the environment of the user. In general, presence of any magnetic field generates perturbation on the sensor and the north value get inaccurate as a result. It is very much possible to happen in a military scenario, where objects such as weapons can be carried. Furthermore, because we intend to use this glove with a robot, we

must keep in mind that this robot will generate strong magnetic fields nearby due to motors.

The real question is whether magnetometers are useful at all in our application. Remember, their purpose is to make sure the sensor is well-aligned on the compass plane (parallel to the ground). Without this correction, the orientation will still be correct, but the calculations might indicate that a given gesture happens, say, in the North direction, for instance, whether it is done to the East. However, it is straightforward to understand that it doesn't harm gesture recognition: indeed, if a user does a gesture while North-oriented, and another while East-oriented, it is still the same gesture, and our gesture recognition system should recognize it as such. Therefore, not only we do not need the compass direction (for which magnetometers would be necessary), but we even actually *need* to discard it so that gestures can be recognized irrespective of their absolute heading.

7.3.3 Gyroscopes

Gyroscopes are MEMS devices which are sensitive to variation in the IMU's rotation. The physical phenomenon making it possible to obtain the rotation velocity is the Coriolis effect applied of a high-speed vibratory structure. Fortunately, IMU chips integrate all necessary components, including vibration regulation, amplification, sensor processing, and data bus interface (in general I2C or SPI) allowing us to retrieve the rotation speed without further calculations. Typical gyroscope data transmitted by the IMU chip are given in degrees per seconds (dps).

Gyroscopes are not perfect and are subject to many kinds of fine-grained discrepancies between the true rotation and the actual sensor output. Those include errors in the scale factor (linearity error, non-linearity, asymmetry, stability) and in the zero-rate angle drift, also known as bias, usually defined in terms of the Allan variance components (angle random walk, bias instability, rate random walk) [1].

The most prominent source of error in gyroscope is the bias, which is usually constant on a short-term basis (around one minute), but varies on longer-term time scales (around one hour). Seen in the shorter time scale, the typical correction is to wait for a time where the IMU is immobile, in order to estimate the bias, store it and subtract it from further signal. However, in our gesture recognition application, it has turned out that bias estimation procedure would usually estimate a false bias, for two reasons: first is the difficuly to have the hand truly immobile, as it is very often in motion, all the more in our mobility-induced context; second is the short timescales on which this can happen, leading to poor averaging of the gyroscope values for bias estimation and thus giving false bias estimators. The outcome of a bad bias estimation can be worse than no estimation at all, because it could increase the error instead of the intended goal of decreasing it.

Our experience is twofold regarding gyroscope bias estimation. On the one hand, we found out that not correcting the gyroscope bias will lead to slow degradation of the orientation; without correction the orientation could be out of place in as few as 30 seconds. On the other hand, it will be necessary, in any way, to have a strong correction scheme for the orientation; in this case not bias-correcting the gyroscopes is fine since it will be handled by the orientation correction scheme anyway. The technique we will describe later for orientation-correction is aggressive enough to maintain orientation in-place, making gyro bias correction useless in the first place. The same reasoning holds for scale factor correction. We have therefore decided to leave the gyroscopes uncorrected at this point, and it

works quite well as we shall see below.

The gyroscope signals will be outputted as is (after scaling, 3 scalar signals for rotation around the x, y, z axes) and will also be fed to our IMU feature engineering stage, along with raw acceleration in order to calculate the pure acceleration and the orientation.

7.3.4 Accelerometers

IMUs also contain accelerometers. These are devices returning a combination of the IMU's own acceleration and the gravity, as a 3D vector as seen from the sensor frame of reference. When the device stands still, on a table for example, it returns a 3D vector indicating "upwards", i.e. the opposite of the gravity. When in motion, the sensor still returns the gravity added to the motion of the IMU. The typical units returned by accelerometers are in g, where $1\text{g} = 9.81\text{m.s}^{-2}$. Thus g does not stand for a force but for an acceleration, which is always the case with IMUs.

As to avoid any further confusion, let us describe two kinds of accelerations:

- "pure" acceleration: the true acceleration underwent by the IMU (zero if no motion)
- "impure" acceleration: the raw accelerometers output, which is simply the vector addition of the gravity component and pure acceleration.

The relationship between both can be found in Figure 7.4.

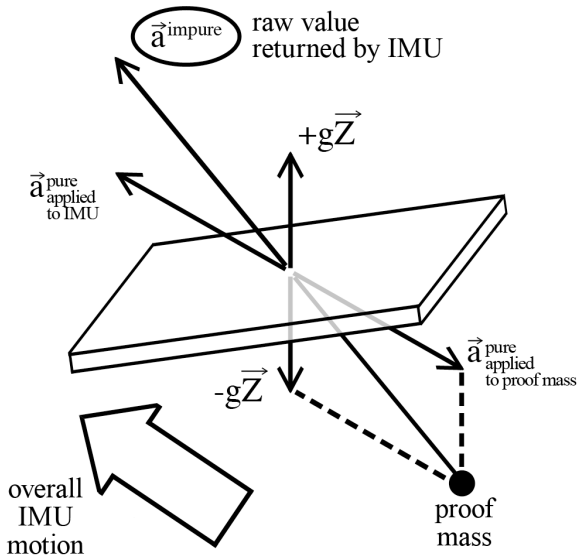


Figure 7.4: The overshoot phenomenon is the peak in resistance values at the beginning of the flexion phase.

Early in this thesis, usage of the sole acceleration data suggested that impure acceleration obtained from the sensors without further processing is not suitable for time series recognition. Therefore, we will not output the raw acceleration data directly, unlike gyroscopes. On the other hand, we will proceed to sensor fusion in order to "purify" the acceleration (remove the gravity component) and also recover the orientation. This will be explained in the next subsection.

7.3.5 Sensor fusion: orientation and pure acceleration

This step is arguably the most important of the whole motion processing stage. It counts as a feature engineering step: indeed, we will obtain two essential vector features: orientation and pure acceleration. Roughly speaking:

- orientation refers to the 3D rotation between the sensor and the ground
- pure acceleration is the "purified" version of the impure acceleration given by accelerometers, such that it represents only the IMU's real acceleration without being tainted by the gravity component.

We chose to use quaternions because, among other ways to represent rotations (Euler angles, rotation matrices, axis-angle), they do not suffer from their counterparts issues. Euler angles are a popular, three-dimensional representation requiring low computation overhead, but they suffer from a singularity issue which leads up to diverging values in some corner cases; this problem is related to the famous gimbal lock issue in mechanical systems responsible for some trouble in Apollo missions [83]. Rotation matrices solve this problem but are too costly: 9 component matrices lead to a high CPU footprint. Finally, an axis-angle representation would be possible but is just impractical to use, unlike quaternions which have a solid theoretical framework and are very easy to use thanks to simple primitives (multiplication and conjugation) able to represent complex operations (rotating, composing rotations, unrotating, etc).

In the following, we reuse the notations of [132]. By convention, all quaternions represent rotations and therefore are unit, i.e. their norm is equal to one. Quaternion multiplication is denoted $\otimes$ and the conjugate of a quaternion q is writen q^*. In multiplications involving quaternions and vectors, it will be necessary to identify a 3D vector $\vec{v} = (v_1, v_2, v_3)$ with the quaternion $\vec{v} = (0, v_1, v_2, v_3)$.

Let us consider two frame of references (represented by direct orthonormal bases), the Earth frame and the Sensor frame, respectively abbreviated E and S:

$$\begin{aligned} E &= (\vec{X}, \vec{Y}, \vec{Z}) \\ S &= (\vec{x}, \vec{y}, \vec{z}) \end{aligned} \tag{7.1}$$

Let us agree that in accordance with Figure 7.16, $\vec{Z}$ points upwards, i.e it is the unit vector of opposite direction with respect to the gravity, and that $\vec{X}$ and $\vec{Y}$ are not particularly constrained apart from forming a direct orthonormal basis with $\vec{Z}$. Hence, by definition, the plane yielded by $(\vec{X}, \vec{Y})$ represents the ground. We require $\vec{X}$ and $\vec{Y}$ to stay immobile with respect to the Earth.

The rotation between both frames can be described thanks to the quaternion ${}^S_E q$, so that a vector v, described in frame E by ${}^E v$ and in frame S by ${}^S v$, satisfies the following relationship:

$${}^E v = {}^S_E q \otimes {}^S v \otimes {}^S_E q^* \tag{7.2}$$

Note the property: ${}^S_E q^* = {}^E_S q$.

Accelerometers return a value ${}^S\vec{a}^{\text{impure}}$ which represents the acceleration in the sensor frame S; the link between pure and impure acceleration is described by the following relationship [173]:

$$\vec{a}^{\text{impure}}(t) = \vec{a}^{\text{pure}}(t) + g\vec{Z} \tag{7.3}$$

where $g = 9.81\,\text{m/s}^2$ is the gravitational constant (relative to the proof mass) and $\vec{a}^{\text{pure}}$ is not tainted by the gravity component. When the IMU is still, we therefore have $\vec{a}^{\text{pure}}(t) = \vec{0}$ and $\vec{a}^{\text{impure}}(t) = g\vec{Z}$, which means the IMU points upwards.

Orientation is therefore important, not only in itself but also for pure acceleration. Indeed, it is necessary to known at any time where the gravity is pointing in order to subtract it from the impure acceleration.

Knowing orientation is possible thanks to integration of the physical equations of motion. It suffices to sum the infinitesimal rotations returned by the gyroscopes at each time step. This first step is named *estimation*; it accumulates errors, though, and must therefore be recalibrated by a step known as *correction*.

Often, estimation correction systems for IMUs are based on Kalman filters [135, 178]. Unfortunately, the implementation is complex, requiring high calculation costs and a lot of fine hand-tuning for the determination of the various parameters. We have based this work on those of [132] who shows that simpler systems can be satisfying.

7.3.5.1 Estimation

Thanks to rotation speed $\vec{\omega}_t$ returned by gyroscopes at each time t, one can obtain the derivative of quaternion ${}^S_E q$ then its new value after rotation [132]:

$$\begin{aligned} {}^S_E\dot{q}_t &= \frac{1}{2}{}^S_E q_T \otimes \vec{\omega}_t \\ {}^S_E q_{t+\Delta t} &= {}^S_E q_t + {}^S_E\dot{q}_t \Delta t \end{aligned} \tag{7.4}$$

The result of this addition must be renormalized in order to guarantee the obtained quaternion represents a rotation.

7.3.5.2 Correction

7.3.5.3 Correction scheme #1: ZUPT

The first correction scheme we used during this thesis was based on ZUPT (Zero Update) correction [101], which means that the correction mechanism is only triggered when the IMU is known to be static. In this case, when the IMU undergoes motion, we cannot rely on the accelerometer raw values to determine the upwards-pointing direction $\vec{Z}$. We can only compute a $\vec{Z}_{\text{estimated}}$, by writing the upwards vector ${}^E\vec{Z} = (0, 0, 1)$ in the sensor frame S, thanks to our orientation estimator:

$${}^S\vec{Z}_{\text{estimated}} = {}^S_E q^*_{\text{estimated}} \otimes {}^E\vec{Z} \otimes {}^S_E q_{\text{estimated}} \tag{7.5}$$

However when the IMU is known to be static, which is the case when the norm of the standard deviations of the acceleration is near-zero (found out by comparison to a low

threshold), we can seek to rotate the previously obtained quaternion: ${}^{S}_{E}q_{\text{estimated}}$, into a corrected quaternion ${}^{S}_{E}q_{\text{corrected}}$ obtained by rotation composition:

$$ {}^{S}_{E}q_{\text{corrected}} = q_{\text{corrector}} \otimes {}^{S}_{E}q_{\text{estimated}} \tag{7.6} $$

Ths is possible statically because we can obtain an alternative measure $\vec{Z}_{\text{alt}}$ for $\vec{Z}$, making it possible to define a $q_{\text{corrector}}$ bringing $\vec{Z}_{\text{estimated}}$ back to $\vec{Z}_{\text{alt}}$:

$$ q_{\text{corrector}} = r({}^{E}\vec{Z}_{\text{alt}}, {}^{E}\vec{Z}) \tag{7.7} $$

This correction scheme has worked well for the capture of small time series, because the sensors do not have enough time to diverge. However, it relies on the hypothesis that the user will stop moving on a regular basis; this hypothesis is liability as it is likely that any scenario will lead to continuous mobility, just by virtue of walking, for example.

7.3.5.4 Correction scheme #2: always-on correction with correction limiting

Therefore, it was necessary to find a technique making it possible to proceed with continuous correction of the sensor data. In order to achieve this, we designed a simple yet effective technique which does not rely on periodical resting of the hand.

In general, the estimated upwards-pointing vector should be same as the accelerometer output. Correction scheme #1 is based on this assumption, but it falls short when the IMU is in motion precisely because the accelerometer does not represent the upwards-pointing vector. To solve this, in this second scheme we:

- turn the orientation so that the upwards-pointing vector $\vec{Z}$ is always attracted to the accelerometer output, even during motion;
- set a small "clamping" angle so that this corrective rotation is limited in effect (i.e. orientation cannot be attracted too fast to the impure acceleration). This is the novelty.

A more technical description is given below. Begin by fixing the angle ψ, relatively small. This angle is the key of the the correction limiting mechanism. At each new acceleration value, first find out which rotation would be needed to rotate the current $\vec{Z}_{\text{estimated}}$ (in sensor frame) to the current impure acceleration. This leads to a rotation with axis $\vec{a}$ and angle $\theta \leq \pi$. Second, clamp angle θ so that it does not exceed ψ. If ψ is small enough, this rotation is very subtle; we call it the corrector rotation. Finally, apply the corrector rotation to the current orientation ${}^{S}_{E}q_{\text{estimated}}$:

$$ {}^{S}_{E}q_{\text{corrected}} = q_{\text{corrector}} \otimes {}^{S}_{E}q_{\text{estimated}} \tag{7.8} $$

The interest of limiting the correction angle is that the correction does not have time to follow the high-motion parts caused by the IMU's motion. It is, however, always attracted to the constant gravity component (the upwards-pointing vector) of the impure acceleration. The effectiveness of this technique stems from the fact that the typical acceleration of the hand has a wave-like signal, and in general, a spike on one direction will be followed by a similar spike in the opposed direction [20].

7.3.5.5 Comparison of both schemes

In order to illustrate the usefulness of correction schemes, and give the reader a better idea of what raw IMU data looks like, we applied our correction algorithms on seven seconds of typical hand gesture. In this case, the user did the following gesture: raise the hand, then put the hand back in resting state. This "sentence" was repeated three times.

This is best seen on the gyroscopes capture: the first "bump" corresponds to rotation due to raising the hand; the second "bump" refers to the reverse operation where the user goes back to resting state. This couple of bumps can be seen three times, leading to the six bumps seen on the figure.

Let us make a few explanations of these graphics:

- At the beginning of the capture, the hand does not move; the reader can see the impure acceleration being almost constant. This is the ZUPT situation. We know that the IMU undergoes no acceleration, so the output of the impure acceleration is exactly the upwards-pointing vector $\vec{Z}$ expressed in the IMU frame. In this case, we can approximately read (look at the first point of the impure acceleration) the vector (x=4, y=5, z=7.5), whose norm is approximately 9.8, the magnitude of the gravity.
- At the beginning, the pure acceleration is zero, because the sensor is calibrated thanks to ZUPT. As far as we are concerned, *the goal of the correction scheme is to have the pure acceleration being zero at the end of the capture*, after the seven seconds.
- Without a correction scheme, the pure acceleration has diverged during the 7 seconds: indeed it is equal to a non-zero constant. This is the initial situation.
- With correction scheme #1, the pure acceleration is brought back to zero when the system finds itself in another ZUPT situation, i.e. at the end. We can see the brutal change of the pure acceleration on Figure 7.8.
- With correction scheme #2, the pure acceleration does not suffer from such a limitation; it is always enabled, yet the high-frequency spikes of the acceleration are still well drawn, as one can see by comparing them with the impure acceleration. If the correction scheme was too aggressive, the spikes in acceleration would be lost, which is a problem since they are the most useful information for gesture recognition.

Figures 7.7 to 7.14 suggest the most acceptable correction scheme is #2. This is on purpose: it was designed to accomodate fast motion with no resting periods, especially in the context of hand motion where acceleration peaks are usually short and exhibit wave-like forward-backward behaviour. In our final application, we use correction scheme #2.

7.3.5.6 About integration

It could be interesting to obtain first- and second- order integrals of the acceleration, also known as velocity (or speed) and position. Such features could be interesting in the context of gesture recognition.

Unfortunately, it is a well-known issue that IMUs do not tend to give correct velocity and position outputs. The reason is due not only to sensor non-linearities and other errors,

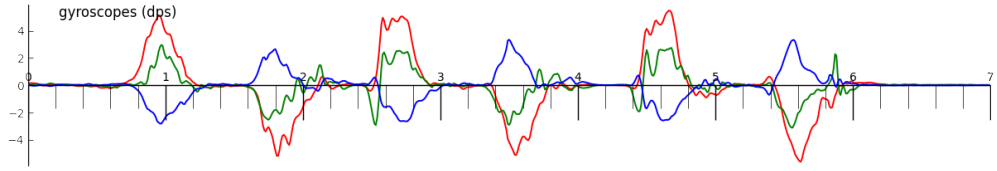


Figure 7.5: Gyroscopes. Peaks represent the sentence "hand up, hand down" three times.

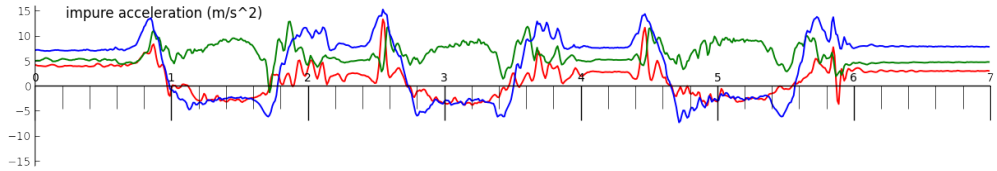


Figure 7.6: Impure acceleration (raw sensor data). Goal: take the gravity off this sensor. We can see the gravity at the beginning and end: signal is not zero even though there is no motion.

but also to the fact that a single constant error in the acceleration leads to a linear error in the velocity and a quadratic error in position. As such, the signal diverges very fast. In Figure 7.15, we show the behaviour of such integrations, with Correction scheme #2. Other correction schemes lead to similar curves. As can be seen, after seven seconds only, the position can be as inaccurate as 10 meters off!

The most common way to correct this kind of errors is by using an absolute positioning system, such as a motion tracker or a GPS, both of which are not an option for us. Fortunately for us, experiments (shown later, in Chapter 8) suggest the pure acceleration is sufficient to recognize gesture accurately.

7.3.5.7 Output

Finally, these output enable us to obtain the orientation and the pure acceleration as follows:

- pure acceleration is the impure acceleration with gravity subtracted (see Equation 7.3. It is a 3D vector.
- orientation is composed of two scalars that we call u and v:
  $u = \vec{z}.\vec{Z}$ $v = \vec{y}.\vec{Z}$

The orientation scalars u and v represent the two only useful information regarding orientation. They both take values in the range [-1,1] since they are dot products of unit vectors. The first, u, represents whether the back of the hand is facing up ($u = 1$) or down ($u = -1$), and of course intermediary value in between; while the second, v, indicates whether the back of the hand is facing backward ($v = 1$) or forward ($v = -1$).

The reason for not including a third orientation scalar is explained in Section 7.3.2. If we included it, it would indicate in which compass direction the back of the hand is pointing; since the gestures have the same meaning irrespective of the heading, we must discard such information.

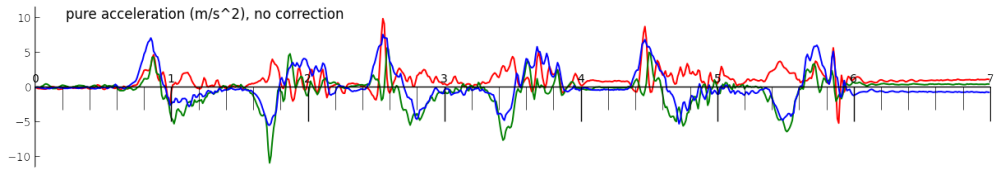


Figure 7.7: Uncorrected pure acceleration has deviated: at the end, signal is not zero. This is because estimated gravity differs from true gravity, hence subtracting estimated gravity from impure acceleration fails.

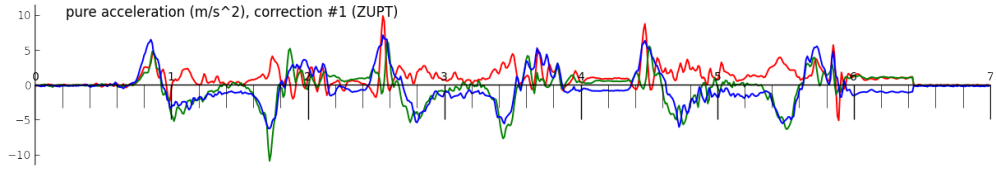


Figure 7.8: Correction #1 needs a time span without motion to recalibrate: see at the end.

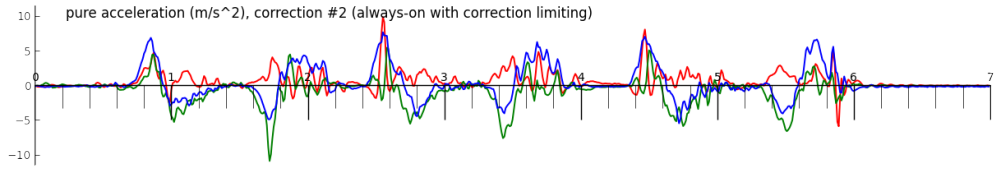


Figure 7.9: Correction #2 corrects continuously but not aggressively, which is what we want. ($\psi = .005\pi$)

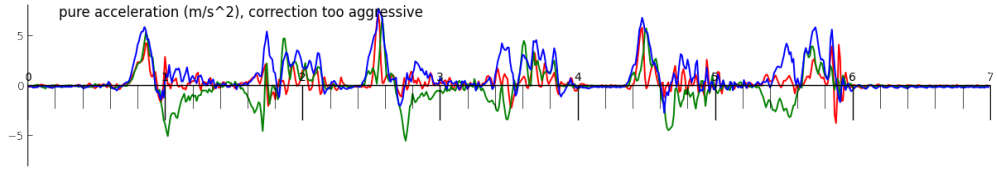


Figure 7.10: If the correction is aggressive (try to rally gravity too fast), we get the expected output at the end, but we diverge from the true value during high-motion periods.

Finally, the output of the sensor fusion stage is comprised of five scalars, three for the pure acceleration and two for the orientation.

7.3.6 Scaling

In this step, we simply multiply the signal by a constant factor so that it roughly falls in the range [-1, 1], making it suitable for the next stage based on time series recognition.

Typical acceleration of the hand generally falls in the range [-4g, +4g], so the scaling procedure for the pure acceleration is to divide by $1/(4g)$. The IMU in our custom glove can sense up to 16g of acceleration before thresholding; even though exceeding 4g might happen in rare cases, it does not cause problems in the subsequent stages. Regarding gyroscopes, we have found that dividing the original signal in degrees per second by the value $1000\pi/180$ led to satisfying results, making the signal fall in [-1, 1] most often. Last, orientation scalars

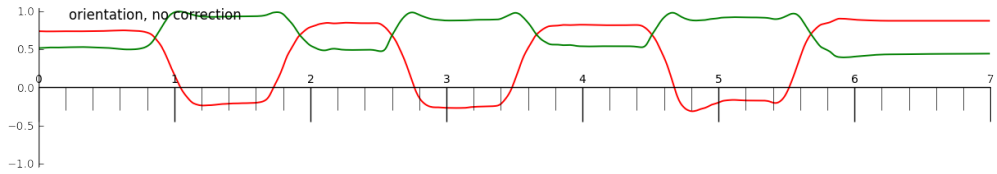


Figure 7.11: Uncorrected orientation deviates: beginning and end signals differ, but shouldn't since the begin/start positions are the same.

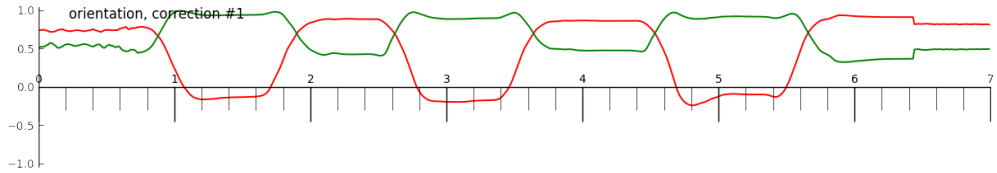


Figure 7.12: Correction #1 needs a time span without motion to recalibrate: see at the end.

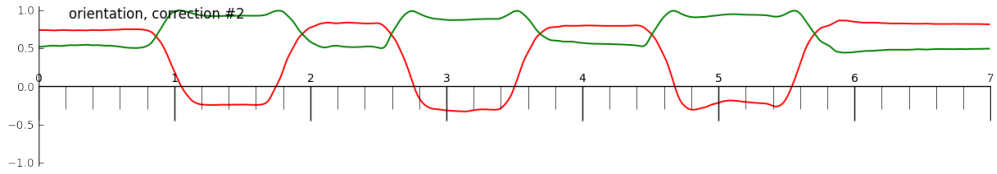


Figure 7.13: Correction #2 corrects continuously but not aggressively, which is what we want. ($\psi = .005\pi$)

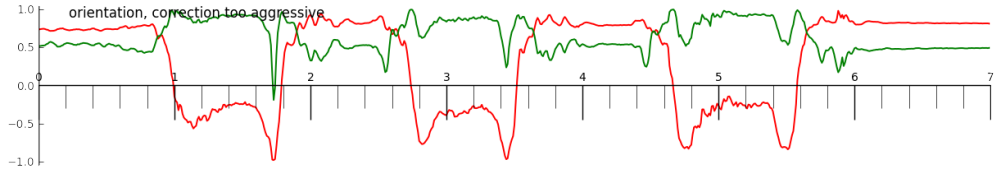


Figure 7.14: If the correction is aggressive (try to rally gravity too fast), we get the expected output at the end, but we diverge from the true value during high-motion periods.

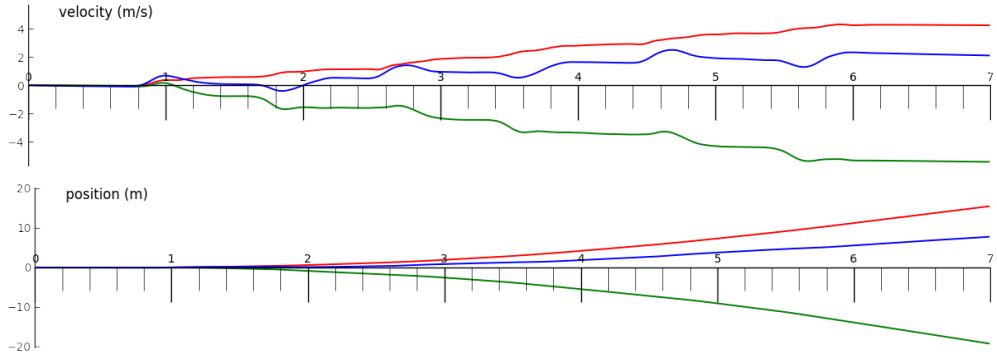


Figure 7.15: First- and second-order integration of the pure acceleration signal. Result is unusable because it diverges very fast.

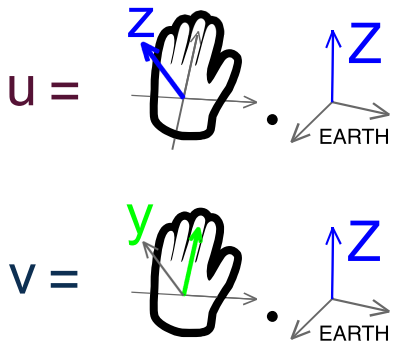


Figure 7.16: The two orientation scalars u and v, computed by dot products, indicate the rotation state of the back of hand (or equivalently, the palm).

are already in the range [-1, 1] since they are dot products of unit vectors. Finally, fingers are scaled in the range [0, 1].

Part IV

Learning gestures

Chapter 8

Classifying isolated gestures

Contents

8.1 Overview of the task: recognizing isolated gestures.

Before tackling the problem of making our gesture recognition work in streaming, we will look at a simpler challenge: recognizing segmented gestures. Here, the difference is that gestures have well-delimited boundaries. In machine learning terminology, we face a typical classification problem: indeed, we wish to learn a function whose input is a gesture and whose output is a label. The input gesture is finite and corresponds to exactly one class.

A common view of a gesture is to consider it as a succession of postures inside an interval [146, 166]. In the same spirit, we consider gestures to be a sequence of data points captured at a uniform time rate. For example, a gesture could be two seconds of observations at 100 Hz, leading to 200 data points. Those data points are vectors in a fixed-dimension real-valued vector space $\mathbb{R}^n$.

In the scientific community, such sequences of multidimensional vectors are usually referred to as "time series", because one axis of this data is the time, and we are looking at vectors describing the same measurements as they change over time. For example, time series are well suited to describe the temperature of New York each day of the year; the speed of a car over time, or motion quantities of a user's hand during a gesture. Obviously, we are interested in the last case, but the point is that there is no fundamental difference with the two former cases: time series just describe changes of the same quantities over time at an uniform rate.

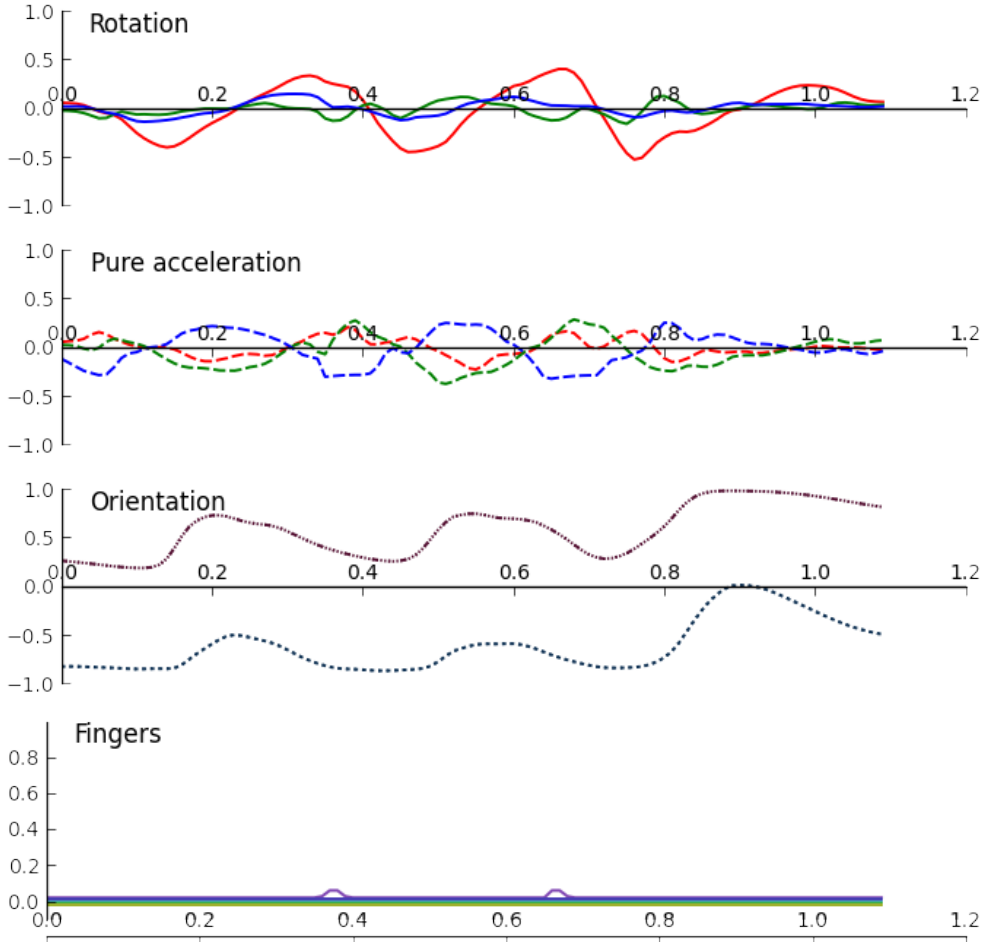


Figure 8.1: A gesture time series after motion processing and scaling. RGB represent XYZ axes. Orientation scalars u, v are respectively dotted and dash-dotted lines. Palm is up (indicated by $u = \vec{z}.\vec{Z} = -1$, i.e. Earth vector "up" and sensor vector "up" are colinear of opposite direction.). Oscillation reveals an alternating motion up and down. Fingers are all open. Therefore, it is gesture "**faster**" (Fig.).

The challenge we tackle in this section can be presented as the following maching learning task. Given a data set with labeled gestures (r, ℓ) where r is a time series (r stands for "reference") and ℓ a label, we wish to learn a function f : Timeseries $\rightarrow$ Labels, so that $f(r)$ returns the correct label ℓ for time series r. There is always one, and only one correct label for each time series, and there is a finite number of labels. Therefore, f is classifier. The two main objects f manipulates are:

- Time series: they adequately describe gestures, as they contain several quantities describing hand position and motion, changing over time at an uniform rate.
- Labels: discrete objects describing which gesture is being executed. The most common label representations are either integers or strings. In general we will associate gestures and their meaning, i.e. we will rather call a gesture "faster" than "oscillate up and down with palm up and open fingers".

Of course, the classifier f depends on the training dataset. This is common for usual machine learning tasks. However, note that the function f also depends on the meaning one

Figure 8.2: Gesture "**faster**".

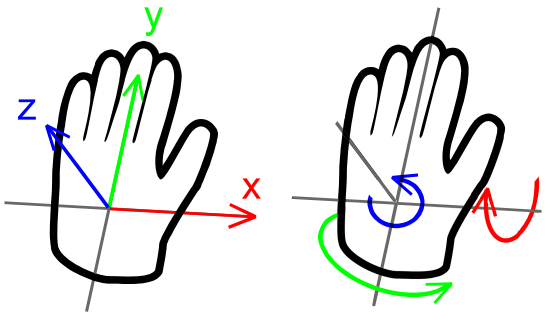


Figure 8.3: The orthonormal basis convention we use throughout this thesis (reminder from p. 91).

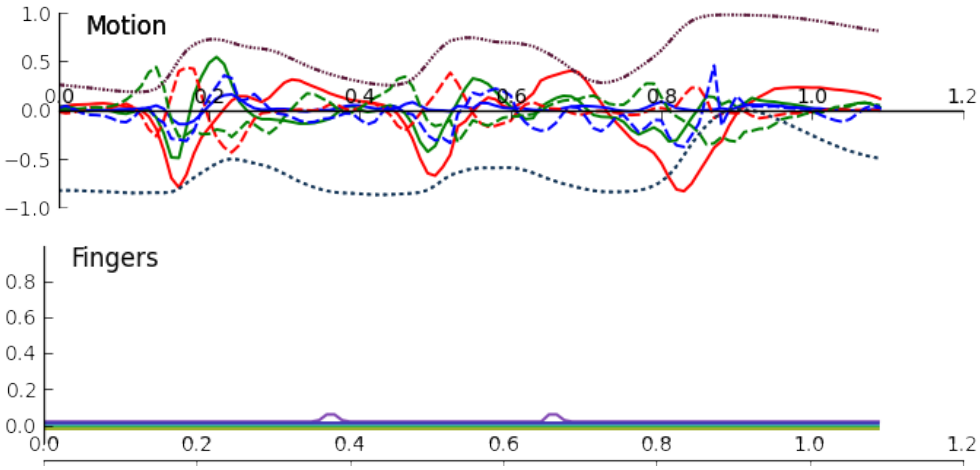


Figure 8.4: For conciseness, it is more compact to gather all motion data together. The data is the same as Fig.

assigns to each gesture, or equivalently, the gesture vocabulary that was decided ahead of time. For example, it could be that we design a first gesture vocabulary designed for robot control, where waving the hand would be labeled “start” so that the robot starts moving. Now consider a second gesture vocabulary designed for sign language, where waving hand would be labeled “hello” as a greeting between people. Here, the gesture is exactly the same, but the meaning differs according to the gesture vocabulary. This in turn impacts the data set, since recorded gestures in the training set will be labeled according to the chosen gesture dictionary.

The most important point to note here is that the gestures have been segmented. That is, the time series beginning and end correspond to the respective beginning and end of the recorded gesture. This is a simpler task than streaming recognition, where the additional

problem of *detecting* (or *spotting*) the gestures in the stream has to be addressed. We will follow up on this in Chapter 9; our goal here is to make a first investigation of how well gestures can be detected and which algorithms can be used.

8.2 Existing techniques

Different techniques to classify time series have been developed in the machine learning community. Here, we will discuss two categories:

- techniques which work on a calculation of arbitrary statistics on the time series, leading to a "summary" vector [164]
- techniques which work on the time series itself directly without first converting it.

8.2.1 Working with a "summarization" of the time series

The first category describes the algorithms transforming the time series into, a single vector which has always the same size no matter the length of the time series. This vector represents the result of different computations on the time series. For example, these can be statistics describing the average value of each measurement over time, their standard variation, their higher-order derivatives, etc. [72] There are many ways to compute a "summary" representation of the time series behavior; which summarization works best is usually found by trial and error for a particular classification task. Of course, as with any machine learning problem, such schemes highly depend on the particular task at hand, and often requires expertise on the domain. Summarization also incurs a loss of information: for example, if 100 data points are reduced into a single 3-vector such as $[\bar{r}, \sigma(r), \bar{r'}]$ (mean, standard deviation, mean of 1st order derivative), it is very likely that some information was lost in the process (in the information-theoretic sense).

The biggest advantage of summarizing the time series into a vector is the adaptability into classic machine learning algorithms. Indeed, many classification learners will best work in real-valued vector spacing, where notions such as proximity, decomposition into bases, addition, etc. are well defined. Such algorithms leverage the power of many axioms of vector spaces to compute classification functions. For example, Gillian [72] satisfactorily used SVMs to classify isolated gesture time series with statistical time domain features, but noted that such an approach can be difficult to extend to a streaming scenario. Also, Baglioni et al. [8] used 1-NN with peaks, means and medians of a two axes from a mobile phone's accelerometer, and Murao and Terada [151] used SVM over mean and variances of accelerometers for activity detection.

8.2.2 Working with the time series directly

Another approach is to work on the time series itself; however it leads to the problem that directly applying the classic machine learning techniques designed for a metric space does not usually lead to good results. If one makes the hypothesis of same-length input time series, a naive way of working without losing information is to treat the input time series as if it was an ordinary vector, and force them into an SVM or k-NN classifier, for example. Indeed, a time series of T time points each containing N dimensions can be seen either as an

$N \times T$ matrix or as a vector of dimension $N \times T$. On non trivial time series, this generally leads to poor results, mostly because learners based on vector spaces such as SVM or k-NN do not take into account the order of the scalars in the vector, and thus lose the information that subsequent points of the time series are almost always correlated.

Therefore, we need techniques which deal with time series by taking into account the order of points. In the following, we will discuss two techniques which are built to deal specifically with time series:

- elastic distances
- neural networks

8.2.2.1 Elastic distances

8.2.2.1.1 Distance functions and metric spaces

It is common to use classification algorithms on vector space $\mathbb{R}^n$, which is also a metric space. Indeed, it is possible to attach a positive function d known as "distance", which satisfies several axioms, namely for all $x, y, z \in \mathbb{R}^n$:

- identity of indiscernibles: $d(x, y) = 0$ iff $x = y$
- triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$
- symmetry: $d(x, y) = d(y, x)$

If one works with a normed vector space, which $\mathbb{R}^n$ is, one can derive a distance function by taking the norm of the difference between two vectors:

$$d(x, y) = \|x - y\| \tag{8.1}$$

The connection between both is strong enough that writers often write the name of the norm to describe the name of the underlying distance.

A classic norm on $\mathbb{R}^n$ is the Euclidean norm. When derived into a distance, it maps well to our human perception of distance; for $\mathbb{R}^3$, Euclidean distance describes exactly the notion of distance as we work in the everyday life. The Euclidean norm is also written L^2, where the 2 refers to the exponent in the summation process.

$$d(x, y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{8.2}$$

Another common choice of norm is the L^1 norm, giving the L^1 distance also called "Manhattan distance" because one can imagine it in a grid-shaped city-like environment. It is used, for example, in video games where the 2D world is represented as a grid. More generally, on any $\mathbb{R}^n$ space, there exists a whole family of L^p distances:

$$d(x, y) = \|x - y\|_p = \left(\sum_{i=1}^{n} (x_i - y_i)^p \right)^{\frac{1}{p}} \tag{8.3}$$

L^p norms and associated distances are directly applicable on $\mathbb{R}^n$ and have various properties, however they are not quite relevant to our topic here. These distances work well when objects to be compared belong to a vector space. For instance, it is possible to use the Euclidean (L^2) distance with classifier 1-NN. It can be applied on a summarization [8] or on the time series directly, as we shall see below.

8.2.2.1.2 Distance between time series

Even though time series are complicated objects on which applying an L^p norm doesn't work well, they still are well-defined objects. An easy way to compare two same-length time series would be applying the Euclidean distance directly. In principle, this amounts to computing the difference between pairs of aligned points in the time series and summing them. That is, for two time series $r_1 = (r_1[1], r_1[2], \ldots, r_1[l])$ and $r_2 = (r_2[1], r_2[2], \ldots r_2[l])$:

$$\begin{aligned} |r_1| &= |r_2| = l \\ D(r_1, r_2) &= \sum_{i=1}^{l} \|r_1[i] - r_2[i]\| \end{aligned} \tag{8.4}$$

If the time series are well aligned, this works as desired, however a tiny offset in the time description of the time series makes it mess up with the alignment [110]. As a result, many point-to-point differences are high, summing to a large distance even when we would intuitively say that two time series look very similar.

In the 1970s, Sakoe and Chiba [189] proposed an algorithm named DTW (Dynamic Time Warping) to compare time series by considering some elasticity on the temporal axis, thereby allowing time series which are similar yet stretched to be still reported as close one to another. Since then, DTW has been widely used for a variety of applications, including gesture-related ones [**hartmann_gesture_2010** , 6, 35, 36, 51, 72, 112, 181, 190].

The novelty of DTW was the use of a technique to get rid of a static alignment as in the Euclidean computation. Instead of working with a fixed point-to-point correspondence, the idea behind "time-warping" is to discover an optimal alignment which would better match the variations of both time series together, even if they don't have the same time offsets.

A DTW computation between two time series of possibly different lengths $r_1 = (r_1[1], r_1[2], \ldots, r_1[l_1])$ and $r_2 = (r_2[1], r_2[2], \ldots r_2[l_2])$ unfolds as follows:

Algorithm 1 DTW

```
1: procedure DTW(r_1, r_2)                          ▷ (r_1 and r_2 are 1-indexed; A is 0-indexed)
2:     A = new matrix [0..l_1, 0..l_2]              ▷ accumulated costs matrix
3:     A[0, .] = A[., 0] = ∞                        ▷ base conditions of recursive equation
4:     A[0, 0] = 0                                  ▷ force warping path to start at (0, 0)
5:     for i_1 = 1 to l_1 do
6:         for i_2 = 1 to l_2 do
7:             A[i_1, i_2] = d(r_1[i_1], r_2[i_2]) + min(A[i_1−1, i_2   ],
8:                                                      A[i_1   , i_2−1],
9:                                                      A[i_1−1, i_2−1])
10:        end for
11:    end for
12:    return A[l_1, l_2]      ▷ accumulated cost of minimal warping path from (0, 0) to (l_1, l_2)
13: end procedure
```

The main advantages of DTW are:

- resilience to stretched signals thanks to time warping;
- provides a numerical measure of the dissimilarity (good for comparing several distances).

On the other hand, DTW has a few weaknesses:

- somewhat slow to compute (quadratic complexity);
- not a true distance (does not satisfy triangle inequality).

The dynamic programming algorithm described above, which is the most common version to compute the shortest path in the DTW matrix of pairwise costs, has a complexity $\mathcal{O}(l_1 l_2)$ that we can understandably call quadratic. This stems from the double loop computation taking place on both time indexes i_1 and i_2.

8.2.2.1.3 Using elastic distances to classify time series

A single notion of distance does not provide us directly with a learning algorithm, that is, a technique able to compute a classification function from the data. Indeed, a distance function only supplies a quantitative way to tell how dissimilar two time series are.

Fortunately, several classification rules work in conjunction with a distance function in order to make classification decisions. In this work, we will talk about two rules:

- k-NN (k-Nearest Neighbor), and 1-NN (First-Nearest Neighbor, a special case of k-NN)
- SVM (Support Vector Machines)

k-NN In essence, k-NN associates an unknown data point with the most voted label of the k nearest neighbors. Thus, training is very simple, one just needs to remember the points of the training set along with their label. The computations take place during the testing

phase, where a new input point is compared to all training set data points. Distances are sorted and the k smallest distances refer to labels, the most numerous of which is returned as the output label.

More precisely, consider the training set of pairs (x_i, y_i) made of data points x_i and associated labels y_i. During the training phase, no operation is required, apart from storing the training set in memory. During the testing phase, we are given an input data point x for which we would like to find out the unknown label y. The procedure in k-NN amounts to:

- computing distances with all points: $d_i = d(x, x_i)$
- retrieving the k closest data points, i.e. the index points $i_1, i_2, \ldots, i_k$ with the lowest distances $d_{i_1}, d_{i_2} \ldots d_{i_k}$ among all d_is
- return the most represented label y for those k data points

In practice, k-NN is a very simple yet very powerful decision rule.

A special case of k-NN is the first-nearest neighbor rule, where $k = 1$. We will denote it as 1-NN. In this case, the rule just amounts to returning the label of the training set instance minimizing the distance function, hence the name "first nearest neighbor".

In our case, using the elastic distance to work with k-NN or 1-NN is relatively straightforward. If suffices to plug DTW as the distance function of the decision rule; when we need to classify an unknown gesture, that is, an unknown time series, we compute DTW for all time series and return the label of the closest element. In other words, the steps for the 1-NN decision rule with DTW are as follows:

- for each instance x_i of training set:
  - compute DTW between x and x_i
  - remember i as i_{min} if x_i if is the smallest distance so far
- return label of instance $x_{i_{\text{min}}}$ with the smallest distance

Note that in order to carry those multiple DTW computations faster, a collection of enhancements known as "lower bounds" have been designed. They provide confidence that an in-progress computation is going to be higher than the best-so-far result, in terms of 1-NN classification. Hence, one can abort an on-going computation instead of carrying on with useless inner distance computations which are usually the bottleneck in time; this leads to considerable acceleration [110].

SVM SVMs (Support Vector Machines) provide another classification learning rule. The fundamental technique behind SVM is more complex than the case of k-NN. In a nutshell, a multiclass SVM is able to classify points in a vector space by finding a separating hyperplane in higher-dimensional spaces. Several parameters must be chosen:

- a kernel k: positive definite function mapping data into a higher-dimensional space where inputs are mostly linearly separable;
- a soft-margin parameter C: indicates the tolerance of the SVM for non linearly separable data.

Often, an SVM is used in combination with the Gaussian kernel, which expresses dissimilarity of two data points x and y as such:

$$k(x,y) = \exp\left(\frac{d(x,y)}{\sigma^2}\right) \tag{8.5}$$

In the case of the Gaussian kernel, an SVM must be given two hyperparameters before training: C and σ. In general a joint search of both parameters is executed and classification accuracy is computed thanks to cross-validation.

Although other kernels exist, such as the polynomial kernel $k(x,y) = (x,y)^d$, the hyperbolic tangent $k(x,y) = \tanh(\kappa x.y + c)$, we will not discuss them further.

The most striking feature of the Gaussian kernel is the use of a distance function d which expresses the dissimilarity. For typical data in $\mathbb{R}^n$, one would use an L^p distance, in general the L^2 distance since it is the most natural.

However, time series are not simple data points and they don't live in $\mathbb{R}^n$, so we cannot apply the L^2 distance on them (or rather, we can, but it leads to poor results due to the problem of Euclidean distance used with time series highlighted above). On the other hand, it is possible to substitute the distance d with DTW to compute the dissimilarity between two time series (we write them x, y here since they are considered as data points):

$$k(x,y) = \exp\left(\frac{\mathrm{DTW}(x,y)}{\sigma^2}\right) \tag{8.6}$$

However, a problem arises: the Gaussian kernel with DTW as a distance function is not positive-definite [136], and thus it breaks the hypotheses of a traditional SVM. Indeed, equations behind the SVM theory reduce the learning rule to the resolution of an associated quadratic problem (QP) which admits a unique solution, provided the kernel is definite-positive [233]. If not, it is a general minimization problem and the QP-based algorithms are not guaranteed to converge anymore, since they can get stuck in a local minimum, possibly far from the global minimum.

Some techniques exist to mitigate this issue, such as positive-definite kernels based on DTW. In particular, positive-definite time series kernel K-DTW [136] has been successful for time series classifications; the basic operation takes place by computing the probabilities of all paths instead of just the minimal path for each cell. After rescaling, the training dissimilarity matrix is positive-definite and usable DTW in the context of time series classification while satisfying all necessary hypotheses.

Although we conducted some experiments over SVM [53, 54], but ultimately we chose to pursue the path of 1-NN over DTW:

- SVM over DTW without renormalizing the kernel leads to poor results in time series classifications [136];
- SVM over K-DTW gives better results in some cases [136]; it can exhibit noisy behavior at times [53];
- 1-NN over DTW behaves more predictably [53], and is parameterless compared to SVM needing a 2D grid search;
- 1-NN over DTW is extended naturally from a classification rule to an outlier detection rule for streaming (Chap. 9).

8.2.2.2 Neural networks

While neural networks can take a variety of shapes and architecture, in general they work by stacking multiple layers of "neurons" or "units", composed of simple functions with a very dense interconnexion. Each unit's behavior can be changed thanks to some parameters known as weights; given some input and expected output, one can train the network by using backpropagation techniques, which ultimately compute gradient information indicating the magnitude for each parameter change to reach a better accuracy.

Neural networks, and in particular deep learning techniques [120], were able to tackle many interesting problems such as speech recognition, image classification, etc. and surpassed the state of the art in many domains [193]. It is therefore very natural that we consider the pertinence of neural network for time series classification.

In time series related problems, encouraging results have surfaced recently, as highlighted by solving problems such as handwritten text recognition [75, 76], speech recognition [74], text synthesis [106]... The main technological breakthroughs behind these works are RNNs (Recurrent Neural Network) and LSTMs (Long Short-Term Memory) which treat streams and time series as first-class objects by design. Their appeal lies the network's ability to remember patterns that happened earlier and locking this information (thanks to "gates") so that it is not forgotten while other parts of the network converge.

However deep learning techniques have downsides too. First, neural network techniques are very power-hungry; nowadays one would use one or several GPUs (Graphical Processing Units) to make a deep network converge. The more complicated the model, the more processing power it needs. Also, the processing power increases with the amount of training data. Speaking of data quantity, it is necessary to have huge amounts of data in order to yield significant results. Otherwise, the network behavior might be unstable; even though neural networks work well in practice, they lack solid theoretical basis and it is difficult to have an idea of their behavior with truly unknown data when the training set is sparse.

Our gesture recognition application has the following properties:

- it should not use too much processing power due to the embedded requirement of the robotics application;
- very few instances of each gesture are available ("small data");
- erratic behavior is highly undesirable.

For these reasons we have chosen not to use deep neural techniques. First, we do not have much processing power available, although this could be possibly mitigated by using the training power of a GPU on the machine running the training GUI. The "small data" problem is much more embarrassing; indeed, when only 2 or 3 instances of some gestures are given, it is almost impossible for the network to converge correctly. This might lead to erratic results, because even though many local optima are available and correctly suit the training data, the low number of samples is not representative of the distribution of the training space, which can lead to unexpected behavior. Therefore, DTW over 1-NN seems to have better arguments in terms of rapidity and expectation of stability with regards to the "small data" problem because it always finds a neighbor.

8.3 Experiments

In this section, we show different experiments whose goal was to certify that DTW is indeed a serious contender to distinguish gestures. As specified in the beginning of this chapter, these experiments only consider the gesture *classification* problem, that is, gestures are well segmented and analyzed with comparison one to another. The joint detection and classification problem will be tacked in the next chapter.

8.3.1 Experiment 1. One user, static

In this experiment, one subject performed gestures in a separated manner. Each time, only one gesture at a time was asked for a single capture round, and the capture begins and ends with a resting position from the hand. Though we could use the captures this way, we segmented them further in order to keep only the interesting part at the middle of each capture: either the motion in the case of a dynamic gesture, or in the case of a static gesture, the meaningful position (i.e. different from the resting position). The segmentation was performed offline by manual "expert" analysis of the gesture files.

For each gesture, we made 4 separated captures.

The gestures were:

- go (3x): starting from an open hand upwards, palm facing the cheek, swing the arm forwards. (Swing repeated 3 times in a row.)
- stop "open hand": open hand upwards, palm facing the cheek
- lock: all fingers closed except pinky, back of the hand facing up. Quickly rotate wrist inwards to bring the pinky inside. Back of the hand should then face down.
- unlock: all fingers closed except pinky, back of the hand facing down. Quickly rotate wrist outwards to bring the pinky outside. Back of the hand should then face up. (It is the reverse of the "lock" motion.)
- steer: "L-shape" with thumb and index; other 3 fingers closed. Back of hand facing left. Thumb should point up, index forward. No motion.
- left: "L-shape" with thumb and index; other 3 fingers closed. Back of hand facing down. Thumb should point left, index forward. No motion.
- right: "L-shape" with thumb and index; other 3 fingers closed. Back of hand facing up. Thumb should point right, index forward. No motion.
- yes: "O-shape" with finger and thumb touching extremities. Other fingers open. Forearm pointing upwards.
- no (3x): open hand, palm facing down, back facing up; by balancing the forearm in transverse plane (the hand's plane), gently swing left and right. (Swing repeated 3 times.)
- slower (3x): open hand, palm facing down, back facing up; swing moderately up and down (sagittal plane). (Swing repeated 3 times in a row.)
- faster (3x): open hand, palm facing up, back facing down, swing moderately up and down (sagittal plane). (Swing repeated 3 times in a row.)

The gestures having the mention "3x" correspond to dynamic gestures for which repetition is intended to increase confidence in the command. For example, gesturing "go, go,

go!" leads to more confidence than a single "go" gesture. In this dataset we find two gestures which are dynamic yet do not accomodate to repetitions well: the "unlock" and "lock" gestures, which are opposed to one another, so repeating "unlock, unlock" would actually give the sequence "unlock-lock-unlock". The other gestures are sustained gestures with no motion, which is why repeating makes no sense for them. Sustained and repetitive gestures are discussed in Chapter 4 page 57.

In order to evaluate the performance of DTW on this dataset, we computed its similarity matrix, i.e. the pairwise matrix of distances between all examples. It is a square matrix, symmetrical because the i-th row corresponds to the same time series (gesture) as the i-th column. Formally:

$$G[i,j] = D(x[i], x[j]) \tag{8.7}$$

where G is the dissimilarity matrix, D is the time series distance algorithm or dissimilarity function (DTW, for example), and x is the dataset (i.e. $x[i]$ is the i-th gesture of the dataset).

We kept ordering intact in the dataset so as to facilitate visual cluster analysis: we thus expect 4×4 clusters. By using a distance algorithm such as DTW, the goal is to obtain both small intra-cluster and high extra-cluster distances. In other words, similar gestures should have a small distance (not very dissimilar), while different gestures should have a high distance (very dissimilar).

In Fig. 8.16, we can observe the dissimilarity matrix for DTW computation between all examples of the dataset. As expected, we see the structure of clusters as 4×4 sub-matrices; this suggests DTW performs well on our gesture time series, and also validates our preprocessing as a good source of feature information. Indeed, obtaining the orientation is extremely important to distinguish gestures such as "faster" and "slower", both of which are basically the same motion with a different orientation.

Some clusters show some small weaknesses, because of comparisons between several different DTW results obtained for time series of different length. DTW will sum more costs if there are more points in the time series. Therefore a normalization scheme is necessary so that the distance is length-algorithm. To do so, we used a simple yet effective technique: dividing by the diagonal of the time series length, i.e. for two time series $r_1 = (r_1[i_1])_{1 \leq i_1 \leq l_1}, r_2 = (r_2[i_2])_{1 \leq i_2 \leq l_2}$, of respective lengths l_1 and l_2, we compute DTW* (pronounced DTW-star) as follows:

$$\text{DTW*}[r_1, r_2] = \frac{\text{DTW}[r_1, r_2]}{\sqrt{l_1^2 + l_2^2}} \tag{8.8}$$

the denominator of which provides a good order of magnitude to both time series length, because it represents the length of the diagonal in the accumulated cost matrix in the computation of DTW.

Obtaining the dissimilarity matrix for the same dataset with DTW* computation shows better cluster, removing inaccuracies due to time series with different lengths.

All dissimilarity matrices in this chapter are colored with an exponential scale to help visual analysis.

In order to analyse the performance of 1-NN classification over this dataset, we proceeded to a leave-one-out validation. For each instance i, we retrieved the label of the closest example with respect to the chosen distance function (DTW or DTW*) and returned its label. (Of course we excluded example i itself which would always be the nearest neighbor since $\mathrm{DTW}(v, v) = 0$ for any time series v.)

Distance	accuracy
DTW	100%
DTW*	100%

Table 8.1: DTW (length-dependent) and DTW* (length-agnostic) recognize gestures accurately.

From this, we can draw the following conclusions:

1. DTW is excellent to distinguish between different gestures, and also to obtain an unknown gesture's label in nearest-neighbor classification (1-NN). This was expected, as DTW over 1-NN is already known to work well with time series [72].
2. One might wonder was DTW* was needed at all since unmodified (raw) DTW was already at 100% accuracy. In reality the gestures here were still controlled and lengths were not too dissimilar, but we intend to leave full control of gestures to the user, including allowing any-length gestures; thus DTW cannot be left unnormalized.

Regarding speed, these experiments were run in pure Python (Numpy for the representation but not for accelerated algorithms). Time of execution was a real concern, as the DTW dissimilarity matrix above took as long as 83 seconds to be computed, for 1+2+...+44 = 990 DTW computations, thereby averaging around 12 DTW computations per second. This is too slow to run well in a multi-gesture, real-time setting as we intend to in the next chapter. However, it appeared that implementing the same DTW code in carefully-optimized C++ made the same computations run orders of magnitude faster and rendered real-time DTW computation possible. Indeed, computing this same dissimilarity matrix in C++ took a bare 0.12 second, providing a 700x speedup from the pure Python version.

8.3.2 Experiment 2. One user, in motion

Since mobility (walking, running) is an important constraint, it was important for us to investigate its effect early. In this experiment, we sought to explore the time series of gestures performed in the context of some body motion. In order to achieve this, the operator was walking during all captures.

This time, the operator was asked to execute several gestures in a single capture; afterwards, the captures were analyzed manually ("expert analysis") and segmented accordingly. Because some captures contain several gestures, a single capture can give rise to multiple gesture time series.

In the following list, each line corresponds to a single capture scenario (that was repeated 4 times separately), each of which have been manually splitted offline. Commas in each line separate isolate different gestures in the same capture.

- faster 1x

- faster 2x
- faster 3x
- slower 1x
- slower 2x
- slower 3x
- go 1x
- go 2x
- go 3x
- stop "open hand"
- stop "closed hand"
- unlock
- lock
- yes
- no 3x
- steer
- left
- right
- nothing (no particular gesture was made)

Please note that a "faster 1x" gesture is allowed to match with a "faster 3x", because they have the same intent and thus the same label. This is on purpose, since it is a repetitive gesture as described in Chapter 4. Thus the robot will keep acting (here, increasing speed) until the gesture repetition is over, indicating the end of user intent.

At this point we also started to investigate the effect of a closed-hand "stop" over an open-hand "stop", as concerns over the open-hand "stop" started to surface. Indeed, we had just identified that "stop"-open being a subgesture of "go" could cause some trouble.

In Figures 8.19 and 8.20, we can see a mostly successful recognition of the different clusters. In more detail, we find that:

- cluster structure is still apparent, indicating good recognition by DTW/DTW*
- 1x, 2x, 3x gestures seem to match quite well between themselves. In DTW*, the difference fade away, due to length-normalization which is desired (the label must be recognized independently of how many times the gesture was done).
- length renormalization boosts the cluster structures and puts every gesture class on the same scale. See for example the cluster of "no".
- a problem arises with "stop-open hand", it seems to merge with the cluster of "go" gestures. As we feared, this gesture begins to be problematic and might cause issues of confusion between "go" and "stop". In contrast, "stop-closed hand" has no such problem, which is a good indication that we should keep this gesture for the final dictionary.

All in all, recognition results are very positive with both DTW and DTW*. This inspires confidence for the use of DTW in the next phase of our project, stream recognition. More importantly, this experiment also successfully validates the potential of a DTW-based system to function when the operator is walking, which is in itself a step forward in terms of mobility.

Figure 8.18: "**stop**" with a closed hand was introduced to investigate if is more suitable than "stop" with an open hand, with regards to subgesture conflict with "go".

8.3.3 Experiment 3. Two users, static

This experiment aimed to obtain basic results of the behavior of DTW and DTW* for cross-user training.

In order to obtain such information, we asked two different users to perform the same gestures. We will call those users A and B. The condition is standing for both of them: they were not walking or running. Each gestures was performed separately with expert segmentation when it was necessary. The gestures performed were:

- faster 1x
- faster 2x
- faster 3x
- stop-closed
- go 1x
- go 2x
- go 3x
- lock
- no 1x
- no 2x
- no 3x
- slower 1x
- slower 2x
- slower 3x
- left
- steer
- right
- stop-open
- unlock
- yes

Each gesture was repeated four times, for each subject A and B. Unfortunately, post-hoc analysis revealed three captures for subject B were faulty. In order to keep the 4-clusters intact for visual inspection of the dissimilarity matrices, we kept them in the dataset as null time series. This will be apparent in the dissimilarity matrices below. Captures from user A are all correct.

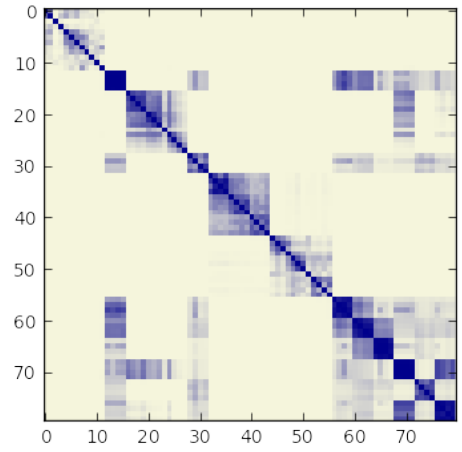


Figure 8.21: DTW User A

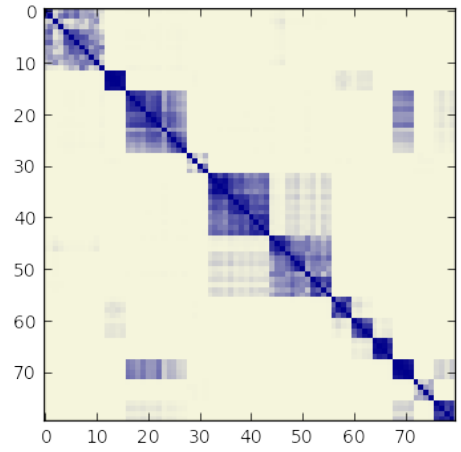


Figure 8.22: DTW* User A

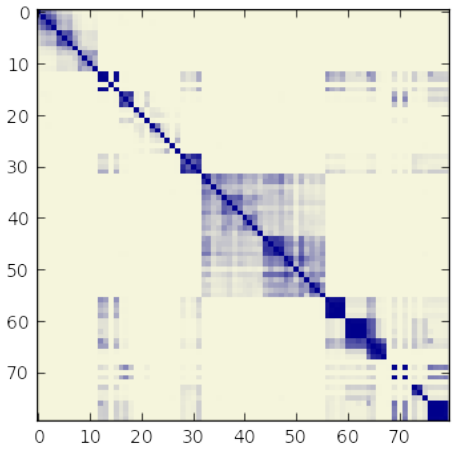


Figure 8.23: DTW User B

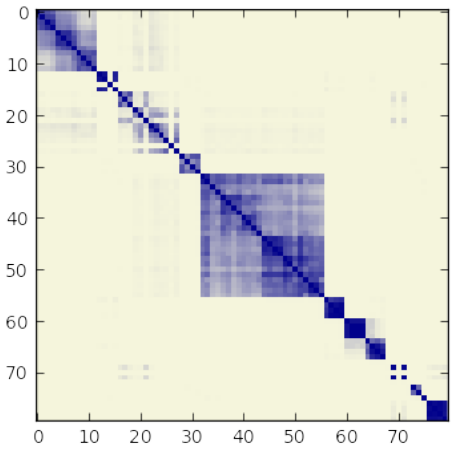


Figure 8.24: DTW* User B

First, we will start by examinating the one-subject dissimilarity matrices as we did for the previous experiments. In Figures 8.21 to 8.24, we show the DTW and DTW* dissimilarity matrices of subject A and subject B without cross-subject interaction, as well as Table 8.3 which shows the accuracy for each subject. We can make the following observations:

- DTW* matrices reveal much better cluster structure (less out-of-cluster low distances) than DTW. From now on, the interest in DTW* should be pretty clear. Indeed the distribution of length in the time series has more variation than in previous experiments, as we illustrate in Table 8.2.
- the main confusion of DTW* matrix for user A is shown by the "trails" out of the clusters diagonal. They highlight confusion between the "go" (1x, 2x, 3x) and the "stop-open" gestures. As was hinted in Experiment 2, we have yet another indication that if DTW* is to be retained, we need to avoid the "stop-open" gesture and use the "stop-close" gesture (which, as shown here, is not problematic and well detected within its cluster).
- User A obtains excellent results, but User B has some errors. The accuracy of 92.5% = 74/80 comes from 3 capture errors (the 3 missing gestures for which capture has failed)

and 3 "real" errors for which the time series exists but does not match a correct nearest neighbor. After further analysis, it seems that the 3 real errors come from the user not having performed the gesture well enough. This is an important point in general, because errors are sometimes not due to the classification engine but rather to user mistakes. We will address bad input in training in Chapter 10.

percentiles	5-th	median	95-th
Previous experiment in static	10	45	155
This experiment, User A	10	141	272
This experiment, User B	10	155	292

Table 8.2: Distribution of the time series lengths. In this experiment we can see that short time series still exist, but longer time series have appeared, which highlights the need for a length-agnostic distance such as DTW*.

Distance	Accuracy
User A DTW	100%
User A DTW*	100%
User B DTW	92.5%
User B DTW*	92.5%

Table 8.3: Each user's own accuracy results on leave-one-out validation

In conclusion, analysis from both subjects alone continue to validate DTW* as a successful recognition algorithm. We also keep seing "stop" with an open hand as a gesture to avoid due to confusion with "go".

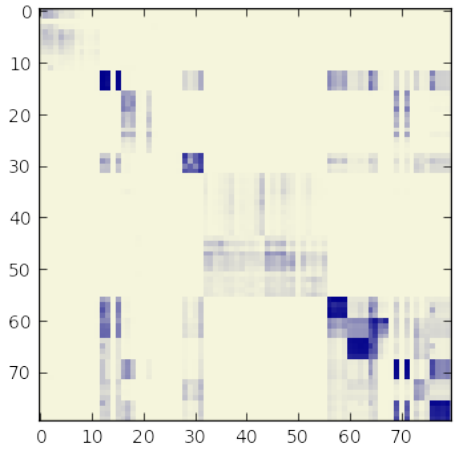


Figure 8.25: DTW between two users

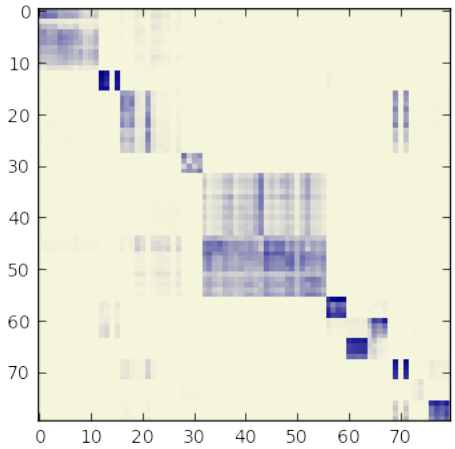


Figure 8.26: DTW* between two users

We will now discuss to the intra-subject classification results. In Figures 8.25 and 8.26, we have displayed the matrix of distances where all gestures of subject A are shown on the rows, and subject B on the columns. Of course, the matrix is no longer symmetrical. The main observations are the following:

- DTW is barely useful anymore: many dark spots appear in off-clusters areas of the matrix. Here DTW* seems to handle the gesture time series much better.

- The distances seem to blur between both users. We can still see some cluster structures, but there is in particular a very high confusion between the "no" (1x, 2x, 3x) and "slower" (1x, 2x, 3x) gestures. Also, two clusters seem to have been mixed up: the "lock" and "unclock" gestures have most likely been interpreted differently by both users (i.e. one of them did the gesture, which consists of a rotation, in the wrong direction).
- Table 8.4 indicates that DTW* still works better in both cases. Furthermore, we obtain better results with a combination [bad teacher, good performer] than with a combination [good teacher, bad performer]. We cannot claim those results transpose in a more general manner, though.

	Train A / Test B	Train B / Test A
DTW	92.5%	68.8%
DTW*	98.8%	70.0%

Table 8.4: Cross-accuracy between users. Results depend highly on the training performer.

The cross-subject results hint at the fact that the system could be handed from one user to another, without proper training from the latter; in some cases, the recognition results can stay elevated (train B, test A: 98.8%) but in others, the recognition rates fall dramatically (70% in the opposite case). In conclusion, *a proper training from the intended end-user should always be carried out by himself or herself*, in order to maximize the accuracy.

Figure 8.5: **go**

Figure 8.6: **stop** (open)

Figure 8.7: **lock**

Figure 8.8: **unlock**

Figure 8.9: **steer**

Figure 8.10: **left**

Figure 8.11: **right**

Figure 8.12: **yes**

Figure 8.13: **no**

Figure 8.14: **slower**

Figure 8.15: **faster**

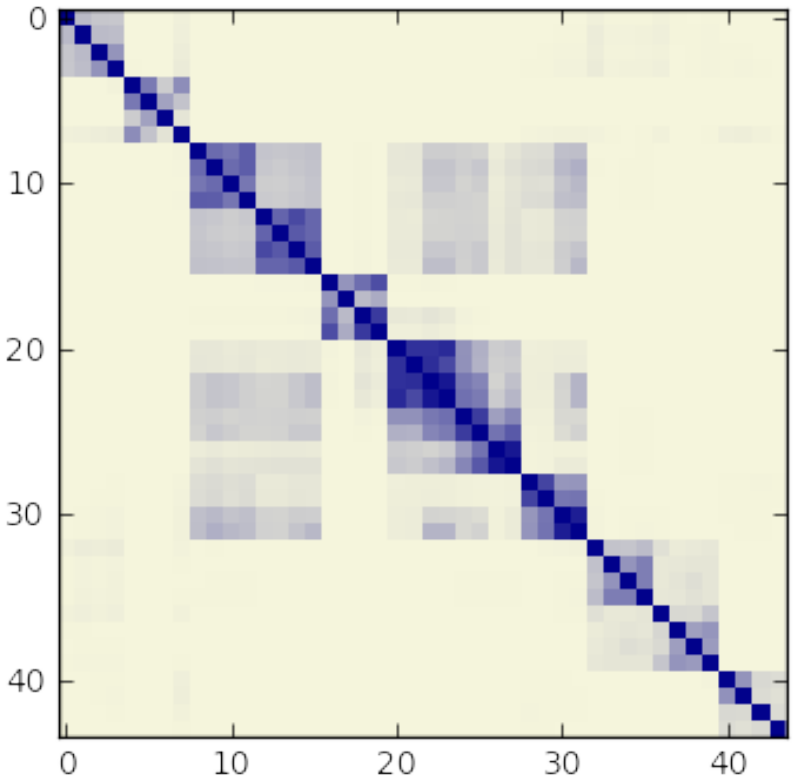


Figure 8.16: DTW dissimilarity matrix. 4-clusters are well detected by DTW. However, for some gesture classes DTW does not behave well, as shown by the out-of-diagonal clusters.

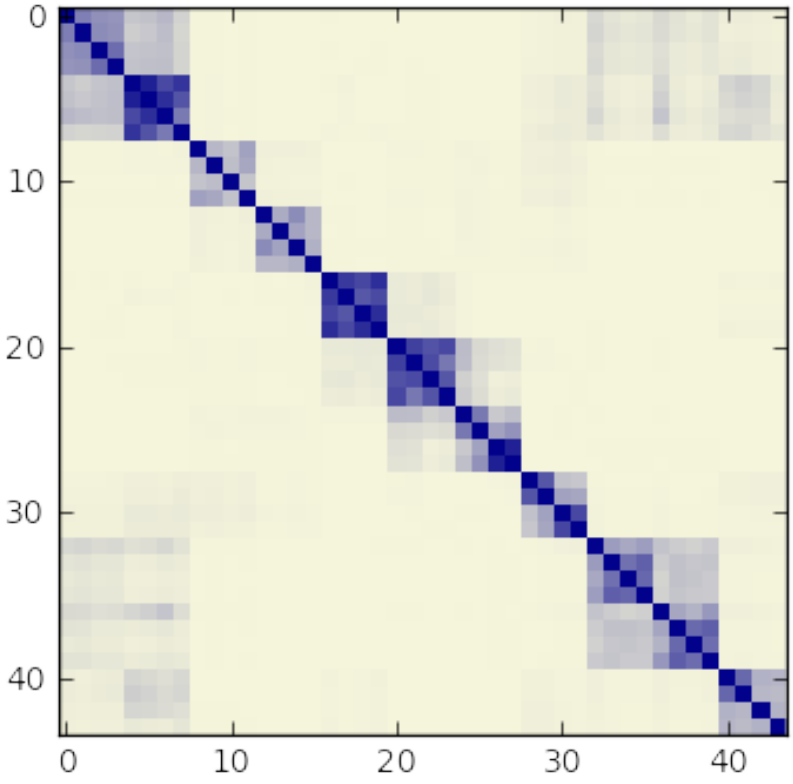


Figure 8.17: DTW* dissimilarity matrix. Normalizing with respect to time series lengths erases the out-of-diagonal clusters.

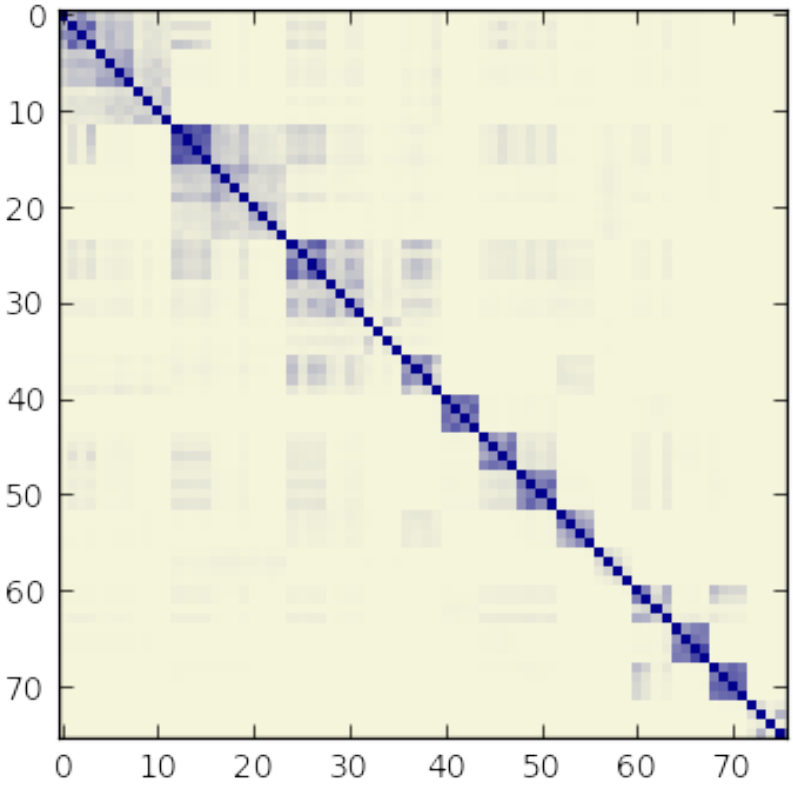


Figure 8.19: DTW in motion

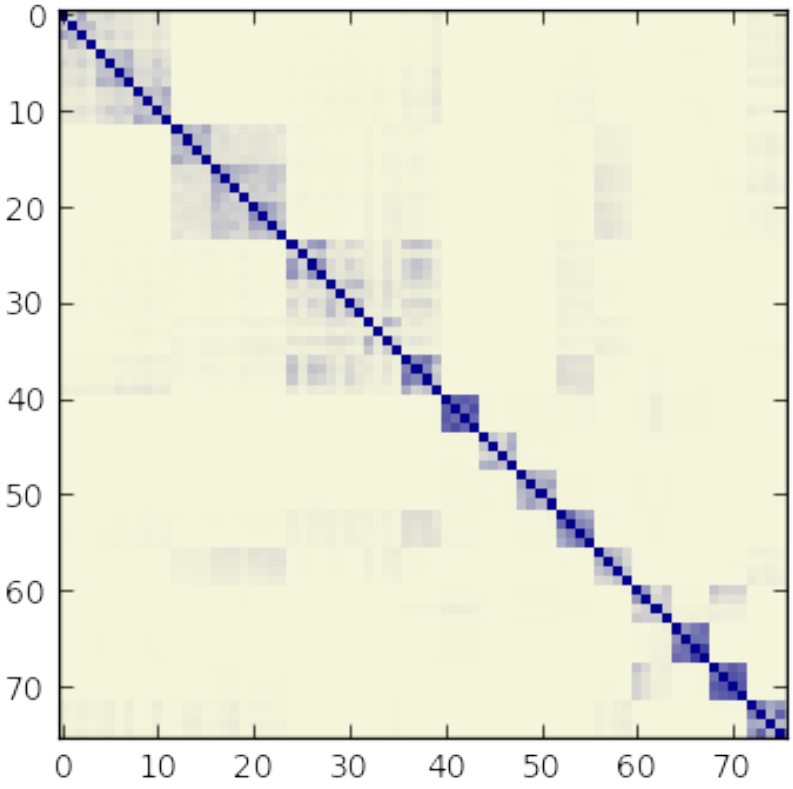


Figure 8.20: DTW* in motion

Chapter 9

Gesture recognition in real-time streaming

Contents

In the previous chapter, we discussed the "easy" task of recognizing well-segmented time series. However, it does not fully address the problem of recognizing gesture in a real-time setup. Indeed, the input data is not a well separated set of gestures to be recognized, but rather a stream of endless, unseparated motion data points.

In this chapter, we describe how we can extend ideas from the previous chapter into a stream-enabled technique for recognizing gestures. Then, we discuss how to evaluate the performance of our stream classifier, which is also a non-trivial task compared to simple classifier. When the evaluation procedure is agreed upon, we move on to a full experiment with more than 200 gesture instances, in which we prove the performance of our technique in streaming.

9.1 Extending isolated recognition to a stream

The previous chapter has uncovered a lot of ground for stream recognition. Most relevant is the encouraging ability of length-normalized, elastic distance DTW* coupled with the 1-NN classifier, for classifying isolated gesture time series. How could we naturally extend both 1-NN and DTW* in a streaming fashion?

9.1.1 Extending 1-NN

9.1.1.1 Problem: a classifier always returns a label

By definition, a clasifier such as 1-NN will always return a label for any input data. Although this is good when there is indeed a gesture to be detected, it falls short when there is no gesture to detect because it will still try to return the label of the class best matching the data; even if "best" means "the least worse".

In order to mitigate this issue, here are two possible solutions:

- introduce a "blank" label representing a complementary class of all non-gestures [57, 72];
- turn the problem into a "reversed outlier detection" question so that a class is only triggered when there is enough certainty it matches well.

9.1.1.2 Classification with a blank class

The first approach relies on the creation of a new class, the "blank" class, which is expected to trigger when no gesture is performed. While it might be made to work, the problem with this approach is that the blank space is generally enormous compared to the "normal" class spaces. Indeed the blank space is comprised, in our case, of all gesture which do not represent a class. It seems seldom possible to train the system on all non-gestures, and especially difficult to ask an user to perform all possible non-gestures. Therefore there will be holes in the blank space, meaning that some gestures happening in operation phase will be totally unknown and not represented by a blank instance even though no normal class should match.

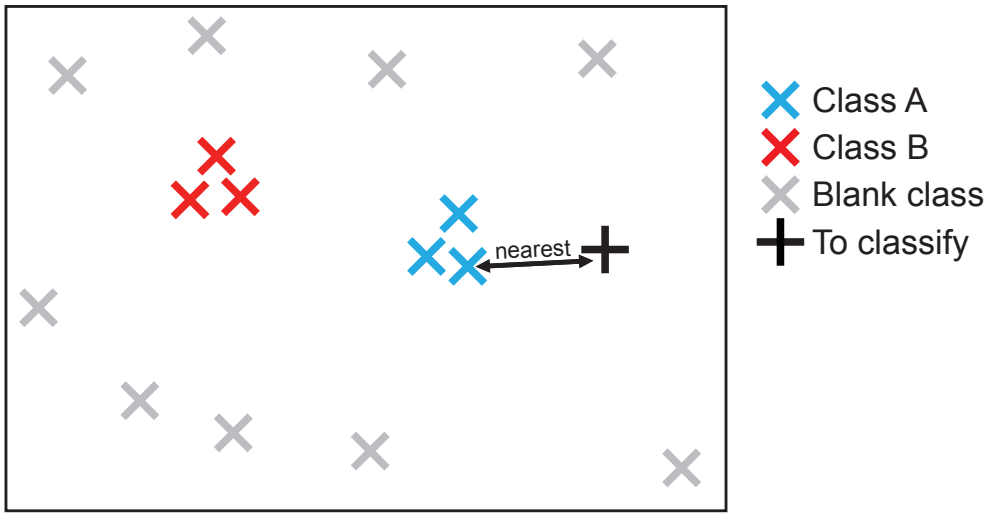


Figure 9.1: The problem with a learnt blank class: here 1-NN matches the unknown gesture with a normal class, even though it would be best labeled blank.

9.1.1.3 Reverse outlier detection with threshold recognition

The approach we have chosen is the second one. We reformulated the problem as a "reverse outlier detection" problem. In outlier detection, there is generally a single class which represents the normal case; the goal is to detect when a new input point is an outlier not corresponding to the normal case. For example, outlier detection might be applied on an airplane maintainance system, where the "OK state" refers to all sensors being in some region of the sensor value space, and the outlier condition would be triggered when one sensor goes off the normal zone.

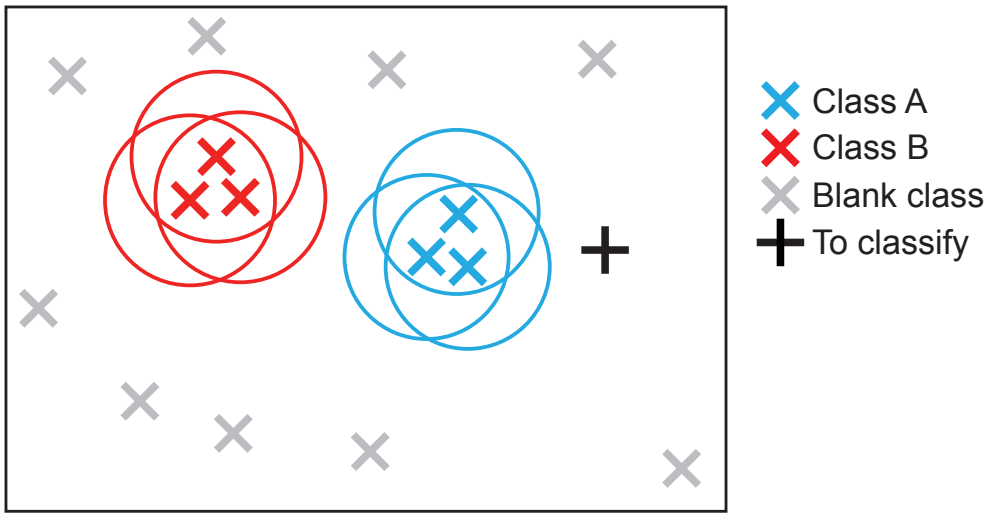


Figure 9.2: Introducing thresholds (circles around instances) prevents the blank gesture to match with a normal class even though it has never been seen before.

Our approach is to describe the normal case as "no gesture", and the outlier case as "some gesture must be detected". This is different from the traditional point of view in outlier detection, hence the name "reverse". It is, however, simply a semantical difference of interpreting what is normal and what is abnormal; this does not change the underlying algorithm.

It turns out 1-NN is very easily extendable to an outlier detection algorithm with the use of thresholds. One just needs to set a fixed threshold, and the class is triggered if and only if one instance is close enough (i.e. distance is under the threshold) to the new input data point. Conceptually, one might imagine spheres around each instance in the space; the continuous stream of data describes an endless curve in the input space, and a class is triggered only when this curve enters one of these spheres.

It extends very well from 1-NN thanks to the distance computation. Therefore this is the algorithm we will use to turn isolated, for-classification 1-NN into stream-enabled, for-outlier-detection threshold recognition. The component which performs threshold recognition in our pipeline is accordingly called a Threshold Recognizer.

More precisely the algorithm of the Threshold Recognizer is the following:

- at each new motion point, update the distance (see next section) between the stream

and each training time series.
- for each instance, if its updated distance is under the class threshold, trigger this class label (gesture detected);
- if no instance was under the threshold, return the blank label (no gesture detected).

Initially, we used to set a single threshold for all instances. However, experimentation showed us that different gesture classes would need different thresholds. For example, gesture "go" is dynamic and has quite a high variation between two instances. However, gesture "stop" is static and almost always performed the same way. Therefore we decided to set one threshold per class: as we shall see in the experiment at the end of this chapter, successful recognition results suggest choosing one threshold per class is a very good alternative to augment the model's expressive power while retaining a low complexity.

Additionally, a disambiguation rule decides which label takes precedence in case two labels match. In our case, we return the label of the first example to match without further disambiguation. Note that in any case, if two instances of different labels match, it probably means that either the classes are too close to one another (gesture confusion) and that, unless thresholds are decreased to avoid this confusion, it will lead to possibly bad results during operation. Two things are needed to avoid this undesirable situation: first, gesture classes must not confused. We took care of this in Chapter 4 and identified problematic gestures in Chapter 8. Second, thresholds should be set accordingly; thresholds too big can lead to gesture confusion, while too low might generate false negatives. Threshold selection is discussed along with the experiment at the end of this chapter, and tackled in precise detail in Chapter 10.

9.1.2 Extending DTW

Our stream-enabled reverse outlier detection technique needs distances to decide whether it triggers a class or not; hence it is necessary to extend the traditional isolated computation of DTW in a stream-enabled version. There are two possible techniques:

- start over with a new matrix: compute fixed DTW on a sliding window
- keep the same matrix: compute only a new column of accumulated costs for each new data point

9.1.2.1 Computing DTW on a sliding window

This technique is the simple succession of two steps. First, consider a sliding window on the data stream; it returns a fixed time series whose length is that of the sliding window. Then, compute traditional, fixed DTW with the two time series: one from the sliding window, the other from the training dataset. The result of DTW is the updated distance.

The main drawback of this approach is the high computation time needed to carry out the DTW computations. For each new data point, the sliding window leads to a new time series (the same as previous one, shifted by one point), which requires a complete full DTW computation again. Therefore the time complexity is quadratic since it is the traditional DTW algorithm on top of a sliding window.

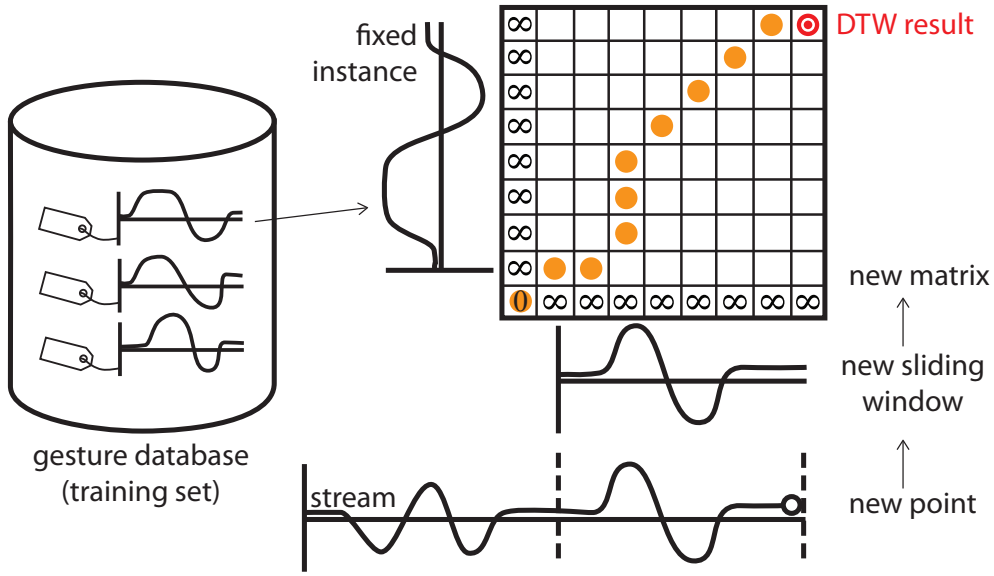


Figure 9.3: DTW over sliding window computes a new matrix for each new point (quadratic complexity).

9.1.2.2 Stream DTW: extending DTW column-wise

This technique (discussed by [190] in the context of SPRING) is an extension of the DTW equations, designed to work with one of both axes being infinite. Here, we do not compute a complete quadratic matrix each time; rather, we re-use the previous matrix by adding a new column at its end. Hence we prefer to say we work not with a "matrix" but with a "stripe" of accumulated costs, of which one axis is infinite. We will refer to this technique as Stream-DTW from now on.

At each new data point, one new column is added to the stripe using the DTW recursive equation. The updated distance is the top-right cell as in DTW, i.e. it is the top-cell of the newly computed column.

As an important modification, unlike the DTW base cost for each new column which is set to ∞ to force the matching of the whole time series, we set the Stream-DTW base column cost at 0. The rationale is that we do not want the matching to take place on the full stream, but rather on the end of the stream with the best available "root". Therefore the base cost of each new column should be set to 0 to allow "rooting" from any point in the stream. The one-time initialization of the left-wise column (fixed time series) should still be left to ∞ for the few instants at the very beginning of the stream.

Unlike the sliding window approach, this technique does not require a full DTW matrix computation, but a single column. Therefore the time complexity is not quadratic, but linear, which represents an important gain of time in terms of CPU usage. In terms of memory usage, it is only necessary to keep in memory the two last columns, the older of which can be reused when a new column must be reallocated.

Algorithm 2 Stream-DTW

```
1:  procedure STREAM-DTW(r_1, s)          ▷ (r_1 and s are 1-indexed; A is 0-indexed)
2:      A = new matrix [0..l_1, 0]        ▷ accumulated costs stripe
3:      A[., 0] = ∞                       ▷ init with 1 column
4:      A[0, 0] = 0
5:      for t = 1 to ... do
6:          Allocate a new column A[t]
7:          A[0, t] = 0                   ▷ Allow a root at each column
8:          for i_1 = 1 to l_1 do
9:              A[i_1, t] = d(r_1[i_1], s[t]) + min(A[i_1−1, t  ],
10:                                                 A[i_1  , t−1],
11:                                                 A[i_1−1, t−1])
12:         end for
13:         yield A[l_1, t]               ▷ acc. cost of min. warping path from root to (l_1, t)
14:     end for
15: end procedure
```

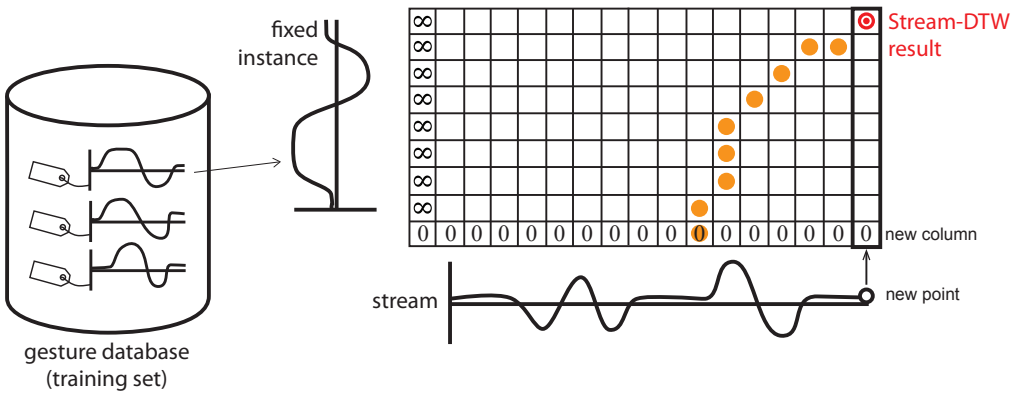


Figure 9.4: Stream-DTW: extending the DTW matrix with a new column makes it possible to obtain a linear complexity for each new point.

9.1.3 Extending DTW*

We have seen in Chapter 8 that length normalizing the gestures was mandatory to achieve accurate recognition in the case where different length time series are compared. For this purpose, we developed DTW*, which is simply DTW divided by the accumulated cost matrix diagonal. It is easy to do when the two time series have a fixed length, but it is not trivial in this case where one axis is infinite. In this subsection, we explain how to obtain all lengths information to apply DTW* normalization.

We need to have the length of both time series being matched, l_1 and l_2, so as to divide by the length of the diagonal:

$$\sqrt{{l_1}^2 + {l_2}^2} \tag{9.1}$$

In Stream-DTW, one of the time series is fixed (the one coming from the training dataset), so its length l_1 is directly known. However, there is no definite length m for the time series on the stream axis. Obviously, we should not take the length to be the whole stream length, since it is always incrasing and will just make the denominator tend to infinity and the resulting normalized distance tend to zero over time. Rather, we should find out the significant part of the stream which has led to time series matching, since we allow rooting from any point in the stream axis (rooting is discussed above).

Algorithm 3 Stream-DTW*

1: **procedure** STREAM-DTW*(r_1, s)
2: $A =$ new matrix $[0..l_1, 0]$ ▷ $W =$ new matrix $[0..l_1, 0]$
3: $A[., 0] = \infty$
4: $A[0, 0] = 0$ ▷ $W[0, 0] = 0$
5: **for** $t = 1$ to $\ldots$ **do**
6: Allocate a new column $A[t]$ ▷ Allocate a new column $W[t]$
7: $A[0, t] = 0$ ▷ $W[0, t] = 1$
8: **for** $i_1 = 1$ to l_1 **do**
9: $A[i_1, t] = d(r_1[i_1], s[t]) + \min(A[i_1-1, t\ \ ],$ ▷ $W[i_1, t] = W[i_1 - 1, t\ \ ]$
10: $A[i_1\ \ , t-1],$ ▷ $W[i_1, t] = W[i_1\ \ , t-1] + 1$
11: $A[i_1-1, t-1])$ ▷ $W[i_1, t] = W[i_1 - 1, t-1] + 1$
12: **end for**
13: **yield** $A[l_1, t] \,/\, \sqrt{l_1^2 + l_2^2}$ ▷ with $l_2 = W[l_1, t]$
14: **end for**
15: **end procedure**

Obtaining l_2, the horizontal length of the matching subsequence in the stream, is achieved by tracking the width of the warping path. A root cell has a zero accumulated cost and no parent. To do this, we introduce a second matrix or stripe W, known as the "width" matrix, for which a cell $W[i_1, t]$ stores the width of the warping path leading to $A[i_1, t]$" in the accumulated cost matrix. In terms of recursive calculation, it is incremented if and only if the parent cell (choice of min operator on DTW recursive equation) was on the previous column. Remember that in the recursive equation, there are three choices: the *left* cell, the *lower-left* cell, and the *lower* cell. The width is thus incremented for the *left* and *lower-left* choices, but it is untouched when the choice is the *lower* cell. See Figure 9.5.

Last, when reading the final cell $A[l_1, t]$ at the top of the newly-computed column, we have also access to the width of the backtracking path: l_2. With this information it is now possible to apply DTW* normalization, that is, dividing the final cell's accumulated cost by $\sqrt{{l_1}^2 + {l_2}^2}$.

9.2 Evaluating accuracy of our system in a streaming context

9.2.1 The need for a streaming-specific evaluation metric

When the system is trained, we operate in testing phase. In this setup, the ground truth labels are known, but are not handed to the recognition system, since we wish to evaluate

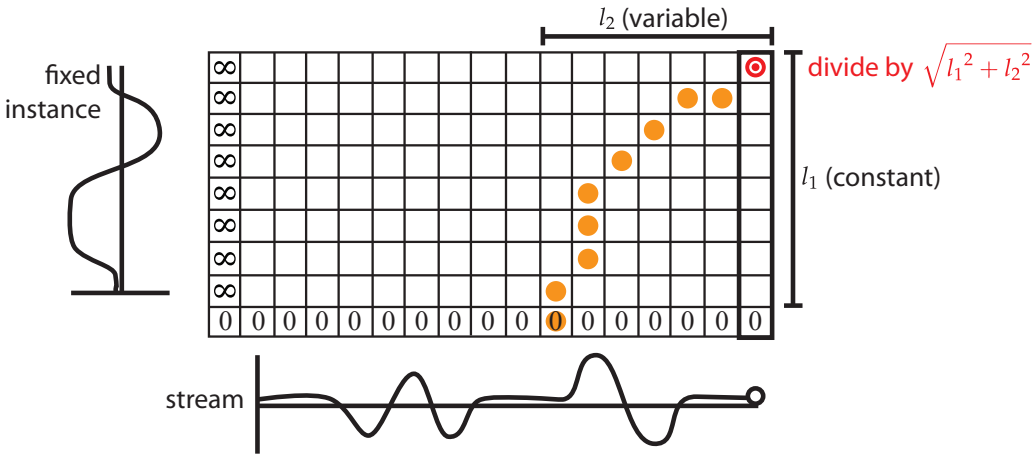


Figure 9.5: Stream-DTW*: tracking the width l_2 is necessary to obtain the length-agnostic version.

its performance. In the same time, the output of the recognition system is compared to the ground truth labels in order to determine if the recognition was correct.

Offline classification systems are designed to label separate inputs which are not necessarily related one to another. Unlike a classification system, our pipeline is designed to handle the detection as well as the classification, that is, not only label a gesture correctly but also detecting it at the right time.

Evaluation metrics for streaming systems depend on the nature of the machine learning task itself. In our case, it helps to understand how our gesture recognition system behaves. In general, it follows the pattern below:

1. true gesture is started;
2. distances of each instance (with the correct label) start to decrease;
3. during the gesture performance, at least one instance's distance becomes small enough to pass under the threshold. As a result, the recognized label is emitted;
4. true gesture ends;
5. distances increase anew, ending up all above the class threshold. As a result, the recognized label is no longer emitted.

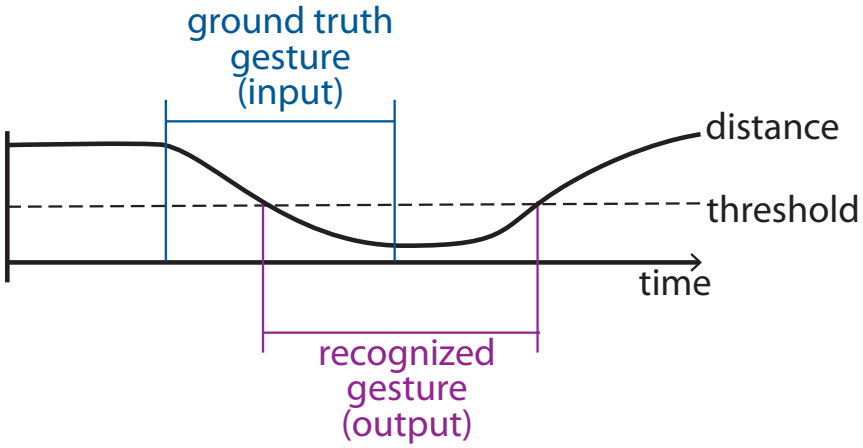


Figure 9.6: Correct detection.

In this case, we would probably like to give the recognition system a good score and not penalize it too much. Indeed, it accomplished its task: recognizing the gesture. It does not have to recognize the gesture before it begins; it is correct to emit the label during the gesture.

However, consider the case where the gesture is recognized slightly later due to a smaller threshold (Figure 9.7). The answer is still correct, apart from latency, but there is no difference between both; the gesture was still recognized. We do not aim to penalize the recognition.

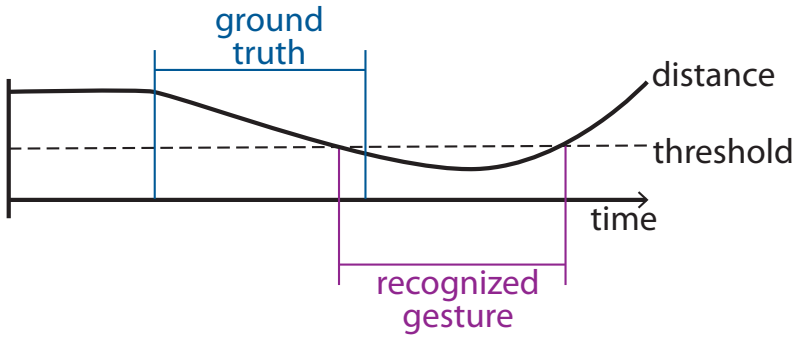


Figure 9.7: Correct detection but gesture is recognized a bit later. It should not be penalized compared to Fig. 9.6.

Sometimes, a ground truth may be annotated a bit late, which can make the recognition trigger before the expected annotation. We consider this behaviour to be fine, since the system correctly recognized the gesture when it was done, as long as the recognized gesture does not end until the ground truth starts.

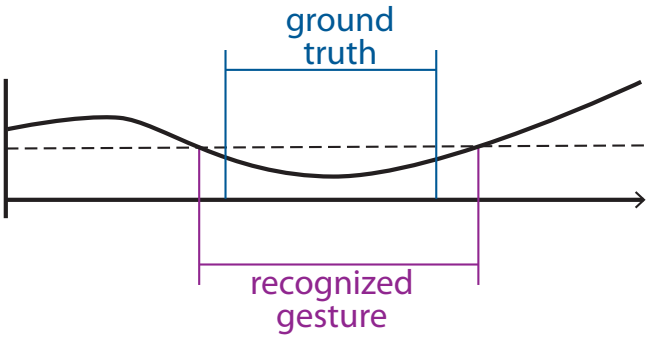


Figure 9.8: Ground truth gesture within recognized gesture.

There may be also cases where the gesture is just recognized within the ground truth gesture bounds.

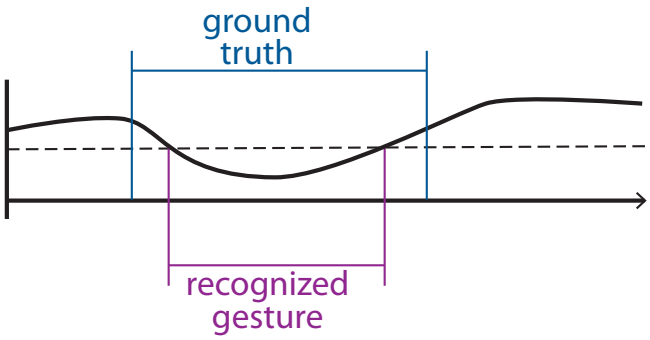


Figure 9.9: Recognized gesture within ground truth gesture.

Of course, there are cases where the gesture recognition is wrong:

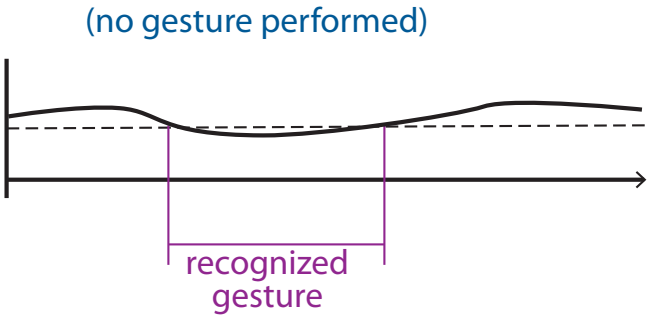


Figure 9.10: Wrong/FP.

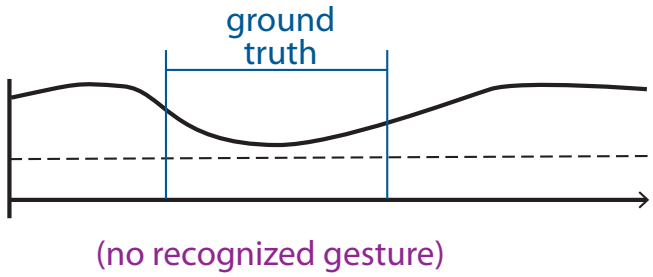


Figure 9.11: Miss/FN.

9.2.2 Recall and precision in streaming

In order to tailor to the specific behaviour of this system, we have devised our own counting system to keep track of its performance. We were inspired with traditional classification metrics, which defines the four core counts TP (true positive), TN (true negative), FP (false positive), FN (false negative). For each of them, are they able to work in streaming?

- TP: yes. It represents a gesture expected and well recognized (match)
- FN: yes. It represents a gesture expected, but not recognized (miss)
- FP: yes. It represents a gesture not expected, but recognized (wrong)
- TN: no. It makes little sense; it would represent times when no gesture is expected and no gesture is emitted. Although it does happen indeed, unfortunately it is difficult to *count* these.

Therefore only three of these four available counts are available. Note that a gesture detected at the right place but with the wrong label would count as 1 FP + 1 FN.

Typical binary classification metrics are the following:

$$\begin{aligned} \text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{fallout} &= \frac{\text{FP}}{\text{FP} + \text{TN}} \\ \text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{false omission rate} &= \frac{\text{FN}}{\text{FN} + \text{TN}} \end{aligned} \tag{9.2}$$

These metrics are ratios, varying from 0% to 100%, where higher values reflects better performance. A value equal to 100% is perfect in regards to the measured quantity.

There exist four additional metrics which are just transformations of those four: namely, miss rate, specificity, false discovery rate, and negated predictive value; they do not offer

substantial improvement except another way to read the same values ("1% miss rate" instead of "99% recall", for example).

Among these metrics, we can keep only those whose equation depends on TP, FN, FP, all of which are well-defined. On the other hand, TN being ill-defined, we should avoid metrics whose equation involves TN since we cannot compute it reliably.

Analysing the previous four metrics, only two of them can be retained; recall and precision. Those metrics are complementary: recall stays high if not not too many gestures were missed, while for precision it is the case if not too many wrong detections were triggered.

Furthermore, precision and recall can be used to obtain a specific metric known as F_1:

$$F_1 = 2\,\frac{\text{precision.recall}}{\text{precision} + \text{recall}} \tag{9.3}$$

which stays high as long as both precision and recall are high, and drops whenever one of both decreases. It can be generalized to a metric

$$F_\beta = (1 + beta^2)\frac{\text{precision}\,.\,\text{recall}}{(\beta^2\,.\,\text{precision}) + \text{recall}} \tag{9.4}$$

in order to put more weight on either the precision or the recall.

9.2.3 Definition of markers

The only task remaining is to define precisely the TP, FN, and FP counts, in regards to the ground truth and recognized time markers. In order to accomodate all cases presented in previous figures, we propose te following definitions.

A gesture is the combination of a label and two time markers: $G = (\ell, t_{\text{beg}}, t_{\text{end}})$.

Consider a ground truth gesture $(\ell_{\text{truth}}, t^{\text{truth}}_{\text{beg}}, t^{\text{truth}}_{\text{end}})$, and similarly, a recognized gesture $(\ell_{\text{reco}}, t^{\text{reco}}_{\text{beg}}, t^{\text{reco}}_{\text{end}})$.

Let us define a *match* as the conjunction of two events:

- same labels: $\ell_{\text{truth}} = \ell_{\text{reco}}$
- time spans overlap: $(t^{\text{truth}}_{\text{beg}} \leq t^{\text{reco}}_{\text{beg}} \leq t^{\text{truth}}_{\text{end}})$ or $(t^{\text{reco}}_{\text{beg}} \leq t^{\text{truth}}_{\text{beg}} \leq t^{\text{reco}}_{\text{end}})$

This temporal equation includes all wanted cases:

- truth starts, reco starts, truth ends, reco ends (overlapping succession truth ⇒ reco)
- truth starts, reco starts, reco ends, truth ends (reco fully included in truth)
- reco starts, truth starts, reco ends, truth ends (overlapping succession truth ⇒ reco)
- reco starts, truth starts, truth ends, reco ends (truth fully included in reco)

And rejects cases where there is no overlap:

- truth starts, truth ends, reco starts, reco ends
- reco starts, reco ends, truth starts, truth ends

When there is a match, we mark both the truth gesture and the recognized gesture as OK. After analyzing all combinations, the remaining gestures that are not OK are either a FN (if it was a ground truth gesture) or a FP (if it was a recognized gesture).

An already-OK gesture can be used to make another gesture match; in particular this allows a recognized gesture to match with several ground truth gestures as long as each (truth, reco) pair's temporal bounds overlap.

In fact, counting FNs, FPs and TPs does not have to be done as a post-hoc analysis, it can also be carried out in streaming by analyzing the succession of time markers related to ground truth and recognized gestures:

- FN: incremented when a ground truth gesture ends without having seen a recognized gesture (same label) during its timespan.
- FP: incremented when a recognized gesture ends without having seen with a ground truth gesture (same label) during its timespan.
- TP: incremented when a ground truth sees a recognition gesture (same label) before ending.

9.2.4 Delays, merging gestures

With experience, we noticed this technique occasionally led to an FN being closely followed by an FP of the same label. It indicates a gesture was well detected, but a bit too late. Also, the opposite case when a gesture is detected a bit too soon can happen. In general it is because the annotation time markers are a bit too strict and do not leave enough time for the detection to trigger.

In order to comply with both cases and give some flexibility to the recognition metric, we introduced two parameters that we call "delays", whose function is to increase the time bounds: the *ground truth delay*, and the *recognition delay*. Both are simple time values which indicates how long the "end" marker of each gesture should be retained.

A small difference between th ground truth delay and the recognition delay is that recognized gestures are allowed to merge, whereas this is not allowed for ground truth gestures. Indeed, if we allowed merging two consecutive gestures, we would change the number of labels to be detected (the sum TP+FN, denominator of the recall).

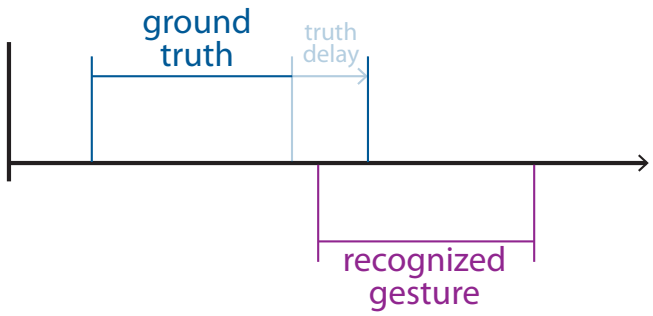


Figure 9.12: Sustaining the ground truth allows a correct matching in case the ground truth bounds are too tight.

It does not cause problems to merge recognized gestures, since it does not change either the denominator of the recall TP+FN or of the precision TP+FP.

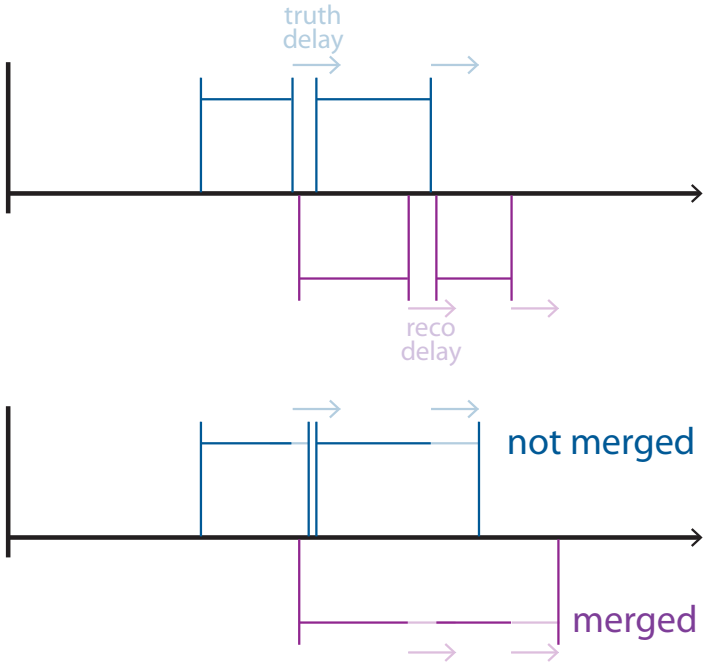


Figure 9.13: Recognition delays are allowed to be merged in a single gesture. However, ground truth gestures are not allowed to merge, since that would change the sum TP+FN and create instability in the evaluation metrics.

Furthermore, not from an evaluation perspective, but from a robot control one, it is beneficial to merge the recognized bounds. This will introduce a hysteresis in the output system. The improvement takes place when the distance is very close to the threshold, leading to an oscillation below and above it; with recognition delays, it allows the system to stay on for some more time instead of oscillating on/off rapidly, what we call a “blinking effect”. In our experience, setting both delays to 0.5 seconds leads to satisfactory results.

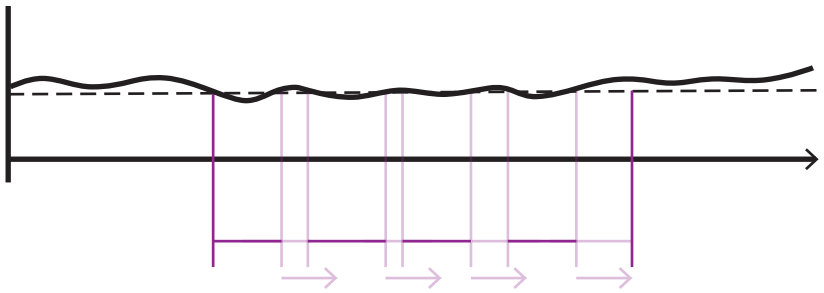


Figure 9.14: Recognition delays create some hysteresis to avoid a "blinking effect" in case the distance oscillates around the threshold.

9.3 Experiment of gesture recognition in streaming

9.3.1 Experiment: one user, four gesture streams, static.

In this chapter, we present a first experiment which summarizes some progress we did with stream-enabled distances and gesture streams. Later experiments will be conducted to gather more data and test subtle problems which will be uncovered by this experiment in particular the absence of an automatic threshold discovery and bad examples pruning, an issue we tackle in Chapter 10.

We captured four gesture streams with several examples of each gestures from the following dictionary:

- go (3x): starting from an open hand upwards, palm facing the cheek, swing the arm forwards. (Swing repeated 3 times in a row.)
- stop: closed hand upwards
- steer: "L-shape" with thumb and index; other 3 fingers closed. Back of hand facing left. Thumb should point up, index forward. No motion.
- left: "L-shape" with thumb and index; other 3 fingers closed. Back of hand facing down. Thumb should point left, index forward. No motion.
- right: "L-shape" with thumb and index; other 3 fingers closed. Back of hand facing up. Thumb should point right, index forward. No motion.
- slower (3x): open hand, palm facing down, back facing up; swing moderately up and down (sagittal plane). (Swing repeated 3 times in a row.)
- faster (3x): open hand, palm facing up, back facing down, swing moderately up and down (sagittal plane). (Swing repeated 3 times in a row.)

Each gesture was repeated multiple times; precise details are given in Table 9.1, but each capture file contains more than 50 gestures in total; all four files totalize 228 gestures. Capture files have respective lengths of 10m50, 2m30, 4m, 3m41, the sum of which makes exactly 21 minutes and 10 seconds of stream data to analyze. Capture were made with the V-Hand data glove we purchased for evaluation purposes of our algorithms; it operates at 100 Hz.

Ground truth data were given by the user via clicking on a button while performing the gesture. Pressing the button would indicate the beginning of a gesture (t_{beg} marker), while releasing it would indicate the end (t_{end}). All captures were made by a single operator. Since this labeling system was somewhat brittle (it was replaced in a later version of our stream capture GUI, see Chapter 6), no mobility could be achieved. Therefore all captures were realized in the "standing" condition.

This experiment was performed at a time when the advanced training procedure given in Chapter 10 was not developed yet. Therefore the learning rule at this time was the following: time series are extracted from the training stream and are included in the gesture database given to the threshold recognizer without further analysis. This is a very simple yet effective way to train the system.

Furthermore, the procedure of Chapter 10 also explains how to set the thresholds automatically. Hence, when this experiment was done, we relied on offline manual tuning of the thresholds. In effect, we mostly toyed with the different thresholds and observed how

Pass	Label	FP	FN	TP	Precision / Recall
Pass 1	stop	0	0	18	100% / 100%
	left	0	0	17	100% / 100%
	steer	6	3	17	73.9% / 85%
	slower	3	0	33	91.6% / 100%
	right	0	0	20	100% / 100%
	go	0	0	33	100% / 100%
	faster	0	1	31	100% / 96.8%
	(all)	9	4	169	94.9% / 97.6%
Pass 2	stop	0	0	17	100% / 100%
	left	0	0	17	100% / 100%
	steer	11	4	14	56% / 77.7%
	slower	2	1	29	93.5% / 96.6%
	right	0	0	18	100% / 100%
	go	1	1	33	97% / 97%
	faster	0	0	33	100% / 100%
	(all)	14	6	161	92% / 96.4%
Pass 3	stop	0	0	17	100% / 100%
	left	0	0	15	100% / 100%
	steer	0	9	6	100% / 40%
	slower	1	2	28	96.5% / 93.3%
	right	0	0	17	100% / 100%
	go	0	2	36	100% / 94.7%
	faster	0	2	29	100% / 93.5%
	(all)	1	15	148	99.3% / 90.7%
Pass 4	stop	0	0	20	100% / 100%
	left	0	0	17	100% / 100%
	steer	0	0	19	100% / 100%
	slower	0	0	30	100% / 100%
	right	1	2	18	94.7% / 90%
	go	0	0	42	100% / 100%
	faster	2	0	33	94.2% / 100%
	(all)	3	2	179	98.3% / 98.8%
All passes	stop	0	0	72	100% / 100%
	left	0	0	66	100% / 100%
	steer	17	16	56	76.7% / 77.7%
	slower	6	3	120	95.2% / 97.5%
	right	1	2	73	98.6% / 97.3%
	go	1	3	144	99.3% / 97.9%
	faster	2	3	126	98.4% / 97.6%
	(all)	27	27	657	**96.0% / 96.0%**

Table 9.1: Streaming recognition results with Stream-DTW* and manual thresholds.

accuracy results changed; this fragile procedure is exactly why we designed that advanced procedure.

Here, we used thresholds on Stream-DTW* with the following values: { go: .25, steer: .10, right: .10, slower: .16, left: .25, (others): .20 }. As we shall see in Table 9.1, these values lead to balanced results with regards to FPs and FNs, reflected by similar Precision/Recall scores.

Because evaluating such a low number of streams might lead to much statistical noise, we proceeded to a four-fold cross-validation procedure on the four-streams. More precisely: for each pass, one file is the training stream and the three remaining files are the test streams. While it might not be enough to obtain a statistically satisfying result, it is nonetheless better than a single test/run and it will allow us to draw conclusions.

During training, motion-processed time series in $\mathbb{R}^{13}$ are simply stored along with their labels in a gesture database. This database is then given to build a Threshold Recognizer, that is, a component analyzing streaming distances for each instance in the database and triggering a detected label whenever a distance goes under some threshold. These values are the "hand-crafted" thresholds, found by trial and error, discussed above. This Threshold Recognizer is used for the test phase of all 3 remaining streams.

As Table 9.1 suggests, the technique we developed seems to work quite well in streaming. Regarding chosen gestures for this experiment, mostly positive conclusions can be drawn since all of them but one are recognized with very good accuracies (precision and recall above 95%).

The gestures which seems to cause trouble is "steer" ("L-shaped" hand with thumb pointing up), intended to be a neutral position between "left" and "right". We are not sure why this gesture causes problems. It appears that Pass 4 runs better with respect to this gesture, so a possibility is that the fourth capture file is not as well labeled as the others. Such mislabeling is unfortunately a risk with our specific application of gesture recognition. Anyway, it did not have a real specific meaning, and was rather an experiment to check whether it would be interesting to include a "pre-gesture" before turning left or right; but this experiment suggests to use "left" and "right" gesture alone, since they both have excellent recognition scores (4-fold Precision/Recall: left 100%/100%, right 99.3%/97.9%).

One might want to note that removing the gesture "steer" from this data set makes Precision/Recall jump from 96% / 96% to 98.5% / 98.4%.

As we hinted above, a problem with our current approach is its reliance upon an user-submitted (possibly found via off-line trial-and-error, alas) hand-crafted set of threshold values. Here the values have been set by us in order to find a good equilibrium between misses (FN) and wrongs (FP) which explains the balanced results obtained. Ultimately, this highlights the need for an automated technique to select threshold values. We will tackle this problem in Chapter 10.

In summary, this experiment suggests the following conclusions:

- the recognition pipeline works well in the current context;
- all proposed gestures are well detected, except "steer" which may be removed;
- there is a need for an automatic labeling system;
- there is a need for an automated technique to select thresholds, possibly removing low-quality instances along the way.

Chapter 10

Advanced training procedure

Contents

In the previous chapter dedicated to stream recognition, we have demonstrated that associating a Threshold Recognizer and Stream-DTW* led to succesful recognition. We have also discussed how the "simple extraction" learning technique can be enhanced by not penalizing the user for occasional low-quality gestures; in this chapter we describe that "advanced training procedure". It is used to analyze user gestures for possibly removing the low-quality instances from the training set. It also automatically sets class thresholds so that the user does not have to tweak them: we want the training to be as user-friendly as possible and to avoid asking numerical values for each class on the GUI.

This chapter is largely inspired of a paper we published at the International Conference of Pattern Recognition 2016 [55]. We tried to make our work on time series quality assessment as general as possible so that it can benefit not only researchers interested in gesture recognition, but the time series community as a whole. Of course, we use this general procedure in our gesture recognition pipeline.

In this general context, we can reformulate the key outcomes from previous chapters as follows. On-line supervised spotting and classification of subsequences can be performed

by comparing some distance between the stream and previously learnt time series. However, learning a few incorrect time series can trigger disproportionately many FNs. Below, we propose a fast technique to prune bad instances away and automatically select appropriate distance thresholds. Our main contribution is to turn the ill-defined spotting problem into a collection of single well-defined binary classification problems. This is achieved by segmenting the stream and by ranking subsets of instances on those segments very quickly. Naturally, we demonstrate this technique's effectiveness on our gesture recognition application.

10.1 Related work

A lot of work has been tackled during the last decade in the area of instance selection. In the big data paradigm, instance selection has been addressed through the angle of data reduction to maintain algorithmic scalability, but also to reduce the noise in the training data [160] [44] [65]. As the instance selection problem is known to be NP-hard [79], heuristic approaches have been developed, among them evolutionary algorithms have been largely experimented [46]. It is quite noticeable that very few of those works, to our knowledge, have specifically addressed the problem of instance selection in a data stream recognition scheme. Because the processing of streaming data is particularly resource demanding, data reduction and in particular instance selection, is a very pressing question. [24] have addressed this issue as an active learning problem in a streaming setting, while mainly considering the detection of mislabeled instances. We also address the noise reduction angle, although the reduction of the redundancy in the training set is indeed an issue and can be viewed as an extension of our work. Nevertheless, the noise we tackle is much more located at the feature level rather than at the class variable level. It has also been recently raised that a nearest neighbor classifier does not necessarily yield optimal results by using the complete training set [224].

10.2 The stream recognition problem

As we have seen in Chapter 9, the problem we seek to solve can be formulated as follows. Let (s_t) be a stream of points in $\mathbb{R}^N$, that can be infinite in time. In our case, it represents the stream of motion values coming from our data glove. Our goal is twofold:

1. spotting: localize meaningful subsequences in this stream, and
2. classifying: label those subsequences with a discrete class label.

In our motion data example, those subsequences are the gestures that we need to both spot and classify. As before, we refer to this dual task "spot + classify" as "recognition".

In Chapter 6 we have described how this recognition is done in a supervised manner: before testing, the recognizer is given access to a training stream in which subsequences are all marked with two temporal bounds (for spotting) and a label (for classifying), i.e. a tuple $(t_{\text{beg}}, t_{\text{end}}, \ell)$. Most of the stream points will not be meaningful, in the sense that there is no detection expected. In our gesture example, resting, scratching one's arm or moving

naturally because of walking are three kinds of data that could be considered meaningless because they are not considered explicitly as gestures (assuming they were not explicitly labelled so in the training set). In a way, those meaningless data points can be interpreted as some kind of "silence" (or "noise").

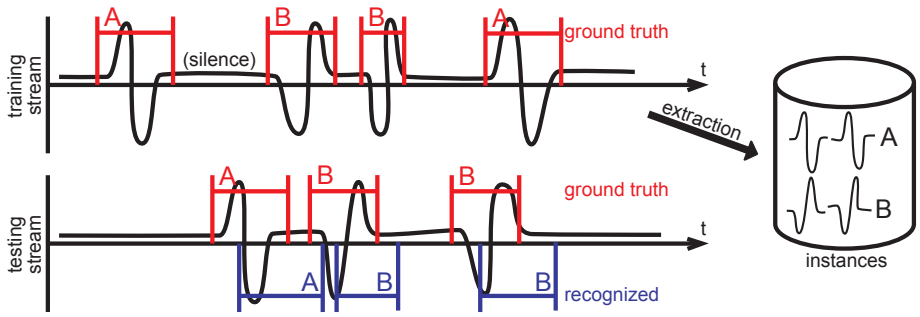


Figure 10.1: Training and testing a threshold recognizer. All spotted subsequences in blue are correct because they each overlap with a ground truth subsequence with the same label.

Of course, since we design a real-time system that works on line, at testing time our recognizer must be able to read the incoming data stream and label it continuously; it is not allowed to read the whole stream first and output all subsequences afterwards.

The task of the recognizer is, given input points of the unlabelled stream, to emit a BEGIN bound when it detects a subsequence (along with the classified label) and a END bound when it believes the subsequence is over. Of course, it is difficult to detect a subsequence before it has even started. Hence we do not expect the recognizer to emit the exact ground truth boundaries of the spotted subsequences. We simply say that recognition is correct iff recognized and ground truth subsequences overlap and have the same label, as illustrated in Figure 10.1. This is the multiclass recognition problem; unfortunately, as we have discussed in Chapter 9, notions such as TN (True Negative) are not well defined because there is no proper delimitation of the objects to be classified.

10.3 Threshold recognizer

In this section we will summarize the important elements of the Threshold Recognizer component we described in Chapter 9.

The Threshold Recognizer answers to the recognition problem by emitting BEGIN and END bounds thanks to a dissimilarity function. Chapter 9 has shown that Stream-DTW* worked well for our purposes. The Threshold Recognizer thus outputs the time markers corresponding to a gesture class when the distance goes under the threshold.

The Threshold Recognizer relies on three internal components. First, a dissimilarity measure (informally referred to as "distance") between one time series and the stream (for example, Stream-DTW*). Second, a training set of labelled subsequences (the "instances"). Third, one threshold per class. In Chapter 6 we also called the association of the two latter (instances and thresholds) a "user profile".

The Threshold Recognizer behaves as follows:

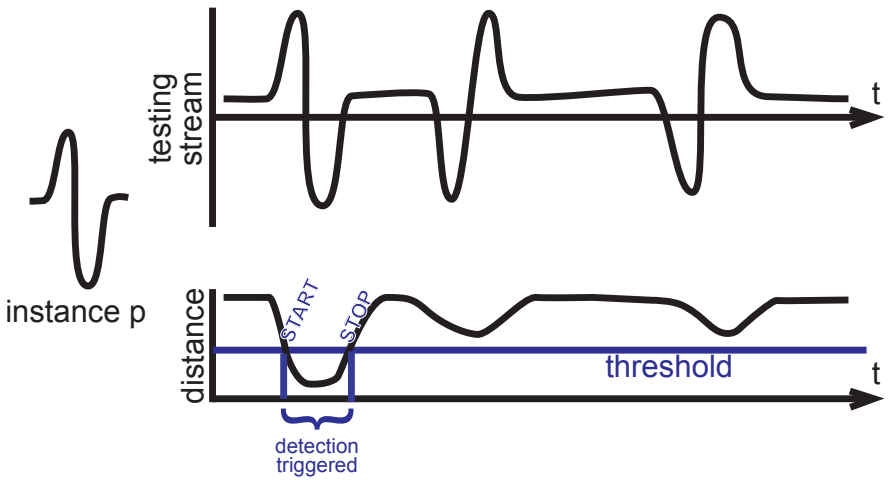


Figure 10.2: A threshold recognizer triggers detection when distance is under the threshold. While a single instance analysis is shown here, in practice all instances are considered.

- for each incoming stream point, compute/update distances between each instance and the stream
- emit BEGIN when at least an instance goes under its label's threshold and emit END when no more instance is below threshold
- (advanced: in the case where more than one label is detected, a disambiguation rule such as "pick the label having the closest instance to the stream")

10.4 Handling an imperfect training set

The major drawback of such a threshold recognizer is the following: it takes only one instance to trigger a BEGIN bound. What if one instance in the dataset is particularly "bad"? It might have drastically bad consequences for the whole recognition. Imagine if the user were to record an instance of "left", for example, but accidentally did not perform the gesture at the correct time: now the dataset would be polluted by an instance labelled "left" whose time series describes the hand resting instead. That would imply that label "left" would be triggered *all* times where the user rests the hand afterwards! This is an unacceptable price to pay for a single misrecorded instance.

In the following, we describe a procedure to fix this issue, so that a few incorrect instances in the training set are pruned to avoid harming recognition rates. We wish to jointly discover the user profile, that is:

1. subset: which instances should be kept in the dataset (or alternatively: which instances should be removed)
2. thresholds: which thresholds should be given, for each class.

These two problems of selecting a subset and setting thresholds might seem to be independent at a first glance, but they are actually highly correlated. Indeed, it might be better

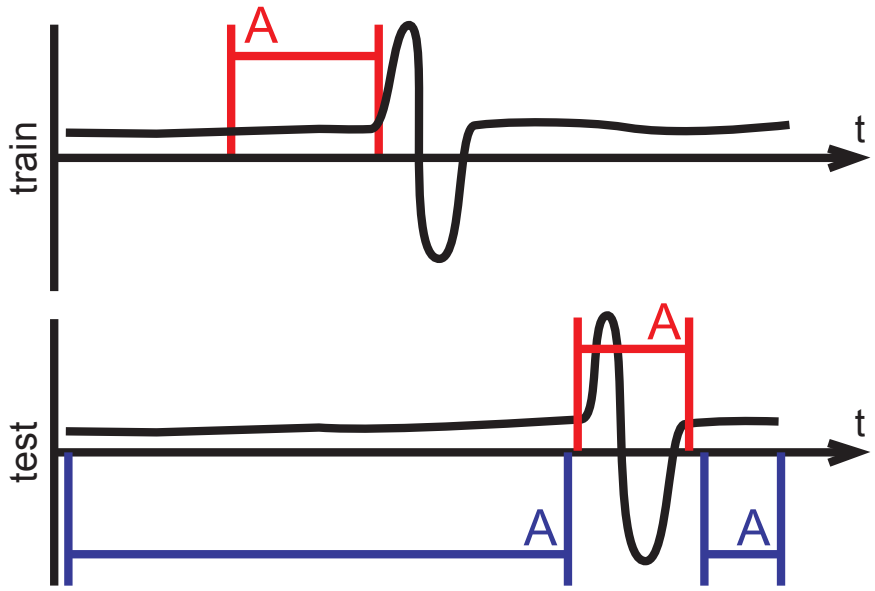


Figure 10.3: Misrecording a single instance can lead to dramatic consequences during recognition.

to have a low threshold if low-quality instances are going to match too often; or have a high threshold if we know examples will match only when they are expected to. Hence, subsets and thresholds must be analyzed jointly when seeking for optimal recognition rates.

In order to do this, a naive, brute force strategy would be to prepare all different subsets and different threshold values, then run the whole recognition procedure for each (subset, thresholds) pair. However, the full recognition procedure: computing all streaming distances, comparing distances and thresholds at each point... is too slow to be run many times. Furthermore, the space or pairs (subsets, thresholds) is enormous; combinations of subsets grow exponentially in the number of instances; and thresholds take continuous values, which already makes exploring their spaces disjointly untractable (proved NP-hard [79]), let alone jointly. Rather than analyzing the parameter space naively, we hereby describe a strategy that is much faster and allows for many runs of subset selection in a short period of time.

More importantly, it is key to observe that our algorithm is not only made for subset + threshold selection, but even more: it actually provides the basis for an extremely fast stream analysis in general, able to evaluate the performance of a recognizer on a validation stream and provide in turn accuracy metrics for multiple classes. It does so by turning the ill-defined multiclass stream analysis detection problem, into a collection of binary classification problems, for which notions such as FP, FN, TP, and more importantly, TN, are unambiguous. This enables the use of well established metrics and tools that have been known for decades, such as the ROC curve.

10.5 Overview of the procedure

Before diving into the details of the procedure, we will outline a broad overview of the steps involved.

- INPUT:
  - Two streams (training and validation) with subsequence boundaries and labels
- OUTPUT:
  - "high-quality" subset of extracted instances
  - threshold values
- SIDE OUTPUT (optional but possibly useful):
  - ranking (ordering) of instances from worst to best
  - numerical measure indicating how a given subset performs for one class, independently of any threshold selection
- STEPS:
  1. Extract instances from the training stream.
  2. Cut the validation stream into *segments*, and define *expectations* for each segment.
  3. Run the streaming distance on each instance.
  4. Find out *events* between instances and segments, i.e. when and where they can possibly trigger a recognition. Group events per label and per segment and sort them.
  5. Find out the worst candidates; prepare some promising subsets, from which the worst instances are removed.
  6. Analyze each promising subset on the segments to get a score out of them (very fast). Select the best subset. Select the optimal threshold on this subset.

10.6 In-depth description

Consider our input data is a labelled stream, that is, a sequence of points $s_t \in \mathbb{R}^d$, where d is the dimension of the stream, and $t = 0, 1, ...T - 1$ is the time. Usually this stream will represent sampled data acquired by sensors, monitoring metrics, etc. at a regular sample rate.

The subsequences are given as a collection of tuples $(t_{\text{beg}}, t_{\text{end}}, \ell)$, indicating the beginning and end boundaries of the subsequence along with its label. We assume subsequences do not overlap.

10.6.1 Extraction of training instances

For all subsequences $(t_{\text{beg}}, t_{\text{end}}, \ell)$ in the training stream, extract the timeseries between t_{beg} and t_{end}, and store them (see Figure 10.1).

10.6.2 Segmenting the validation stream

The validation stream (v_t) is also labelled with a collection of tuples $(t_{\text{beg}}, t_{\text{end}}, \ell)$ representing subsequences. This labelling represents the ground truth where the trained recognizer is expected to spot and classify subsequences.

In order to understand why we propose to cut the stream into segments, let us start by making an essential remark:

An instance will trigger recognition on a segment iff the *minimum* of its distances on the segment is under the threshold.

It means that instead of analyzing point after point, we can cut the stream into segments, store the *minimum* for each instance, and then check whether the minimum is under the threshold for the segment! This eliminates an enormous amount of work (Figure 10.5).

Each segment comes with one "expectation" and a set of tolerated labels (or simply "tolerances"). An expectation is a label ℓ, possibly $\varnothing$ (the blank label). A non-blank class ($\ell \neq \varnothing$) should be triggered by at least one instance of class ℓ under the threshold. On the other hand, if $\ell = \varnothing$, no class should be detected, hence all instances should be above their threshold. (Of course, $\varnothing$ is just a notation for the blank class and should not be the label of any actual instance.)

Tolerances serve to locally relax recognition constraints. If a segment tolerates ℓ, then it doesn't matter whether ℓ is triggered on this segment: it will never count as a mistake.

In order to create the segments, we will simply make *cuts*. Segments are these portions of time series between cuts:

1. During "silence" (no labelled subsequences), make a cut every T_{segment}. Set *expectation* to $\varnothing$ i.e. no label expected.
2. During meaningful subsequences with $(t_{\text{beg}}, t_{\text{end}}, \ell)$, make cuts at t_{beg} and t_{end} (do not sub-cut inside even if longer than T_{segment}). Set *expectation* to label ℓ.

Furthermore, for each segment that was created due to a subsequence with label $\ell \neq \varnothing$ (rule 2.), we "spread" ℓ onto neighbouring segments tolerance:

3. If segment s expects $\ell \neq \varnothing$, add ℓ to *tolerances* for segments $(s-1)$, $(s+1)$, and $(s+2)$.

The motivation for spreading tolerances 1 segment on the left and 2 segments on the right is that the streaming distance is usually sharp to decrease, but takes a bit longer to increase again after recognition and to leave the triggering zone below the threshold. In our experience 1 and 2 have been a good choice, and we believe they should be acceptable for most applications; it is up to the implementer to decide whether to tune those values for their particular task. Also, they naturally depend on the choice of T_{segment}, the number of points contained in each segment.

10.6.3 Minima computation

Distance computation is the most computing power demanding part of the algorithm. Therefore, we do it as early as possible, only once, and collapse dense distance information into economical representations based on segments and minima.

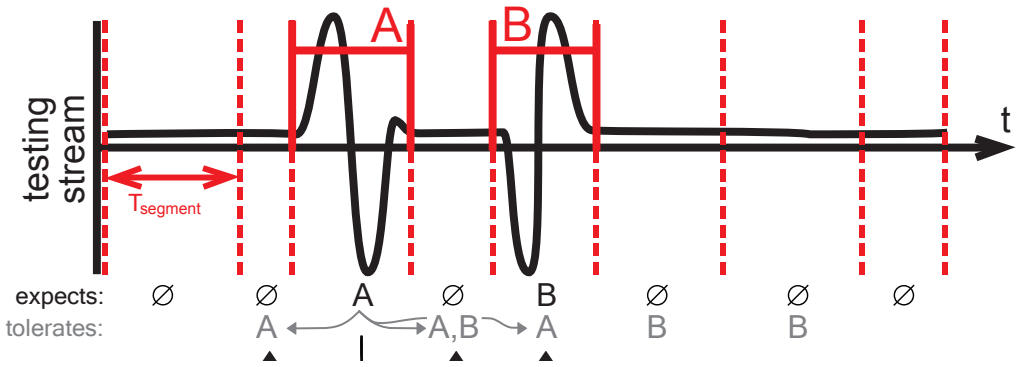


Figure 10.4: Segments delimit labelled subsequences and pieces of unlabelled points ("silence" or "noise"). Each segment *expects* a label (or no label) and can optionally *tolerate* some neighboring labels.

Let p be an instance of the dataset. The streaming distance is evaluated as discussed in Section 10.3 (in our case, Stream-DTW*), in order to map each point of the stream v_t into a positive real describing the distance between p and the stream.

However, we do not need to store all distance values. As noted earlier, in order to know if p will trigger on segment s, we just need to compare the threshold to the *minimum* distance on this segment's points. Hence, when sliding through the stream v_t to compute the distance, taking note when a segment ends, we only store the minimum since it began, and forget other non-minimal values which are useless from now on.

This should yield a sequence m_p ("minima") of positive numbers where each index s is a segment and each element $m_p[s]$ is the minimum. In terms of memory occupation, it is way more economical to store just minima (as many values as segments) compared to all distances for each time t (as many values as stream points).

All sequences $m_p = (m_p[s])_s$ should be computed for each instance p.

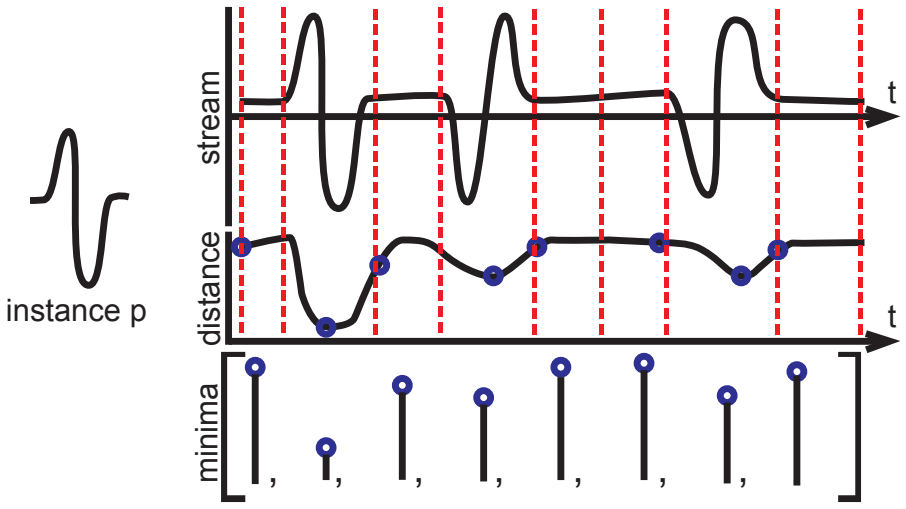


Figure 10.5: Minima of distances are sufficient to determine whether an instance will be triggered during an segment.

10.6.4 Event computation

As seen in Section 10.4, threshold adjustment is a matter of compromise between too tolerant (high threshold) and too strict (low threshold) situations. Therefore, in order to set the threshold accordingly, we propose to list *events*, that is, the segment and threshold at which instances will be triggered (i.e. the minimum of each instance on each segment).

Therefore, an *event* is a tuple (τ, p, s, g) representing the assertion "On segment s, instance p will trigger its label $\ell(p)$ at threshold τ, which is g". The "goodness" g takes one of two values, GOOD if those labels are the same and BAD otherwise. Also, in the following, we will ensure that if a label ℓ is tolerated on segment s, then there are no event linking ℓ and s, because it is inherently neither good or bad to trigger, or not trigger, a label where it is not expected but tolerated. Note that the trigger threshold for each (p, s) pair is just the minimum of p on s: $\tau = m_p[s]$.

In order to find out these events, we need to iterate on all instances p and all segments s:

1. if s *tolerates* current label $\ell(p)$, skip this segment.
2. if "labels don't match", i.e. (s expects $\varnothing$) or (s expects $\ell_s \neq \varnothing$ and $\ell(p) \neq \ell_s$), then register a BAD event with threshold $m_p[s]$.
3. if "labels match", i.e. s expects $\ell_s \neq \varnothing$ and $\ell(p) = \ell_s$, then register a GOOD event with threshold $m_p[s]$.

While iterating through p and s to find out those events, we store them in two data structures:

- Events per label and per segment: let $E[\ell, s]$ be a vector storing all events whose segment is s and whose instance has label ℓ.
- Bad events per label: let $B[\ell]$ be a vector storing all BAD events related to instances with label ℓ.

The same event can be stored in both $E[\ell, s]$ and $B[\ell]$.

It is required that each of these individual vectors are sorted by increasing τ value. Intuitively, it provides a way to "slide" from lowest to highest threshold and discover, in order, which kind of event will happen as we increase the threshold. See Figure 10.6.

10.6.5 Promising subsets identification

Consider we work on a given label ℓ; the vector of bad events $B[\ell]$, after sorting, represents a ranking of worst instances. Indeed, the first event in $B[\ell]$ has a low threshold τ and thus describes which instance is going to generate the first false positive, as we start from threshold $\tau = 0$ and increase progressively. Therefore, when we want to find which instances are worth removing, looking at $B[\ell]$ provides us with an ordering of candidates that should be preferred for removal. When a candidate p is removed, it will no more generate a false positive, which in turn allows us to increase the threshold. However, it could turn out to

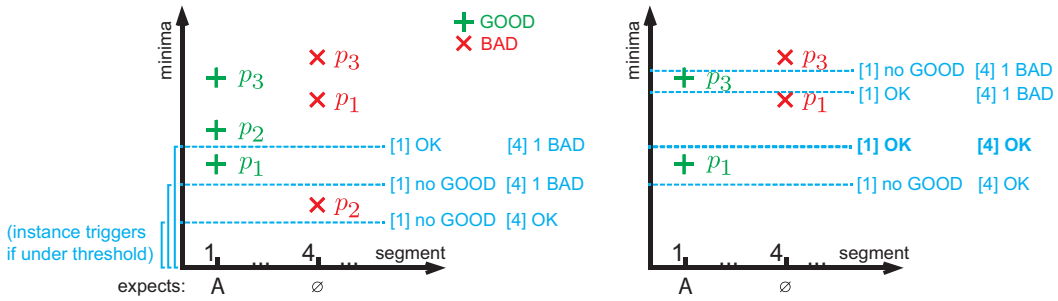


Figure 10.6: Events $E[\ell, s]$ for a fixed label $\ell = \text{A}$ and segments $s = 1, 4$. We look for an optimal threshold, where there would be at least one GOOD event for segments expecting A, and no BAD event elsewhere. The first situation admits no optimal threshold; however, after pruning instance p_2, such a threshold appears.

generate a false negative at some other place where this instance p was needed for the detection. We will take care of this issue in the next section by globally analyzing the performance of removing one or several candidates.

Consider that all possible instances for label ℓ are stored in a set called P_ℓ. The number of instances for this label is therefore $|P_\ell|$. We wish to select a good subset $P'_\ell \subseteq P_\ell$. Instead of brute forcing through all $2^{|P_\ell|}$ subsets, we propose an efficient strategy (although not theoretically optimal) to prune bad instances. It works in linear $\mathcal{O}(|P_\ell|)$ time and thus is tractable even in presence many instances. The idea is to extract, in order, instances from BAD events in $B[\ell]$: say the candidates are, in order, $p_1, p_2, \ldots$ the subsets we propose are $P^0_\ell = \{p_1, p_2, p_3, ...\}, P^1_\ell = \{p_2, p_3, ...\}, P^2_\ell = \{p_3, ...\}, ...$, i.e. P^k_ℓ is the full set P_ℓ without the k worst instances $p_1, ..., p_k$. These are what we call "promising subsets" and they will be analyzed quickly during the next pass to select the best subset among them.

10.6.6 Subsets scoring

Thanks to stream segmentation, we can transform the ill-defined spotting problem zn a collection of well-defined binary classification problems. We proceed as follows: for a given label ℓ, the binary classification problem is to assign either label ℓ or $\varnothing$ to each segment, which expects either ℓ or $\varnothing$ (we turn non-ℓ labels into $\varnothing$ to focus on ℓ only). Furthermore we consider that we are given a subset P'_ℓ of instances. In order to compare these subsets without having to set a manual threshold value, our solution is to compute the ROC (Receiver Operator Characteristic) curve of this binary classification problem and use the AUC (Area Under the Curve) as a quantitative measure to compare subset quality, as in [224].

In order to compute this ROC curve, the fastest solution is to list when each segment will turn from True Negative (TN) to False Positive (FP), or from False Negative (FN) to True Positive (TP). We call these sub-events "switches". This is easily done: for each segment, $E[\ell, s]$ lists the events in order of thresholds; hence the switch event is the event with the lowest threshold (i.e. the first to happen). Thus, it is the first element of $E[\ell, s]$ for which the instance p is included in our subset P'_ℓ. That gives one switch per segment, except for segments tolerating the current label, which do not participate in the binary classification

and are simply ignored.

Now that all switches are established, computing the ROC curve and its AUC is as easy as reading the switches in increasing threshold order. A GOOD switch turns FN to TP (up on the ROC curve); a BAD switch turns TN to FP (right on the ROC curve).

Finally, the best subset is the one with the highest AUC. The optimal threshold is selected by finding an optimal point on the ROC curve, but how we define "optimal" does not have a definite answer. In our case, we consider the point on the curve maximizing an F-score. One can choose either F_1 or F_β for a well-chosen β.

10.7 Experimental analysis of the algorithm behavior

10.7.1 Experiment 1: with fake gesture injection

We ran the instance selection algorithm on motion data captured by the commercial glove V-Hand 3.0. At this time gesture boundaries were added by manual post-processing. Training is composed of 100 Hz streams recorded by two people, totalling 6m40s; testing is composed of 3m04s by a third person.

Some weak instances appear on their own, but in order to better display our algorithm's value, we further injected "fake" time series in the database, taken by reading a random piece of the training stream and giving it a random existing label. This makes the task more difficult by lowering the overall dataset quality.

In Figure 10.7, we show on a single class that our technique is successful in detecting all bad instances and removing them, attaining here an ideal AUC of 1 (which is an objective that is not always reachable). Plotting the AUC of each ROC curve as we progressively remove instances gives Figure 10.8, in which we note that all classes also end up attaining $\mathrm{AUC} = 1$.

Analyzing the dataset on the 3 minute testing stream took only 6.15s on a regular CPU, including extraction, streaming DTW calculation, segmentation, events computation and ROC scoring for 215 identified promising subsets.

10.7.2 Conclusion

With this algorithm, we have shown that it is advantageous to tackle the multiclass streaming recognition problem as a collection of binary classification problems: one for each class. This interpretation enabled us to derive an original algorithm for proposing an efficient joint heuristic to solve the NP-hard instance selection problem and perform threshold tuning. Cutting the stream in intervals allows us to use ROC analysis in order to detect low-quality instances and set the distance threshold by selecting the optimal ROC point, based on the F-score metric. This solution is implemented in our real-time gesture recognition pipeline.

In regards with our gesture recognition application, running the analysis on segments rather that full stream turns out to be fast enough to prune instances in parallel while learning the gestures, which is the case during real-world usage of our gesture recognition GUI. Moreover, the main interest of our technique is to require less tuning from the user. Indeed, with this novel procedure, not only incorrect gestures are removed, but the system also finds the class thresholds instead of relying on brittle hand-crafted values. The only

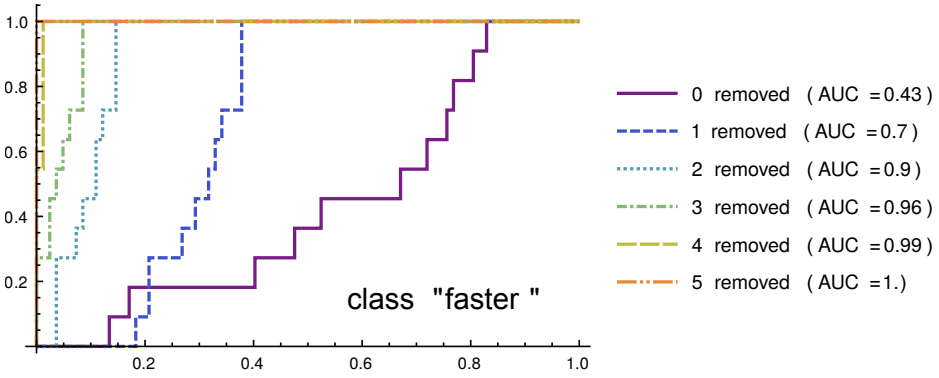


Figure 10.7: ROC curves for one class of the gesture data set in which fake instances were introduced. The base dataset ("0 removed") shows poor performance; our method is able to detect the 5 bad instances and enhance the ROC curve as instances are progressively removed from the base dataset.

Figure 10.8: Evolution of AUC as low-quality instances are removed. For all classes (including those not shown here), we are able to significantly increase AUC (maximum at red point) compared to the original dataset (leftmost point).

thing required from the user is to input gestures: the system takes care of the hyperparameters. Experiments showed that accuracy was high while the user is standing, and stayed high even when the user is in mobility.

10.8 Additional experiments

Now that we have the advanced training procedure, the system reaches a point where it can be used without further user intervention other than providing raw gesture data. In this section, we present more experiments in order to analyze not only the advanced training procedure itself, but the gesture recognition system as a whole. We also investigate how the system reacts to different mobility conditions and whether it performs well for distinct users.

10.8.1 Experiment 2: analysis of gesture executed without mobility (standing)

In this experiment, we aim to show how our algorithm performs in the same conditions than our real-time GUI recognition setup. For now on, the training stream and the validation stream will be the same: we can say that the algorithm self-validates the train stream against itself. This is still interesting, because the training set is only composed of the extracted gestures (to be subset-selected afterwards) but the stream contains more information: in particular, all intervals where no gesture was performed. These "blank" intervals are crucial since they provide some "non-gesture" stream data; and *in fine*, the role of the threshold is to separate gestures from non-gestures. Therefore, reusing the training stream as the validation stream to perform subset selection and threshold computation makes sense and will be done for all subsequent experiments. For threshold selection on the ROC curve, we chose to use the point based on the F_β metric; here we used $\beta = 3$ in order to prefer FNs than FPs. The rationale is that in real world usage, the user can repeat a missed gesture, but it is more difficult to undo a wrong detection. In other words, we aim to keep a high precision even if the recall must decrease.

We have 6 gesture streams each containing 4 instances of the following 9 gestures: *go*, *stop*, *back*, *slower*, *faster*, *left*, *right*, *round* and *come*. Therefore, one finds 36 instances in each stream. The recognition GUI asked a single user to perform gestures in specific intervals so that ground truth labels are marked automatically and not via manual post-processing. We ran a cross-validation procedure by using each combination of three streams for training and three streams for testing, leading to 20 cross-validation passes.

Results shown in Table 10.1 show promising recognition rates: Precision: 96.5% / Recall: 96.2% (with standard deviations on all 20 cross-validation passes $\sigma_P = 2.18\%, \sigma_R = 2.28\%$). It indicates that the automatic procedure described in this chapter is able to recover consistent threshold values for each class, thereby leading to balanced results. In comparison with the experiment of Chapter 9 where we used to set the thresholds manually (Precision: 96.0% / Recall: 96.0%), our system is able to keep similar precision and recall. Also, the results indicate that 3 training stream captures (where a 4 instances of each gesture class are contained in the stream) are enough to teach the system with good recognition rates. With our recognition GUI, 3 captures are recorded in around 5 to 10 minutes.

Label	FP	FN	TP	Precision / Recall
faster	0	9	231	100.0% / 96.2%
come	5	2	238	97.9% / 99.1%
round	9	1	239	96.3% / 99.5%
stop	34	12	228	87.0% / 95.0%
back	15	2	238	94.0% / 99.1%
left	1	17	223	99.5% / 92.9%
go	2	19	221	99.1% / 92.0%
slower	9	18	222	96.1% / 92.5%
right	0	0	240	100.0% / 100.0%
All labels	75	80	2080	**96.5% / 96.2%**

Table 10.1: Results of single-operator recognition on gesture data performed while **standing**, with our custom glove, using the automated procedure described in this chapter. Each pass represents a combination (3 train, 3 test) out of 6 gesture streams.

10.8.2 Experiment 3: analysis of gesture executed in mobility

This experiment is very similar to the one presented in the previous subsection. Here, the goal is to investigate if our gesture recognition pipeline, equipped with the advanced training procedure presented above, is sufficient to recognize gestures in mobility. We therefore took 8 stream captures in mobility, combining walk and occasional light run. Once again, a single user is tested, therefore the results we obtain give us information about the performance of only one user's training, which is how the system is meant to be deployed. The operator is the same as Experiment 2.

Label	FP	FN	TP	Precision / Recall
back	1	0	1120	99.9% / 100.0%
come	57	143	977	94.4% / 87.2%
faster	14	46	1074	98.7% / 95.8%
go	110	150	970	89.8% / 86.6%
left	24	77	1043	97.7% / 93.1%
right	35	91	1029	96.7% / 91.8%
round	10	56	1064	99.0% / 95.0%
slower	7	40	1080	99.3% / 96.4%
stop	0	13	1107	100.0% / 98.8%
All labels	258	616	9464	**97.3% / 93.8%**

Table 10.2: Results of training and testing gesture **in mobility** (walking and occasional light run). Cross-validation of all (3 train, 5 test) combinations. Recognition rates stay high, which indicates good performance of both our Stream-DTW* pipeline and the advanced training technique of this chapter.

The 8 captures stream follow a similar format. We use the same gesture dictionary with 9 classes (*go*, *stop*, *back*, *slower*, *faster*, *left*, *right*, *round* and *come*) and each capture stream contains 9 classes $\times$ 4 repetitions = 36 instances. Labels were again added by the recording GUI: it asks the user to perform the ground truth gestures in a specific interval, so that no manual annotation is needed. The cross-validation procedure is similar to the previous experiment: we consider all combinations of 3 training capture streams out of 8, and the test set is composed of the remaining 5 streams. This leads to 56 cross-validation passes.

In Table 10.2, we can see the performance of our general pipeline, including Stream-DTW* and this advanced training procedure. Results show good precision and recall: more than 97.3% of detected gestures were expected and 93.8% of the expected gestures were detected. (Standard variations: $\sigma_P = 1.87, \sigma_R = 4.11$.) As before, we set $\beta = 3$ (i.e. the optimal threshold on the ROC curves is selected by choosing the ROC point attaining maximal $F_\beta = F_3$), therefore the precision being higher than the recall is expected since we prefer missing gestures rather than making spurious detections.

As we expected by designing the whole glove-based system, our gesture recognition pipeline works with a good accuracy when the operator is in mobility. Compared to the standing experiment (Table 10.1), the average recall loses 2.4 points, which indicates gestures might be slightly more difficult to detect when there is no instance in mobility. However, due to our choice of $\beta = 3$, the system tends to keep a high precision, whose average increase by 0.8 point.

10.8.3 Experiment 4: testing in an unseen condition

The data from Experiments 2 and 3 was reused to check how well the gesture pipeline responds to a new, unseen mobility condition. In this case, the two conditions available are *standing* (Experiment 2) and *mobile* (Experiment 3). In the first case, the system is given only *standing* gestures for the training phase, but is tested against gestures executed in mobility. The second case is the opposite: training data is *mobile*, testing is *standing*. The difficulty resides in the fact that gestures may be executed slightly differently according to the *standing* or *mobile* condition, and can have different intra-class variability, thus requiring more tolerance, which translates as higher thresholds.

As with previous experiments, there are several capture streams of each 36 gestures (9 classes $\times$ 4 instances) in two mobility conditions: 6 tagged *standing* and 8 tagged *mobile*. For each of the two cases, we took every combination of 3 captures streams among all available for training, while the testing set consisted of all available streams for the tested condition. Therefore, for the first case, all 20 passes of 3 out of 6 *standing* training streams were used against all 8 *mobile* streams, and the second case is the opposite: all 56 passes of 3 out of 8 *mobile* streams were used against all 6 *standing* streams.

Label	FP	FN	TP	Precision / Recall
back	1	22	618	99.8% / 96.5%
come	2	232	408	99.5% / 63.7%
faster	16	125	515	96.9% / 80.4%
go	2	341	299	99.3% / 46.7%
left	38	0	640	94.3% / 100%
right	20	0	640	96.9% / 100%
round	37	41	599	94.1% / 93.5%
slower	0	237	403	100% / 62.9%
stop	21	67	573	96.4% / 89.5%
All labels	137	1065	4695	**97.1% / 81.5%**

Table 10.3: The system is trained with gestures performed while standing only, but is evaluated against gestures performed in mobility.

In Table 10.3, the system only sees *standing* gestures. When presented gestures performed in mobility, the precision stays high at 97.1% ($\sigma_P = 1.52$) which means most gestures that are triggered are indeed correct. However, the recall drops at 81.5% ($\sigma_R = 3.89$), which indicates the system is very conservative and misses some gestures which should be detected. This suggests one should train the system with a few gestures from the condition that will be seen in testing. Looking in more detail, some labels are more problematic than others, in particular, *go*, *slower*, *come* and to some extent, *faster* have low recall scores. Unsurprisingly, these are dynamic gestures, and thus a possible explanation would be that dynamic gestures are performed with more variation when in mobility, due to the action of the whole body.

Label	FP	FN	TP	Precision / Recall
back	147	24	1320	89.9% / 98.2%
come	279	42	1302	82.3% / 96.8%
faster	23	73	1271	98.2% / 94.5%
go	165	90	1254	88.3% / 93.3%
left	38	238	1106	96.6% / 82.2%
right	0	82	1262	100% / 93.8%
round	26	53	1291	98.0% / 96.0%
slower	293	20	1324	81.8% / 98.5%
stop	162	33	1311	89.0% / 97.5%
All labels	1133	655	11441	**90.9% / 94.5%**

Table 10.4: The system is trained with gestures performed in mobility only, but is evaluated against gestures performed while standing.

The opposite experiment, presented in Table 10.4, gives mobile gestures for training. This time, the scores are consistent with previous experiments, since both precision and recall are above 90%: Precision = 90.9% ($\sigma_P = 4.63\%$), Recall = 94.5% ($\sigma_R = 3.24\%$). Compared to Table 10.4, this suggests that for the training stage, *mobile* data fare better than *standing* ones.

10.8.4 Experiment 5: usability by novice users

Experiments 1 to 4 reflect captures performed by an user having some experience with the gesture recognition system. In this last experiment, we aim to provide some evidence that the gesture recognition system is also usable by novice users with no prior experience. To do this, two users were given the gesture dictionary with an appropriate explanation of the gestures. We call these users A and B, for this subsection only. They were allowed to rehearse for 30 to 60 minutes, first without the recording GUI and then with the GUI, which provides not only a feeling of the recording tool but also an interactive feedback on whether performed gestures are well recognized, making it possible for the user to grasp how the system reacts to recordings. Thanks to a colorful visualization of the sensors, they could also understand what the system "sees" when fingers are flexed or when the hand moves.

In the end, 6 capture streams of 36 gestures (9 classes $\times$ 4 instances) were kept: 3 in *standing* condition where users didn't move and 3 in *mobile* condition where they were asked to walk in a sinuous path within a restrained space. Since the goal is not to penalize users, they were allowed to restart a recording if they felt they did not perform the gestures well enough.

As for the previous experiments, the advanced training procedure was used, with threshold selection on optimal ROC curve chosen with highest F_β measure with $\beta = 3$.

For each user, we made three sub-analyses:

a. Standing condition analysis. The 3 capture streams were cross-validated in the configuration (train = 2 standing, test = 1 standing), leading 3 passes $\times$ 1 test capture stream $\times$ 36 instances = 108 instances to detect.

b. Motion condition analysis. Similar cross-validation as above: (train = 2 mobile, test = 1 mobile) leading to 3 passes and again, 108 instances to detect.

c. Multi-condition analysis. All combinations of (train = {1 standing, 1 mobile}, test = {2 standing, 2 mobile}) were gathered, leading to 9 passes with 4 test capture streams, thus $9 \times 4 \times 36 = 1296$ instances to detect. This experiment aims to check whether the system can be efficiently trained with one capture stream of each condition and then converge to a single trained model suitable for any mobility condition.

User	Condition	FP	FN	TP	Precision / Recall	σ_P, σ_R
A	Standing (a)	6	7	101	**94.3% / 93.5%**	2.85%, 4.28%
A	Motion (b)	7	2	106	**93.8% / 98.1%**	5.48%, 3.23%
A	Multi (c)	69	57	1239	**94.7% / 95.6%**	1.88%, 2.42%
B	Standing (a)	11	10	98	**89.9% / 90.7%**	8.52%, 8.94%
B	Motion (b)	7	2	106	**93.8% / 98.1%**	3.64%, 1.62%
B	Multi (c)	113	82	1214	**91.4% / 93.6%**	2.32%, 5.26%

Table 10.5: Results of recognition by novice users after one hour of acquaintance with the gestures and the recording GUI. In all mobility conditions, users reach good recognition rates, enough for real-world usage.

Table 10.5 shows that all experiments reach a Precision/Recall ratio of at least 90% for each user (except one precision at 89.9%) and generally higher. This suggests the system can be given to untrained users who, given appropriate explanation, can reach good recognition rates in each mobility condition within less than an hour of acquaintance with the gesture dictionary and the recognition system. Furthermore, multi-condition analysis (c.) shows that the system can recognize gestures adequately when given as few as 1 capture stream of each condition during training, which is quite short to acquire.

10.9 Conclusion of experiments

These additional experiments indicate that one can expect more than 90% for both Precision and Recall on both standing and mobile conditions. Since the whole pipeline is used, these experiments validate our choices regarding sensor selection, sensor processing and feature engineering (including the decision to use orientation as a first-class feature), as well as Stream-DTW* for normalized distance computation between time series. Ultimately, it validates our advanced training procedure for a fully automated system: the system keeps the best instances and more importantly, detects optimal threshold for each class instead of requiring hand-tuning from the user.

Chapter 11

Coarse-DTW and Bubble downsampling

Contents

As we have seen in Chapter 2, our work is designed for possible use in a robotic context; therefore embedded constraints are worth exploring, because power and CPU are scarce. In this scope, we conducted some research on enhancing the representation of the gesture time series. In this chapter, we present the following related innovations. First, we investigate an alternative representation: *sparse* time series, where similar values are omitted; it is extended without difficulty to streams. It serves as a compression scheme compatible with our streaming requirement. Second, we bring some modifications to the DTW algorithm in order to cope with sparse time series. Third, we present a downsampling technique that turns classic time series and streams into their sparse counterparts.

Combining these three insights allows for a reduction in the CPU footprint of our gesture recognition pipeline. However, we shall note that such enhancements are facultative; it is very much possible to implement our pipeline without the ideas from this chapter, since Stream-DTW and Stream-DTW* already work in real time without further compression.

This chapter's content is mainly drawn from a paper we published in the AALTD (Advanced Analysis and Learning on Temporal Data) workshop of ECML-PKDD 2015. As for

the previous chapter, the techniques we present are designed to work in more general time series contexts beyond gesture recognition.

11.1 Introduction

One of the main drawbacks of DTW is its quadratic computational complexity which, as is, may prevent processing a very large amount of lengthy temporal data. Recent research has thus mainly focused on circumventing this complexity barrier. The original approach proposed in this chapter is to cope directly and explicitly with the potential sparsity of the time series during their *time-elastic* alignment.

Our main contribution is two-fold: i) an on-line downsampling algorithm whose aim is to provide a sparse representation of time series in constant time for each data point; ii) Coarse-DTW, a DTW variant that copes efficiently with the sparsity of time series. Furthermore, an experimentation section is provided that highlights the tradeoff between speedup and accuracy on a large number of time series datasets (mainly unidimensional with a few multidimensional) and discusses the benefits of our approach in the gesture recognition pipeline.

11.2 Previous work

DTW speed-up approaches mostly fall into one or several of the following categories:

1. **Reducing the search space**, when looking for the optimal alignment path. In [188] and [97] the search space is reduced by using a fixed corridor with a band (resp. parallelogram) shape displayed around the main alignment diagonal. For these corridor approaches, finding the optimal alignment path is obviously not guaranteed. Recently, [153] proposed the SparseDTW algorithm that exploits the concept of sparse alignment matrix to dynamically reduce the search space without optimality loss. SparseDTW provides thus an exact DTW computation with efficiency improvement in average.

2. **Reducing the dimensionality** of the data, along the spatial or temporal axis. Reducing the time series along the temporal axis leads to a straightforward speed-up (reducing by 2 the number of samples provides a by 4 speedup). [249] and [111] have proposed a piecewise aggregate approximation (PAA) of time series using segments of constant size. In [108] a modification of DTW, called PDTW, has been proposed to cope explicitly with PAA. Adaptive Piecewise Constant Approximation (APCA) has also been used [30] to comprees furthermore the representation of time series. Symbolic representation of time series such as SAX [165] or its variants falls also in this category. Recently, in [137] authors have shown that in the context of isolated gesture recognition sampled using depth-camera or motion capture systems, drastic down-sampling along the time axis in general enhances the accuracy.

3. **Approaching DTW by a low complexity lower bounding function** in an early abandoning strategy to reduce (drastically) the number of DTW calculations. This idea has been first proposed by [114] then progressively improved by [109], [123] that have successively introduced tighter lower bounding functions.

Some mixed approaches have been also proposed such as in ID-DTW (Iterative Deepening DTW) [39] which uses multi-resolution approximations with an early abandoning strategy or Fast-DTW [192] which also exploits multi-resolution approximations on a divide-and-conquer principle. [191] and [197] have also proposed approaches mixing APCA with a lower bounding strategy.

Obviously, optimizing the accuracy/speedup tradeoff is thus a natural concern, and within this line of research, mixing speedup strategies with a dedicated DTW variant seems quite promising. This chapter specifically contributes to this focus.

11.3 Presentation of Coarse-DTW

11.3.1 Classical DTW

Let us first review the classical formulation of DTW. If d is a fixed positive integer, we define a *dense time series* of length l as a multidimensional sequence $(r[i])$, i.e. :

$$r : \{1, \dots, l\} \to \mathbb{R}^d. \tag{11.1}$$

Let $r_1 = (r_1[i_1])_{0 \leq i_1 \leq l_1}$ and $r_2 = (r_2[i_2])_{0 \leq i_2 \leq l_2}$ be two dense time series with respective lengths l_1 and l_2. A *warping path* $\gamma = (\gamma_k)$ of length p is a sequence

$$\gamma : \{1, \dots, p\} \to \{1, \dots, l_1\} \times \{1, \dots, l_2\} \tag{11.2}$$

such that (denoting the k-th element of γ as γ^k, and denoting $\gamma^k = (i_1^k, i_2^k)$)

$$\begin{aligned} \gamma^1 &= (1, 1); \\ \gamma^p &= (l_1, l_2); \text{ and} \end{aligned} \tag{11.3}$$

and for all k in $\{1, \dots, p-1\}$,

$$\gamma^{k+1} = (i_1^{k+1}, i_2^{k+1}) \in \{ \begin{array}{ll} (i_1^k + 1, & i_2^k), \\ (i_1^k, & i_2^k + 1), \\ (i_1^k + 1, & i_2^k + 1)\} \end{array} \tag{11.4}$$

In other words, a warping path is required to travel along both time series from their beginnings to their ends; it cannot skip a point, but it can advance one timestep on one series without advancing the other, effectively amounting to "time-warping".

Let us define the *cost* of a warping path γ as the sum of distances between pairwise elements of the time series along γ, i.e.:

$$\text{cost}(\gamma) = \sum_{(i_1^k, i_2^k) \in \gamma} \delta(r_1[i_1^k], r_2[i_2^k])_2^2 \tag{11.5}$$

For readability, we will denote the squared Euclidean distance used between two points x, y of a time series as follows:

$$\delta(x, y) = \|x - y\|_2^2 = \sum_{m=1}^{d} (x_m - y_m)^2. \tag{11.6}$$

However, δ could be any distance on $\mathbb{R}^d$.

The warping path has a finite length, and there is a finite number of possible warping paths. Hence, there is at least one path whose cost is minimal. We define $\mathrm{DTW}(r_1, r_2)$ as the minimal cost of all warping paths.

In practice, the typical way to compute DTW leverages the recursive structure of the warping path:

Algorithm 4 DTW

procedure DTW(r_1, r_2) ▷ (r_1 and r_2 are 1-indexed; A is 0-indexed)
  $A =$ new matrix $[0..l_1, 0..l_2]$
  $A[0, .] = A[., 0] = \infty$ and $A[0, 0] = 0$
  for $i_1 = 1$ to l_1 **do**
    for $i_2 = 1$ to l_2 **do**
      $A[i_1, i_2] = \delta(r_1[i_1], r_2[i_2]) + \min(A[i_1-1, i_2],\ A[i_1, i_2-1],\ A[i_1-1, i_2-1])$
    end for
  end for
  return $A[l_1, l_2]$
end procedure

11.3.2 Sparse time series

In contrast to dense time series, let us define a *sparse time series* as a pair of sequences with the same length, $s = (s[i])_{0 \le i \le l}$ and $r = (r[i])_{0 \le i \le l}$:

$$\begin{aligned} s &: \{1, \dots, n\} \to \mathbb{R}_+ \\ r &: \{1, \dots, n\} \to \mathbb{R}^d \end{aligned} \tag{11.7}$$

The sequence r represents our multidimensional signal's values, like above; the novelty resides in s, where $s[i]$ is a positive number describing *how long the value $r[i]$ lasts*. We call this number $s[i]$ the *stay* of $r[i]$. In the following, we will also denote a sparse time series as a sequence of pairs: $\{(s[0], r[0]), \dots, (s[l], r[l])\}$.

For example, every dense time series $(r[i])$ is exactly represented by the sparse time series with the same values $r[i]$ and all stays $s[i] = 1$. As another example, the 2D dense time series $\{(0.5, 1.2), (0.5, 1.2), (0.3, 1.5)\}$ is equivalent to the 2D sparse time series $\{(2, (0.5, 1.2)), (1, (0.3, 1.5))\}$.

11.3.3 Coarse-DTW

We may now introduce the Coarse-DTW algorithm. It takes two sparse time series: $(s_1[i_1], r_1[i_1])$ of length l_1, and $(s_2[i_2], r_2[i_2])$ of length l_2. The main idea behind Coarse-DTW is

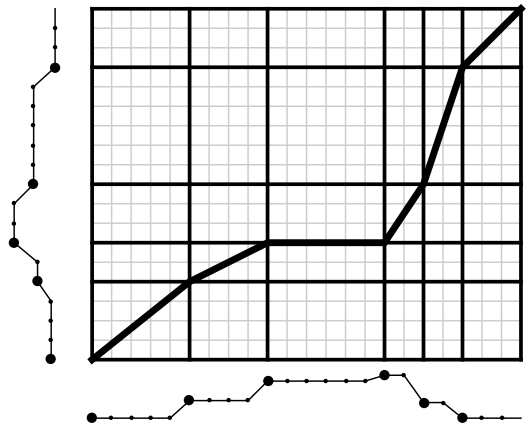


Figure 11.1: A warping path in Coarse-DTW. We superimposed the sparse time series (bigger points) on top of their equivalent dense time series (smaller points). The coarse, thick grid is the Coarse-DTW matrix, whereas the underlying thin grid is the classical DTW cost matrix.

the aggregation of very similar samples in time and space. As such, it shares similar aspects with the approach developed in [5] for symbolic time series.

Algorithm 5 Coarse-DTW

procedure COARSE-DTW($(s_1, r_1), (s_2, r_2)$)
  $A =$ new matrix $[0..l_1, 0..l_2]$
  $A[0, .] = A[., 0] = \infty$ and $A[0, 0] = 0$
  for $i_1 = 1$ to l_1 **do**
    for $i_2 = 1$ to l_2 **do**
      $A[i_1, i_2] = \min($ $s_1[i_1]$. $\delta(r_1[i_1], r_2[i_2])$ + $A[i_1-1, i_2]$,
        $s_2[i_2]$. $\delta(r_1[i_1], r_2[i_2])$ + $A[i_1, i_2-1]$,
        $\phi(s_1[i_1], s_2[i_2])$. $\delta(r_1[i_1], r_2[i_2])$ + $A[i_1-1, i_2-1]$ $)$
    end for
  end for
  return $A[l_1, l_2]$
end procedure

Like classical DTW, we need to define a length-agnostic version of Coarse-DTW (Section 9.1.3). The suitable scaling factor is the duration of the time series. While it is no longer equal to the length l, fortunately, the duration can still be expressed as the sum of all stays: $\sum_{i=1}^{l} s[i]$. Thus, we define:

$$\text{Coarse-DTW}^*(r_1, r_2) = \frac{\text{Coarse-DTW}(r_1, r_2)}{\sqrt{\left(\sum_{i_1=1}^{l_1} s_1[i_1]\right)^2 + \left(\sum_{i_2=1}^{l_2} s_2[i_2]\right)^2}} \tag{11.8}$$

11.3.4 Weights of costs

Coarse-DTW takes advantage of the sparsity in the time series to calculate costs efficiently. However, because the points last for different amount of time, we must adapt the classical

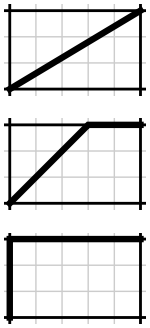
DTW formulation in order to account for the stays $s_1[i_1]$ and $s_2[i_2]$ of each point into the aggregate cost calculation.

Obviously, when a point lasts for a long time, it should cost more than a point which lasts for a brief amount of time. For this reason, the pure cost $\delta(r_1[i_1], r_2[i_2])$ is multiplied by some quantity, called *weight*, linked to how long the points last, as in lines 6–8 of the algorithm. The goal of this subsection is to explain why we set those weights to $s_1[i_1]$, $s_2[i_2]$, and $\phi(s_1[i_1], s_2[i_2])$ respectively.

One Coarse-DTW iteration actually relates to a batch of DTW iterations in a *constant-cost sub-rectangle*. Indeed, suppose that we are operating on two pairs of subsequent points in a sparse time series, say, $(s_1[i_1], r_1[i_1])$, $(s_1[i_1+1], r_1[i_1+1])$ on one time series and $(s_2[i_2], r_2[i_2])$, $(s_2[i_2+1], r_2[i_2+1])$ on the other. If the time series was not sparse, as in classical DTW, there would be several repetitions of the first point $r_1[i_1]$, namely $s_1[i_1]$ times, until it moves to the next value $r_1[i_1+1]$. Similarly $r_2[i_2]$ would be repeated $s_2[i_2]$ times. In the DTW cost matrix, this would create an $s_1[i_1] \times s_2[i_2]$ sub-rectangle where all costs are identical because they match the same values; the constant cost here being $\delta(r_1[i_1], r_2[i_2])$; this is our constant-cost sub-rectangle (see Fig. 11.2).

The choice of weights $s_1[i_1]$ and $s_2[i_2]$ in lines 6 and 7 is motivated as follows: when we advance one time series without advancing the other, we want a lengthy point to cost more than a brief point. In the DTW constant-cost sub-rectangle, advancing the first time series is like following a horizontal subpath, whose aggregated cost would be $\delta(r_1[i_1], r_2[i_2])$ on each of its $s_1[i_1]$ cells. This sums up to $s_1[i_1].\delta(r_1[i_1], r_2[i_2])$, which is why the weight is chosen to be $s_1[i_1]$ in line 6. An analog interpretation holds for a vertical subpath of $s_2[i_2]$ cells.

For the third case (line 8), namely advancing *both* time series, choosing a weight for the cost is less obvious. The question is: which weight should we set for a path joining the top-right corner to the bottom-left one, in a constant-cost sub-rectangle? This question is captured by $\phi(s_1[i_1], s_2[i_2])$, the weight of the cost in this situation. We propose three choices for ϕ, all of which have a geometrical interpretation in the classical DTW cost matrix.



$$\phi_{\text{diag}}(s_1[i_1], s_2[i_2]) = \sqrt{s_1[i_1]^2 + s_2[i_2]^2}$$

$$\phi_{\max}(s_1[i_1], s_2[i_2]) = \max(s_1[i_1], s_2[i_2])$$

$$\phi_{\text{stairs}}(s_1[i_1], s_2[i_2]) = s_1[i_1] + s_2[i_2] - 1$$

Figure 11.2: Three choices for the function ϕ, in a constant-cost sub-rectangle of width $s_1[i_1]$ and height $s_2[i_2]$.

Our first proposition for ϕ seeks to mimic the behavior of classical DTW. In the constant-cost sub-rectangle (of size $s_1[i_1] \times s_2[i_2]$), we know that a path minimizing the aggregated cost is the same as one minimizing the number of cells; precisely because all cells have the same cost. Furthermore, the minimal number of cells is exactly $\max(s_1[i_1], s_2[i_2])$; take, for example, a path going diagonal until it reaches the opposite size and then completing the

remaining route on a line (see Fig. 11.2, middle).

Instead of choosing the weight inspired from DTW, which only approximates a diagonal, we can also choose the weight to be the true diagonal of the sub-rectangle, leading to $\phi_{\text{diag}}(s_1[i_1], s_2[i_2]) = \sqrt{s_1[i_1]^2 + s_2[i_2]^2}$.

Finally, we will also consider the choice of $\phi_{\text{stairs}}(s_1[i_1], s_2[i_2]) = s_1[i_1] + s_2[i_2] - 1$, amounting to a version of classical DTW where only horizontal and vertical paths would be allowed (like stairs, hence the name). Note that it should be used only in cases where sparse time series admit stays of 1 or more (this will be the case with our downsampling algorithm introduced in the next section).

11.4 Downsampling

In this section, we seek to transform a dense time series $(r[i])_i$ into a sparse time series $(s[i], r[i])_i$; the goal is to detect when series "move a lot" and "are rather static", adjusting the number of emitted points accordingly. We propose a downsampling algorithm, called *Bubble*, which essentially collapses a block of several similar values into a single value, augmented with the duration of this block.

Bubble downsampling is described in Algorithm 6.

Algorithm 6 Bubble Downsampling

procedure BUBBLE(r, ρ) ▷ $\rho \geq 0$
  $i_{\text{center}} = 1$ ▷ initialize bubble center
  $r_{\text{center}} = r_1$
  $r_{\text{mean}} = r_1$
  for $i = 2$ to n **do**
    $\Delta r = \delta(r[i], r_{\text{center}})$ ▷ distance to center
    $\Delta i = i - i_{\text{center}}$ ▷ find the stay
    if $\Delta r \geq \rho$ **then** ▷ does the bubble "burst"?
      yield $(\Delta i, r_{\text{mean}})$ ▷ emit stay + point
      $i_{\text{center}} = i$ ▷ update bubble center
      $r_{\text{center}} = r[i]$
      $r_{\text{mean}} = r[i]$
    else
      $r_{\text{mean}} = (\Delta i \times r_{\text{mean}} + r[i]) / (\Delta i + 1)$ ▷ update mean
    end if
  end for
  $\Delta i = n - i_{\text{center}} + 1$ ▷ force bursting last bubble
  yield $(\Delta i, r_{\text{mean}})$
end procedure

The idea behind Bubble downsampling lies on the following approximation: consecutive values can be considered equal if they stay within a given radius ρ for the distance δ. We can picture a curve which makes bubbles along its path (see Fig. 11.4), hence the name. Concretely, the algorithm emits a sparse time series, where each stay is the number of consecutive points contained in a given bubble, and each value is the mean of the points in this

bubble.

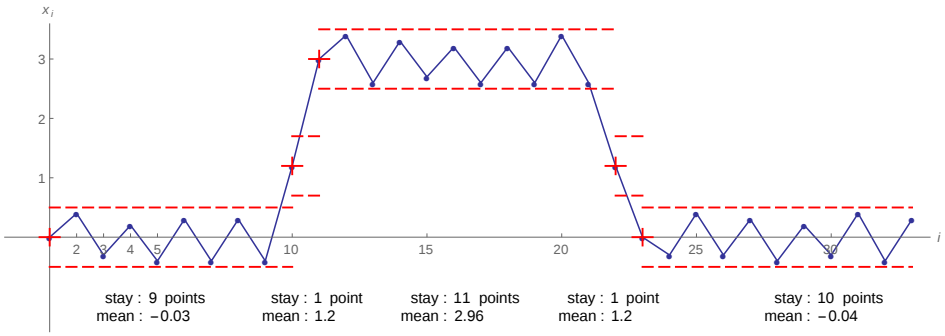


Figure 11.3: Bubble downsampling applied on a 1D time series (blue, solid) with $\rho = 0.5$. The 1-bubbles are represented by their 1-centers (red crosses) and their 1-boundaries (red, dashed lines). The sparse time series emitted is $\{(9, -0.03), (1, 1.2), (11, 2.96), (1, 1.2), (10, -0.04)\}$.

The parameter ρ represents the tradeoff between information loss and density. A large ρ emits few points, thus yielding a very sparse time series, but less accurate; a smaller ρ preserves more information at the expense of a lower downsampling ratio. The degenerate case $\rho = 0$ will output a clone of the original time series with no downsampling (all stays equal to 1). Because speed is a direct consequence of sparsity in Coarse-DTW, a good middle value for ρ must be found, so that time series are as sparse as possible while retaining just the right amount of information.

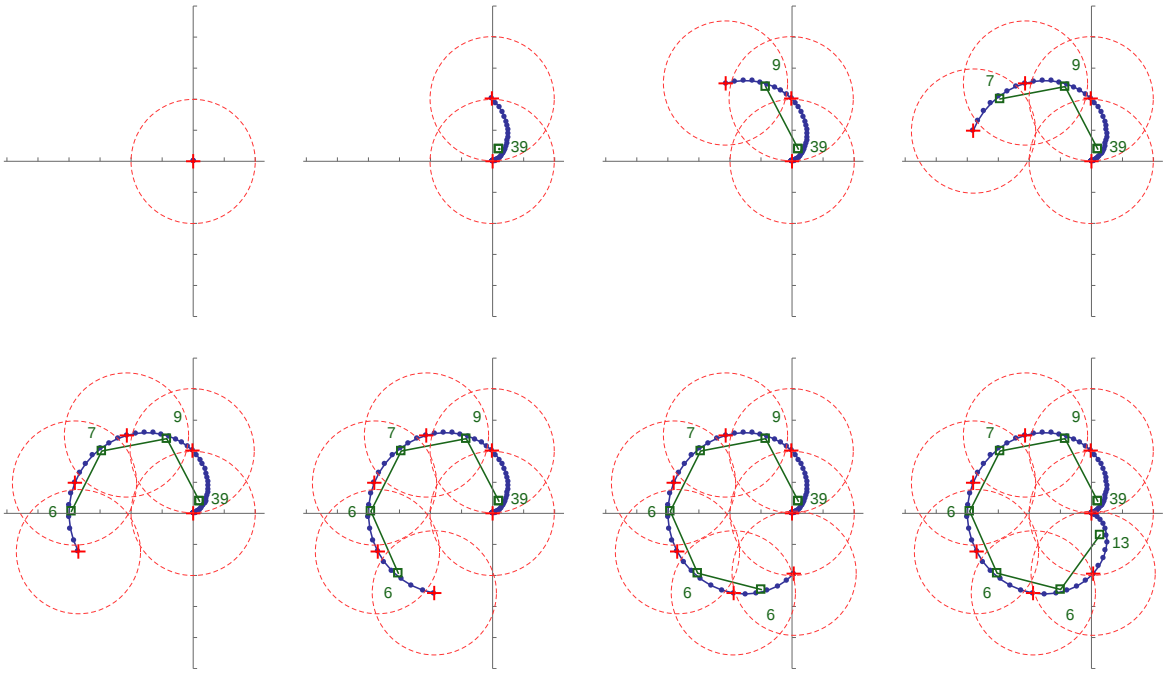


Figure 11.4: Bubble downsampling progressively applied on a 2D time series (outer blue line with dots) with $\rho = 2.0$, along with the sparse time series emitted (inner green line with squares). Again, the 2-bubbles are represented by their 2-centers and their 2-boundaries (red crosses and dashed circles). Numbers indicate the stays. Notice how stays take into account the slowness at the beginning of the signal.

11.5 Optimizations on Coarse-DTW

DTW suffers from a slow computation time if not implemented wisely. For this reason, several optimizations have been designed [109].

The optimizations we considered are called *lower bounds*, designed to early-abandon computations. This kind of optimization mostly makes sense in a classification scenario along with a k-Nearest Neighbor (k-NN) classifier. In the case of 1-NN, as the classification goes on we can track the "best-so-far" distance; if a sample's lower bound exceeds the best-so-far, the computation can be stopped because it is guaranteed not to be a candidate for the nearest neighbor.

The first lower bound $\mathrm{LB}_{\mathrm{Kim}}$ is based on the following remark: whatever the warping path found by DTW, both time series' first and last points will be matched together. More formally,

$$\begin{aligned} \mathrm{DTW}(v,w) &= \sum_{(i,j)\in\gamma} \delta(r_{i_1[k]}, w_{i_2[k]}) \\ &= \delta(v_1,w_1) + \cdots + \delta(v_n,w_m) \\ &\geq \delta(v_1,w_1) + \delta(v_n,w_m) \end{aligned} \tag{11.9}$$

where the ellipsis is a sum of positive numbers. The equation would not be satisfied if both $n \leq 1$ and $m \leq 1$, but fortunately this special case would lead to a trivial computation of DTW rendering this lower bound useless.

In the case of Coarse-DTW, the cost of matching the first points is:

$$\phi(s_1,t_1).\delta(v_1,w_1) \tag{11.10}$$

indeed, the first matching is done diagonally because $A[0,1] = A[1,0] = \infty$. Then, the cost of matching the last points is $\min(s_n, t_m, \phi(s_n,t_m)) \,.\, \delta(v_n,w_m)$.

Therefore, for all pairs of time series (one of which having at least two points), the following inequality stands:

$$\text{Coarse-DTW}(v,w) \geq \phi(s_1,t_1).\delta(v_1,w_1) + \min(s_n,t_m,\phi(s_n,t_m)).\delta(v_n,w_m) \tag{11.11}$$

in which the right-hand side is the $\mathrm{LB}_{\mathrm{Kim}}$ lower bound adapted to Coarse-DTW.

Another lower bound, known as $\mathrm{LB}_{\mathrm{Keogh}}$, has enabled consequent speedup of DTW computation [109]. It is based upon the calculation of an envelope; however this calculation is not trivially transferable to the case of multidimensional time series simply by generalizing the uni-dimensional equations. Thus, we will unfortunately not consider it in our study.

However, a cheap bound can be evaluated several times as DTW progresses as follows: for any row i, the minimum of all cells $A[i,.]$ is a lower bound to the DTW result. Indeed, this result is the last cell of the last row, and the sequence mapping a row i to $\min_j A[i,j]$ is increasing, because the costs are positive. Hence, during each outer loop iteration (i.e., on index i), we can store the minimum of the current row and compare it to the best-so-far for possibly early abandoning. This can be transposed directly to Coarse-DTW without additional modifications.

11.6 Results

11.6.1 DTW vs. Coarse-DTW in 1-NN classification

In this first setup we considered the classification accuracy and speed of various labeled time series datasets. The classifier is 1-NN and we enabled all optimizations described earlier that apply to multidimensional time series, namely: early abandoning on LB_{Kim} and early abandoning on the minima of rows. We report only the classification time, not the learning time.

Dataset MSRAction3D [125] consists of 10 actors executing the same gestures several times, with 60 dimensions (twenty 3D joints). To classify this dataset, we cross-validated all possible combinations of 5 actors in training and 5 in test, thus totaling 252 rounds.

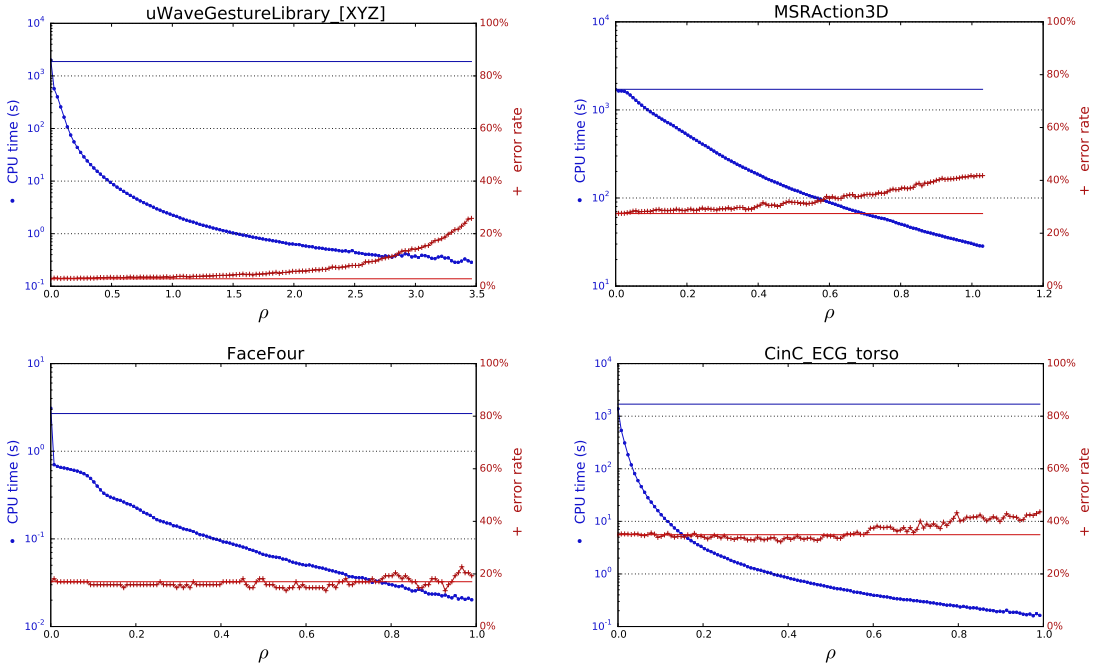


Figure 11.5: 1-NN classification time and error rate of Coarse-DTW as ρ increases. For reference, DTW results are shown as horizontal bars (independent of ρ).

The dataset uWaveGestureLibrary_[XYZ] comes from the UCR time series database [107]. It can be considered as three independent uni-dimensional datasets, but we rather used it here as a single set of 3-dimensional time series. The interest is obvious: in 1-NN DTW classification, we went from individual 1D errors of respectively 27.3 %, 36.6 % and 34.2 %, down to only 2.8 % when the three time series sets are taken together.

Finally, for the sake of comparison, we also ran our tests on the other UCR time series datasets at our disposal. It should be noted that they are all uni-dimensional, however we exclusively considered them as multidimensional time series that *happen* to have a dimension of $d = 1$. This means in particular that some of the traditional lower bounds such as LB_{Keogh} cannot be used, only the multidimensional-enabled ones described earlier.

For each dataset, we ran the classification once with DTW to obtain a reference value both

time- and accuracy-wise. Then, we ran Coarse-DTW, with several values of ρ, as follows: the dense time series are first downsampled with Bubble into sparse time series, according to the current ρ, and then classified with Coarse-DTW. The time and error rate was measured at every run. In Fig. 11.5 we show the full results for a few datasets.

dataset	d	time DTW	time Coarse-DTW	speedup	best ϕ
uWaveGestureLibrary_[XYZ]	3	1850 s	0.769 s	**2413.3x**	ϕ_{stairs}
MSRAction3D	60	1710 s	428 s	**4.0x**	$\phi_{\max}$
Adiac	1	21.0 s	13.0 s	**1.6x**	ϕ_{stairs}
Beef	1	1.35 s	0.014 s	**93.5x**	$\phi_{\max}$
CBF	1	3.18 s	0.0393 s	**80.9x**	ϕ_{diag}
ChlorineConcentration	1	73.2 s	14.0 s	**5.2x**	$\phi_{\max}$
CinC_ECG_torso	1	1690 s	0.413 s	**4100.7x**	$\phi_{\max}$
Coffee	1	0.479 s	0.139 s	**3.5x**	$\phi_{\max}$
DiatomSizeReduction	1	3.81 s	0.241 s	**15.8x**	$\phi_{\max}$
ECG200	1	0.258 s	0.012 s	**20.7x**	$\phi_{\max}$
ECGFiveDays	1	2.34 s	0.283 s	**8.2x**	$\phi_{\max}$
FaceAll	1	73.3 s	21.5 s	**3.4x**	$\phi_{\max}$
FaceFour	1	2.69 s	0.031 s	**85.8x**	ϕ_{stairs}
FacesUCR	1	42.7 s	7.46 s	**5.7x**	$\phi_{\max}$
FISH	1	47.1 s	38.0 s	**1.2x**	$\phi_{\max}$
Gun_Point	1	0.653 s	0.108 s	**6.1x**	ϕ_{diag}
Haptics	1	445 s	0.860 s	**516.7x**	$\phi_{\max}$
InlineSkate	1	1790 s	0.548 s	**3276.1x**	ϕ_{diag}
ItalyPowerDemand	1	0.236 s	0.103 s	**2.3x**	ϕ_{stairs}
Lighting2	1	17.0 s	0.440 s	**38.6x**	ϕ_{stairs}
Lighting7	1	4.66 s	5.21s	**0.9x**	$\phi_{\max}$
MALLAT	1	1460 s	6.408 s	**228.4x**	$\phi_{\max}$
MedicalImages	1	2.92 s	0.261 s	**11.2x**	$\phi_{\max}$
MoteStrain	1	1.14 s	0.0480 s	**23.7x**	$\phi_{\max}$
NonInvasiveFetalECG_Thorax1	1	9820 s	516,s	**19.0x**	$\phi_{\max}$
NonInvasiveFetalECG_Thorax2	1	9720 s	310 s	**31.3x**	$\phi_{\max}$
OliveOil	1	3.15 s	1.43 s	**2.2x**	ϕ_{diag}
OSULeaf	1	59.3 s	2.16 s	**27.4x**	ϕ_{stairs}
SonyAIBORobot_Surface	1	0.465 s	0.245 s	**1.9x**	$\phi_{\max}$
SonyAIBORobot_SurfaceII	1	0.902 s	0.150 s	**6.0x**	ϕ_{stairs}
StarLightCurves	1	44700 s	58.6 s	**763.0x**	$\phi_{\max}$
SwedishLeaf	1	19.0 s	11.5 s	**1.7x**	$\phi_{\max}$
Symbols	1	21.8 s	1.91 s	**11.4x**	$\phi_{\max}$
synthetic_control	1	1.97 s	0.624 s	**3.2x**	$\phi_{\max}$
Trace	1	2.36 s	0.00636 s	**371.3x**	$\phi_{\max}$
TwoLeadECG	1	0.827 s	0.0869 s	**9.5x**	$\phi_{\max}$
Two_Patterns	1	371 s	0.668 s	**556.4x**	$\phi_{\max}$
wafer	1	158 s	0.402 s	**392.1x**	$\phi_{\max}$
WordsSynonyms	1	87.4 s	4.61 s	**18.9x**	$\phi_{\max}$
yoga	1	581 s	4.78 s	**121.7x**	$\phi_{\max}$

Table 11.1: Performance of Coarse-DTW (for a threshold at +2% abs. err.) compared to DTW, in 1-NN classification (datasets from [125] and [107]).

A general trend can be observed (Fig. 11.5): as ρ increases, classification time decreases. However, this comes at the expense of a higher error rate. This is expected: indeed, downsampled time series contain less information than their dense counterparts. Now, we can observe that some time series allow ρ to increase quite a bit (and therefore classification goes much faster) before the accuracy really degrades.

In order to quantify this effect, we proceed as follows. We first set a threshold on the error rate. Here, we select the threshold to be 2% (absolute error) above our reference, the DTW error rate. (For example, if the DTW error rate were 27.1%, we would set the threshold at 29.1%, which might or might not be acceptable depending on the user's constraints.) Then we find the value of

$$\rho^* = \max\{\rho \mid \forall \rho' \leq \rho, \text{err}_{\rho'} \leq \text{err}_{\text{DTW}} + 2\%\} \tag{11.12}$$

which represents the last acceptable value before the error rates first goes above the threshold (the "breakout"). The CPU time associated with the run of ρ^* is likely to be below the DTW CPU time, which is why we define the speedup as their ratio:

$$\text{speedup} = \frac{\text{CPU time}_{\text{ DTW}}}{\text{CPU time}_{\text{ Coarse-DTW at } \rho^*}} \tag{11.13}$$

Furthermore, we tested each of the three possibilities for ϕ. Of all three, we selected only the ρ^* value giving the best time. The values of ρ^* and the speedup are summarized in Table 11.1, along with the winning ϕ.

Additionally, our study aimed to search for the best ϕ function for the diagonal weight. We can conclude from Table 11.1 that the most satisfactory is $\phi_{\max}$, offering the best ratio accuracy/time. Actually, it appears from our experience that ϕ_{diag} was good enough accuracy-wise but was too slow due to the square root. Thus, we recommend selecting $\phi_{\max}$ by default.

11.6.2 Streaming implementation of Coarse-DTW* in our gesture recognition setup

This experiment aims to show the behaviour of Coarse-DTW* and Bubble downsampling with different ρ values in our gesture recognition application. For this experiment, we considered the whole pipeline described so far, including the advanced training procedure described in Chapter 10.

In the same way that we extended DTW* into a streaming algorithm (Chapter 9), here Coarse-DTW* was extended in a stream-enabled version where each new sparse point also creates a single new column. Consistently with equation 11.8, the length normalization takes places on the sum of stays (i.e. the duration) rather than the number of points, and the width tracking (Chap. 9, Fig. 9.5 p. 134) operates on the basis of stays rather than the number of cells. This streaming distance is plugged into the Threshold Recognizer. Following our recommendation in the previous section, we chose $\phi_{\max}$ as the diagonal weight.

Gesture streams each contain 4 instances of each of these nine gestures: go, stop, back, slower, faster, left, right, round, come. Three gesture streams were used for training (and for self-validation to compute the best subset and the thresholds, as our real-time GUI does) and three other streams were given for testing.

We increased the Bubble radius ρ and gathered recognition results with the procedure described in Chapter 9. Each time, the training procedure was fully re-run, therefore for each ρ pass, subsets and thresholds depend on the new distance computations. As we can see in Table 11.2, when the radius increases, fewer time is needed to perform training and recognition. This is consistent with the classification results obtained in Section 11.6.1. Also expectedly, the accuracy metrics are less satisfying as we gain time, but while the precision seems to be well maintained, the recall decreases from 99% to 80.5% for a simple 3x speedup, which is not as convincing as we would like.

On Table 11.3, we have ran the same experiment but we supplied the thresholds ourselves without relying on the automatic threshold selection procedure. The precision and the recall seem more stable. On the one hand, it is a good indicator that Coarse-DTW works

ρ	FN	FP	TP	P/R	CPU time
No downsampling	1	4	107	96.3% / 99.0%	96.7s
0.0001	8	3	100	97.0% / 92.5%	67.4s
0.0002	16	3	92	96.8% / 85.1%	48.6s
0.0003	20	3	88	96.7% / 81.4%	42.9s
0.0004	20	4	88	95.6% / 81.4%	39.4s
0.0005	21	4	87	95.6% / 80.5%	38.3s
0.0006	21	4	87	95.6% / 80.5%	35.3s
0.0007	18	5	90	94.7% / 83.3%	34.1s
0.0008	20	4	88	95.6% / 81.4%	33.2s
0.0009	21	2	87	97.7% / 80.5%	32.5s
0.0010	21	3	87	96.6% / 80.5%	31.6s

Table 11.2: Coarse-DTW provides acceleration when the downsampling radius ρ is increased, since time series are represented with fewer points. Here, the full advanced training procedure with threshold selection was carried out.

ρ	FN	FP	TP	P/R	CPU time
No downsampling	1	3	107	97.2% / 99.0%	151.0s
0.0001	2	1	106	99.0% / 98.1%	79.8s
0.0002	2	0	106	100.0% / 98.1%	55.2s
0.0003	2	0	106	100.0% / 98.1%	46.3s
0.0004	2	0	106	100.0% / 98.1%	41.4s
0.0005	2	1	106	99.0% / 98.1%	38.4s
0.0006	2	2	106	98.1% / 98.1%	35.0s
0.0007	2	2	106	98.1% / 98.1%	35.6s
0.0008	2	2	106	98.1% / 98.1%	30.9s
0.0009	2	3	106	97.2% / 98.1%	32.2s
0.001	2	4	106	96.3% / 98.1%	29.8s

Table 11.3: The same experiment, in which thresholds have been kept identical throughout all ρ passes (hand-crafted values, no automatic threshold computation as in Table 11.2). Precision and Recall are more stable, indicating a possible tension between our automatic threshold procedure and the downsampling algorithm.

well on gesture time series, which is consistent with the results obtained on external labeled time series datasets in Section 11.6.1. On the other hand, it highlights some possible compatibility issues between Coarse-DTW and our procedure for threshold selection. Understanding why would require further investigation, which is a possible direction for future work. Good results from Table 11.3 on manual thresholds suggest we could first use the exact distance computation to find the correct subset and thresholds, and then, downsample these instances and run the operation pipeline on the downsampled time series.

11.7 Conclusions

In this chapter, we transposed DTW into Coarse-DTW, a version accepting sparse time series, and developed Bubble downsampling, an efficient and streamable algorithm to generate such sparse time series from regular ones. By coupling these two mechanisms, we were able to discover that time series can be classified much faster in nearest-neighbor classification;

the desired tradeoff between speed and accuracy can be reached by tuning the parameter ρ in the downsampling algorithm. Some time series are far more subject to downsampling than others, and therefore results can differ depending on which context time series originate.

Experiments on our gesture streams indicated good potential for downsampling (Table 11.3). Next, in order to achieve a good coupling with our threshold selection procedure, it seems that further work is needed since the automatic threshold computation displays some sensitivity to changes in the underlying distance. Finally, while the techniques presented in this chapter offer interesting perspective on the behavior of time series and streams, they are not mandatory to achieve real-time recognition. Indeed, our initial setup without downsampling already offers this experience to the user.

Chapter 12

Conclusion

Contents

12.1 Meeting the goals

It is time to review the work carried out during these three years and thereby presented in this document. We started by choosing the sensors based on the application of choice, a military robot control, and then designed a suitable pipeline for gesture recognition, including motion processing, time series matching, advanced training and stream downsampling.

Let us consider the main constraints highlighted during Chapter 2 to see if our final recognition system fits them. First, the main goal was to achieve robotic control. Unfortunately, during the unfolding of this thesis, the Z-trooper project changed priorities and it was not possible to use it to test our work. Therefore, in order to demonstrate the control of a robot nonetheless, we plugged our recognition pipeline on a smaller prototype robot available in the robotics lab. This one weighs around 5 kgs and possesses 6 wheels, which makes it suitable for moving in and around Thales offices. We used our recognition system to control the robot's motion in the corridors for two hours, which was succesful in the sense that the robot reacted well to motion commands and we were able to make it go wherever we wanted without hitting walls or people.

Our system was built for real time control from the ground up. Therefore it is natural that it works on-line; not only we have been able to drive our robot succesfully validates the real-time control, but we also show it quantitatively throughout this thesis by announcing timing measurements that are always well faster than real-time operation.

Accuracy depends on the training data, but thanks to our robust instance rejection procedure, we obtain high precision/recall scores (both being able to reach more than 95%). Even in mobility, the system is accurate provided the user has trained the system with instances of gestures performed in the same context. Another asset for mobility is the robust motorcycling glove we have chosen as our wearable device.

Regarding stealth considerations, the gesture-based system does not emit any sound, IR or light, even though it will ultimately need a radio medium to be wireless in the future.

Finally, user-friendliness was much addressed throughout this work. First, we designed a gesture dictionary that draws inspiration from other forms of intuitive sign communication. We made the gesture classes simple to perform and easy to understand when it comes to remembering the relationship between a gesture and the associated command. Regarding software aspects, we have designed the advanced training procedure of Chapter 10 exactly with user-friendliness in mind: its main goal is to avoid the user to tweak with threshold values and allow them to occasionally perform an incorrect without it causing recognition problems.

Additional goals are met as well since we allow the user to create a custom dictionary during training. Indeed, the system starts totally untrained and acquires gestures only via user input. By coupling it with our interface-based recognition pipeline, it is able to transmit commands to any controllable system listening to the pipeline's output, making it able to drive other systems, not only a robot.

Industry requirements are fulfilled by the creation of the glove itself, and delivery of a succesful gesture recognition software along with necessary tools to control the robot based on the gesture commands.

In terms of academic novelty, we brought some original work, mainly in the time series domain, thanks to the following ideas: - an advanced training procedure (Chapter 10) including instance rejection and threshold selection, which has so far not been tackled by the state of the art in stream-related contexts; - a downsampling technique (Chapter 11) for use in elastic distance computation between time series and streams.

The loop was finally closed when these new algorithms, whose problems were drawn from our gesture application, were implemented to enhance our recognition pipeline, leading to a practical system for robot control.

12.2 Perspectives

This thesis was the occasion to investigate challenges in many domains, including hardware sensors, human-computer interaction, gesture semantics, and time series recognition. We joined the pieces together and contributed original research in some of these domains. However, some work could be still enhanced.

First, the evaluation procedures could be more consistent by gathering more data. While we could investigate aspects related to mobility, more gesture data in different mobility contexts could provide additional insights in the behavior of our system for real-time outdoors control.

In our work, we came up against a subtle, well-hidden issue that might seem irrelevant, but is actually present every time learning and evaluation are discussed. The problem is

the obtention of ground truth data. Expert annotation are very time-demanding and could be imperfect, which is why this approach undertaken at some point during our work is not ultimately scalable. Our preferred approach so far is to ask the user to perform gestures in specific time intervals. Nonetheless, it is far from perfect because it requires users to be well focused during recording, but they all too often perform a gesture outside of their time boundaries, a mistake that is troublesome to correct after the fact; ideas from Chapter 10 help in removing such problems during training, but it is not clear how we should deal with ground truth data made for evaluation of our work. Last, the most problematic issue is that such an automated annotation system makes our training data somewhat "synthetic", since the gestures are not truly performed in the context of the task in question (robot control, for example), and thus do not well represent the true distribution of the gestures performed at operation time. Investigating semi-automated ways to acquire sensor data, especially during operation, could therefore be an improvement over our current data gathering procedure.

Finally, evaluations in terms of robot control, not only accuracy, could also be designed in the future: a few false detections may actually not matter in the end, if the user is able to quickly reverse a command mistakenly issued. Therefore, one could think of experiments indicating if the robotic task is correctly performed, rather than simply analyzing accuracy metrics of synthetic gesture data.

12.3 Conclusions of this thesis

Finally, I would like to mention how grateful I am to both Thales and IRISA for having been able to study such an exciting topic in excellent conditions. In this document, I hope I have been able to transmit my fascination for deeply technical problems at the borders of human-computer communication and machine intelligence.

Bibliography

[1] Cenk Acar and Andrei Shkel. *MEMS vibratory gyroscopes: structural approaches to improve robustness*. Springer Science & Business Media, 2008.

[2] Thomas Allevard. *Représentation symbolique de la configuration de la main. Application à la reconnaissance de signes et au contrôle d'un robot mobile*. 2005.

[3] AnthroTronix. *AnthroTronix AcceleGlove User Guide*. July 2013. URL: https://web.archive.org/web/20130726103617/http://acceleglove.com/AcceleGloveUserGuide.pdf (visited on 11/17/2016).

[4] AnthroTronix. *AnthroTronix - iGlove/AcceleGlove™*. URL: http://www.anthrotronix.com/index.php?option=com_content&view=article&id=87&Itemid=138 (visited on 04/17/2014).

[5] A. Apostolico, G.M. Landau, and S. Skiena. "Matching for run-length encoded strings". In: *Compression and Complexity of Sequences 1997. Proceedings*. June 1997, pp. 348–356. DOI: 10.1109/SEQUEN.1997.666929.

[6] Tarik Arici et al. "Robust gesture recognition using feature pre-processing and weighted dynamic time warping". In: *Multimedia Tools and Applications* 72.3 (2014), pp. 3045–3062.

[7] Muhammad Asad and Charith Abhayaratne. "Kinect depth stream pre-processing for hand gesture recognition". In: *2013 IEEE International Conference on Image Processing*. IEEE, 2013, pp. 3735–3739.

[8] Mathias Baglioni, Eric Lecolinet, and Yves Guiard. "JerkTilts: using accelerometers for eight-choice selection on mobile devices". In: *Proceedings of the 13th international conference on multimodal interfaces*. ACM, 2011, pp. 121–128. URL: http://dl.acm.org/citation.cfm?id=2070503 (visited on 11/17/2016).

[9] Stephen W. Bailey and Bobby Bodenheimer. "A Comparison of Motion Capture Data Recorded from a Vicon System and a Microsoft Kinect Sensor". In: *Proceedings of the ACM Symposium on Applied Perception*. SAP '12. New York, NY, USA: ACM, 2012, pp. 121–121. ISBN: 978-1-4503-1431-2. DOI: 10.1145/2338676.2338703.

[10] Adeline Bailly et al. "Bag-of-Temporal-SIFT-Words for Time Series Classification". In: *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*. 2015.

[11] J. Baker. "The DRAGON system–An overview". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1 (Feb. 1975), pp. 24–29. ISSN: 0096-3518. DOI: 10.1109/TASSP.1975.1162650.

[12] Pablo Barros et al. "A Multichannel Convolutional Neural Network for Hand Posture Recognition". In: *Artificial Neural Networks and Machine Learning – ICANN 2014*. Ed. by Stefan Wermter et al. Vol. 8681. Cham: Springer International Publishing, 2014, pp. 403–410.

[13] Pablo Barros et al. "Real-time gesture recognition using a humanoid robot with a deep neural architecture". In: IEEE, Nov. 2014, pp. 646–651. ISBN: 978-1-4799-7174-9. DOI: 10.1109/HUMANOIDS.2014.7041431.

[14] Leonard E. Baum and Ted Petrie. "Statistical inference for probabilistic functions of finite state Markov chains". In: *The annals of mathematical statistics* 37.6 (1966), pp. 1554–1563.

[15] B. R. C. Bedregal, G. P. Dimuro, and A. C. R. Costa. "Interval Fuzzy Rule-Based Hand Gesture Recognition". In: *Scientific Computing, Computer Arithmetic and Validated Numerics, 2006. SCAN 2006. 12th GAMM - IMACS International Symposium on*. Sept. 2006, pp. 12–12. DOI: 10.1109/SCAN.2006.25.

[16] Jounghoon Beh, David Han, and Hanseok Ko. "Rule-based trajectory segmentation for modeling hand motion trajectory". In: *Pattern Recognition* 47.4 (2014), pp. 1586–1601.

[17] Jim Bell. "Mars exploration: Roving the red planet". In: *Nature* 490.7418 (2012), pp. 34–35.

[18] Richard E. Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 2015.

[19] Ari Y. Benbasat and Joseph A. Paradiso. "An Inertial Measurement Framework for Gesture Recognition and Applications". In: *Gesture and Sign Language in Human-Computer Interaction*. Ed. by G. Goos et al. Vol. 2298. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 9–20.

[20] Ari Y. Benbasat and Joseph A. Paradiso. "Compact, configurable inertial gesture recognition". In: ACM Press, 2001, p. 183. ISBN: 978-1-58113-340-0. DOI: 10.1145/634067.634178.

[21] M. K. Bhuyan et al. "A novel set of features for continuous hand gesture recognition". In: *Journal on Multimodal User Interfaces* 8.4 (2014), pp. 333–343.

[22] M.J. Bishop and E.A. Thompson. "Maximum likelihood alignment of DNA sequences". In: *Journal of Molecular Biology* 190.2 (July 1986), pp. 159–165. ISSN: 00222836. DOI: 10.1016/0022-2836(86)90289-5.

[23] Amit Bleiweiss et al. "Enhanced interactive gaming by blending full-body tracking and gesture animation". In: ACM, Dec. 2010, p. 34. ISBN: 978-1-4503-0523-5. DOI: 10.1145/1899950.1899984.

[24] Mohamed-Rafik Bouguelia, Yolande Belaïd, and Abdel Belaïd. "Pattern Recognition: Applications and Methods: 4th International Conference, ICPRAM 2015, Lisbon, Portugal, January 10-12, 2015, Revised Selected Papers". In: ed. by Ana Fred, Maria De Marsico, and Mário Figueiredo. 2015. Chap. Identifying and Mitigating Labelling Errors in Active Learning, pp. 35–51. ISBN: 978-3-319-27677-9.

[25] Annelies Braffort. *Reconnaissance et compréhension de gestes, application à la langue des signes*. 1996.

[26] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[27] Bryan Buchholz, Thomas J. Armstrong, and Steven A. Goldstein. "Anthropometric data for describing the kinematics of the human hand". In: *Ergonomics* 35.3 (Mar. 1992), pp. 261–273. ISSN: 0014-0139, 1366-5847. DOI: 10.1080/00140139208967812.

[28] The Duy Bui and Long Thang Nguyen. "Recognizing postures in Vietnamese sign language with MEMS accelerometers". In: *IEEE sensors journal* 7.5 (2007), pp. 707–712.

[29] Xiujuan Chai, Yikai Fang, and Kongqiao Wang. "Robust hand gesture analysis and application in gallery browsing". In: *2009 IEEE International Conference on Multimedia and Expo*. IEEE, 2009, pp. 938–941.

[30] Kaushik Chakrabarti et al. "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases". In: *ACM Trans. Database Syst.* 27.2 (June 2002), pp. 188–228. ISSN: 0362-5915. DOI: 10.1145/568518.568520.

[31] J. Y. C. Chen, E. C. Haas, and M. J. Barnes. "Human Performance Issues and User Interface Design for Teleoperated Robots". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.6 (Nov. 2007), pp. 1231–1245. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2007.905819.

[32] Xiang Chen et al. "Hand Gesture Recognition Research Based on Surface EMG Sensors and 2D-accelerometers". In: *2007 11th IEEE International Symposium on Wearable Computers*. Oct. 2007, pp. 11–14. DOI: 10.1109/ISWC.2007.4373769.

[33] Xiang Chen et al. "Multiple Hand Gesture Recognition Based on Surface EMG Signal". In: IEEE, 2007, pp. 506–509. ISBN: 978-1-4244-1120-7. DOI: 10.1109/ICBBE.2007.133.

[34] Yanping Chen et al. *The UCR Time Series Classification Archive*. www.cs.ucr.edu/~eamonn/time_series_data/. July 2015.

[35] Hong Cheng, Zhongjun Dai, and Zicheng Liu. "Image-to-Class Dynamic Time Warping for 3D hand gesture recognition". In: *2013 IEEE International Conference on Multimedia and Expo (ICME)*. July 2013, pp. 1–6. DOI: 10.1109/ICME.2013.6607524.

[36] Hong Cheng, Jun Luo, and Xuewen Chen. "A windowed Dynamic Time Warping approach for 3D continuous hand gesture recognition". In: *2014 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2014, pp. 1–6.

[37] Hong Cheng, Lu Yang, and Zicheng Liu. "A survey on 3d hand gesture recognition". In: (2015).

[38] Hong Cheng et al. "Real world activity summary for senior home monitoring". In: *Multimedia Tools and Applications* 70.1 (2014), pp. 177–197.

[39] Selina Chu et al. “Iterative Deepening Dynamic Time Warping for Time Series”. In: *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*. Ed. by Robert L. Grossman et al. 2002, pp. 195–212. DOI: 10.1137/1.9781611972726.12. URL: http://dx.doi.org/10.1137/1.9781611972726.12.

[40] A. Corradini and H.-M. Gross. “Camera-based gesture recognition for robot control”. In: IEEE, 2000, 133–138 vol.4. ISBN: 978-0-7695-0619-7. DOI: 10.1109/IJCNN.2000.860762.

[41] Eva Coupeté, Sotiris Manitsaris, and Fabien Moutarde. “Real-time recognition of human gestures for collaborative robots on assembly-line”. In: *3rd International Digital Human Modeling Symposium (DHM2014)*. 2014, 7–p.

[42] M. Cuturi et al. “A Kernel for Time Series Based on Global Alignments”. In: *IEEE ICASSP 2007*. Vol. 2. Apr. 2007, DOI: 10.1109/ICASSP.2007.366260.

[43] *CyberGlove II*. URL: http://www.cyberglovesystems.com/cyberglove-ii/ (visited on 11/17/2016).

[44] Ireneusz Czarnowski. “Transactions on Computational Collective Intelligence IV”. In: ed. by Ngoc Thanh Nguyen. 2011. Chap. Distributed Learning with Data Reduction, pp. 3–121. ISBN: 978-3-642-21883-5.

[45] Thomas Defanti and Daniel Sandin. “Final report to the National Endowment of the arts”. In: *US NEA R60-34-163, University of Illinois at Chicago Circle, Chicago, Illinois* (1977).

[46] Joaquin Derrac, Salvador Garcia, and Francisco Herrera. “A Survey on Evolutionary Instance Selection and Generation.” In: *Int. J. of Applied Metaheuristic Computing* 1.1 (2010), pp. 60–92.

[47] DGTech Engineering Solutions. *VHand 3.0 Datasheet*. Oct. 2013. URL: http://www.dgtech.it/vhand3/files/DG5VHAND3_Datasheet.pdf (visited on 11/18/2016).

[48] N. Diolaiti and C. Melchiorri. “Teleoperation of a mobile robot through haptic feedback”. In: *Haptic Virtual Environments and Their Applications, IEEE International Workshop 2002 HAVE*. 2002, pp. 67–72. DOI: 10.1109/HAVE.2002.1106916.

[49] L. Dipietro, A.M. Sabatini, and P. Dario. “A Survey of Glove-Based Systems and Their Applications”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 38.4 (July 2008), pp. 461–482. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2008.923862.

[50] Laura Dipietro, Angelo M Sabatini, and Paolo Dario. “Evaluation of an instrumented glove for hand-movement acquisition”. In: *Journal of rehabilitation research and development* 40.2 (Apr. 2003), pp. 179–189. ISSN: 0748-7711.

[51] Paul Doliotis et al. “Comparing gesture recognition accuracy using color and depth information”. In: *Proceedings of the 4th international conference on PErvasive technologies related to assistive environments*. ACM, 2011, p. 20.

[52] Anca D. Dragan and Siddhartha S. Srinivasa. *Formalizing assistive teleoperation*. MIT Press, July, 2012.

[53] Marc Dupont and Pierre-François Marteau. “Coarse-DTW: Exploiting Sparsity in Gesture Time Series”. In: *Advanced Analytics and Learning on Temporal Data (AALTD)*. 2015.

[54] Marc Dupont and Pierre-François Marteau. "Coarse-DTW for Sparse Time Series Alignment". In: *International Workshop on Advanced Analytics and Learning on Temporal Data*. Springer, 2015, pp. 157–172.

[55] Marc Dupont, Pierre-François Marteau, and Nehla Ghouaiel. "Detecting low-quality reference time series in stream recognition". In: *23rd International Conference on Pattern Recognition (ICPR)*. 2016.

[56] Richard Durbin, ed. *Biological sequence analysis: probabalistic models of proteins and nucleic acids*. Cambridge, UK : New York: Cambridge University Press, 1998.

[57] Mahmoud Elmezain et al. "A hidden markov model-based continuous gesture recognition system for hand motion trajectory". In: *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. IEEE, 2008, pp. 1–4.

[58] R. Erenshteyn et al. "Recognition approach to gesture language understanding". In: IEEE, 1996, 431–435 vol.3. ISBN: 978-0-8186-7282-8. DOI: 10.1109/ICPR.1996.546984.

[59] David Feil-Seifer and Maja J. Mataric. "Defining socially assistive robotics". In: *9th International Conference on Rehabilitation Robotics, 2005. ICORR 2005.* IEEE, 2005, pp. 465–468.

[60] Fifth Dimension Technologies. *Data Gloves | 5DT*. URL: http://www.5dt.com/data-gloves/ (visited on 11/17/2016).

[61] Pascual J. Figueroa, Neucimar J. Leite, and Ricardo M. L. Barros. "A flexible software for tracking of markers used in human motion analysis". In: *Computer Methods and Programs in Biomedicine* 72.2 (Oct. 2003), pp. 155–165. ISSN: 0169-2607. DOI: 10.1016/S0169-2607(02)00122-0.

[62] David Fischinger et al. "Hobbit, a care robot supporting independent living at home: First prototype and lessons learned". In: *Robotics and Autonomous Systems* 75 (2016), pp. 60–78.

[63] Terrence W. Fong et al. "Novel interfaces for remote driving: gesture, haptic, and PDA". In: *Intelligent Systems and Smart Manufacturing*. International Society for Optics and Photonics, 2001, pp. 300–311.

[64] Rita Francese, Ignazio Passero, and Genoveffa Tortora. "Wiimote and Kinect: Gestural User Interfaces Add a Natural Third Dimension to HCI". In: *Proceedings of the International Working Conference on Advanced Visual Interfaces*. AVI '12. New York, NY, USA: ACM, 2012, pp. 116–123. ISBN: 978-1-4503-1287-5. DOI: 10.1145/2254556.2254580.

[65] Yifan Fu, Xingquan Zhu, and Bin Li. "A survey on instance selection for active learning". In: *Knowledge and Information Systems* 35.2 (2012), pp. 249–283.

[66] Eric Fujiwara et al. "Development of a Glove-based Optical Fiber Sensor for Applications in Human-robot Interaction". In: *Proceedings of the 8th ACM/IEEE International Conference on Human-robot Interaction*. HRI '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 123–124. ISBN: 978-1-4673-3055-8.

[67] J. Funda et al. "Constrained Cartesian motion control for teleoperated surgical robots". In: *IEEE Transactions on Robotics and Automation* 12.3 (June 1996), pp. 453–465. ISSN: 1042-296X. DOI: 10.1109/70.499826.

[68] Shinichi Furuya, Martha Flanders, and John F. Soechting. "Hand kinematics of piano playing". In: *Journal of neurophysiology* 106.6 (2011), pp. 2849–2864.

[69] Luigi Gallo. "Hand shape classification using depth data for unconstrained 3D interaction". In: *Journal of Ambient Intelligence and Smart Environments* 6.1 (2014), pp. 93–105.

[70] D. M Gavrila. "The Visual Analysis of Human Movement: A Survey". In: *Computer Vision and Image Understanding* 73.1 (Jan. 1999), pp. 82–98. ISSN: 1077-3142. DOI: 10.1006/cviu.1998.0716.

[71] Reinhard Gentner and Joseph Classen. "Development and evaluation of a low-cost sensor glove for assessment of human finger movements in neurophysiological settings". In: *Journal of Neuroscience Methods* 178.1 (Mar. 2009), pp. 138–147. ISSN: 0165-0270. DOI: 10.1016/j.jneumeth.2008.11.005.

[72] Nicholas Edward Gillian. "Gesture recognition for musician computer interaction". PhD thesis. Queen's University, 2011.

[73] Michael A. Goodrich and Alan C. Schultz. "Human-robot interaction: a survey". In: *Foundations and trends in human-computer interaction* 1.3 (2007), pp. 203–275.

[74] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[75] Alex Graves et al. "A novel connectionist system for unconstrained handwriting recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 31.5 (2009), pp. 855–868.

[76] Alex Graves et al. "Unconstrained on-line handwriting recognition with recurrent neural networks". In: *Advances in Neural Information Processing Systems*. 2008, pp. 577–584.

[77] Guaskenn. *Langue des Signes*. 2008. URL: http://guaskenn.free.fr/LSF/LSF.html (visited on 11/23/2016).

[78] Isabelle Guyon et al. "ChaLearn Gesture Challenge: Design and First Results". In: 2 (2012), pp. 100–103.

[79] Yaniv Hamo and Shaul Markovitch. "The Compset Algorithm for Subset Selection". In: *Proceedings of The Nineteenth International Joint Conference for Artificial Intelligence*. Edinburgh, Scotland, 2005, pp. 728–733.

[80] Bastian Hartmann and Norbert Link. "Gesture recognition with inertial sensors and optimized DTW prototypes". In: IEEE, Oct. 2010, pp. 2102–2109. ISBN: 978-1-4244-6586-6. DOI: 10.1109/ICSMC.2010.5641703.

[81] Jose L. Hernandez-Rebollar, Nicholas Kyriakopoulos, and Robert W. Lindeman. "The AcceleGlove: A Whole-hand Input Device for Virtual Reality". In: *ACM SIGGRAPH 2002 Conference Abstracts and Applications*. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 259–259. ISBN: 978-1-58113-525-1. DOI: 10.1145/1242073.1242272.

[82] Antonio Hernández-Vela et al. "BoVDW: Bag-of-Visual-and-Depth-Words for gesture recognition". In: *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, 2012, pp. 449–452.

[83] David Hoag. *Consideration of Apollo IMU Gimbal Lock*. MIT Instrumentation Laboratory Document E-1344. Apr. 1963.

[84] Viet Tran Hoang, Anh Nguyen Hoang, and Dongho Kim. "Real-time stereo rendering technique for virtual reality system based on the interactions with human view and hand gestures". In: *International Conference on Virtual, Augmented and Mixed Reality*. Springer, 2013, pp. 103–110.

[85] Jesse Hoey et al. "Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process". In: *Computer Vision and Image Understanding* 114.5 (2010), pp. 503–519.

[86] Anne Hollister et al. "The axes of rotation of the thumb carpometacarpal joint". In: *Journal of Orthopaedic Research* 10.3 (1992), pp. 454–460.

[87] Chao Hu et al. "Visual gesture recognition for human-machine interface of robot teleoperation". In: *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings*. Vol. 2. Oct. 2003, 1560–1565 vol.2. DOI: 10.1109/IROS.2003.1248866.

[88] Yu Huang et al. "A Concept Grounding Approach for Glove-Based Gesture Recognition". In: *2011 7th International Conference on Intelligent Environments (IE)*. July 2011, pp. 358–361. DOI: 10.1109/IE.2011.51.

[89] M C Hume et al. "Functional range of motion of the joints of the hand". In: *The Journal of hand surgery* 15.2 (Mar. 1990), pp. 240–243. ISSN: 0363-5023.

[90] Dan Hwang et al. *Crane Gesture*. US Patent App. 14/098,952. Google Patents, Dec. 2013.

[91] Images SI, Inc. *Flexible Stretch Sensor*. URL: http://www.imagesco.com/sensors/stretch-sensor.html (visited on 04/17/2014).

[92] InvenSense. *MPU-6050 | InvenSense*.

[93] Iron Will Innovations Canada Inc. *Peregrine Tech Specs*. URL: http://theperegrine.com/download/peregrine-tech-specs.pdf (visited on 11/17/2016).

[94] Iron Will Innovations Canada Inc. *The Peregrine*. 2013. URL: http://theperegrine.com/ (visited on 11/16/2016).

[95] ISO. *ISO 16715:2014: Cranes - Hand signals used with cranes*. 2014.

[96] F. Itakura. "Minimum prediction residual principle applied to speech recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1 (Feb. 1975), pp. 67–72. ISSN: 0096-3518. DOI: 10.1109/TASSP.1975.1162641.

[97] F. Itakura. "Minimum prediction residual principle applied to speech recognition". In: *Acoustics, Speech and Signal Processing, IEEE Transactions on* 23.1 (Feb. 1975), pp. 67–72. ISSN: 0096-3518. DOI: 10.1109/TASSP.1975.1162641.

[98] Yannick Jacob et al. "Hand gesture recognition for driver vehicle interaction". In: *IEEE Computer Society Workshop on Observing and understanding hands in action (Hands 2015) of 28th IEEE conf. on Computer Vision and Pattern Recognition (CVPR'2015)*. 2015.

[99] F. Jelinek, L. Bahl, and R. Mercer. "Design of a linguistic statistical decoder for the recognition of continuous speech". In: *IEEE Transactions on Information Theory* 21.3 (May 1975), pp. 250–256. ISSN: 0018-9448. DOI: 10.1109/TIT.1975.1055384.

[100] Shuiwang Ji et al. "3D Convolutional Neural Networks for Human Action Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1 (Jan. 2013), pp. 221–231. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2012.59.

[101] Antonio R Jimenez et al. "A comparison of pedestrian dead-reckoning algorithms using a low-cost MEMS IMU". In: *Intelligent Signal Processing, 2009. WISP 2009. IEEE International Symposium on*. IEEE. 2009, pp. 37–42.

[102] Peter W. Johnson, Per Jonsson, and Mats Hagberg. "Comparison of measurement accuracy between two wrist goniometer systems during pronation and supination". In: *Journal of Electromyography and Kinesiology* 12.5 (2002), pp. 413–420.

[103] Mohamed Bécha Kaâniche. *Human Gesture Recognition*. 2009.

[104] Jean-François Kamp and Gildas Ménier. "Interprétation temps réel de gestes pour la production de schémas rythmiques". In: *RFIA2010-17ème congrès francophone AFRIF-AFIA*. Caen, France, Jan. 2010, p. 891.

[105] Jean-François Kamp, Gildas Ménier, and Sylvie Gibet. "Une interface gestuelle pour l'apprentissage de la rythmique". In: *RFIA 2012 (Reconnaissance des Formes et Intelligence Artificielle)*. Session "Atelier IHMA". Lyon, France, Jan. 2012, pp. 978–2–9539515–2–3.

[106] Andrej Karpathy and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3128–3137.

[107] E. J. Keogh et al. *The UCR Time Series Classification-Clustering Datasets*. http://wwwcs.ucr.edu/ eamonn/time_series_data/. 2006.

[108] Eamonn J. Keogh and Michael J. Pazzani. "Scaling Up Dynamic Time Warping for Datamining Applications". In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '00. Boston, Massachusetts, USA, 2000, pp. 285–289. ISBN: 1-58113-233-6. DOI: 10.1145/347090.347153. URL: http://doi.acm.org/10.1145/347090.347153.

[109] Eamonn Keogh and Chotirat Ann Ratanamahatana. "Exact Indexing of Dynamic Time Warping". In: *Knowl. Inf. Syst.* 7.3 (Mar. 2005), pp. 358–386. ISSN: 0219-1377. DOI: 10.1007/s10115-004-0154-9. URL: http://dx.doi.org/10.1007/s10115-004-0154-9.

[110] Eamonn Keogh and Chotirat Ann Ratanamahatana. "Exact indexing of dynamic time warping". In: *Knowledge and information systems* 7.3 (2005), pp. 358–386.

[111] Eamonn Keogh et al. "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases". In: *JOURNAL OF KNOWLEDGE AND INFORMATION SYSTEMS* 3 (2000), pp. 263–286.

[112] Cem Keskin, Ali Taylan Cemgil, and Lale Akarun. "DTW based clustering to improve hand gesture recognition". In: *International Workshop on Human Behavior Understanding*. Springer, 2011, pp. 72–81.

[113] David Kim et al. "Digits: freehand 3D interactions anywhere using a wrist-worn gloveless sensor". In: *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 2012, pp. 167–176.

[114] Sang-wook Kim, Sanghyun Park, and Wesley W. Chu. "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases". In: *In ICDE*. 2001, pp. 607–614.

[115] Sotaro Kita and James Essegbey. "Pointing left in Ghana: How a taboo on the use of the left hand influences gestural practice". In: *Gesture* 1.1 (2001), pp. 73–95.

[116] Pushmeet Kohli and Jamie Shotton. "Key developments in human pose estimation for kinect". In: *Consumer Depth Cameras for Computer Vision*. Springer, 2013, pp. 63–70.

[117] Cheng-Peng Kuan and Kuu-Young Young. "Challenges in VR-based robot teleoperation". In: *IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA '03*. Vol. 3. Sept. 2003, 4392–4397 vol.3. DOI: 10.1109/ROBOT.2003.1242280.

[118] Alexey Kurakin, Zhengyou Zhang, and Zicheng Liu. "A real time system for dynamic hand gesture recognition with a depth sensor". In: *Proceedings of the 20th European Signal Processing Conference (EUSIPCO), 2012*. IEEE, 2012, pp. 1975–1979.

[119] Leap Motion, Inc. *Leap Motion | Mac & PC Motion Controller for Games, Design, Virtual Reality & More*.

[120] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[121] Chan-Su Lee, SungYong Chun, and Shin Won Park. "Tracking hand rotation and various grasping gestures from an IR camera using extended cylindrical manifold embedding". In: *Computer Vision and Image Understanding* 117.12 (2013), pp. 1711–1723.

[122] Grégoire Lefebvre et al. "BLSTM-RNN Based 3D Gesture Classification". In: *Artificial Neural Networks and Machine Learning – ICANN 2013*. Ed. by David Hutchison et al. Vol. 8131. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 381–388.

[123] Daniel Lemire. "Faster retrieval with a two-pass dynamic-time-warping lower bound". In: *Pattern Recognition* 42.9 (2009), pp. 2169–2180. ISSN: 0031-3203. DOI: http://dx.doi.org/10.1016/j.patcog.2008.11.030.

[124] Hong-Nan Li, Dong-Sheng Li, and Gang-Bing Song. "Recent applications of fiber optic sensors to health monitoring in civil engineering". In: *Engineering Structures* 26.11 (Sept. 2004), pp. 1647–1657. ISSN: 0141-0296. DOI: 10.1016/j.engstruct.2004.05.018.

[125] W. Li, Z. Zhang, and Z. Liu. "Action Recognition Based on A Bag of 3D Points". In: *Proc. IEEE Int'l Workshop on CVPR for Hum. Comm. Behav. Analysis*. Ed. by IEEE CS Press. 2010, pp. 9–14.

[126] Yun Li et al. "Automatic Recognition of Sign Language Subwords Based on Portable Accelerometer and EMG Sensors". In: *International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*. ICMI-MLMI '10. New York, NY, USA: ACM, 2010, 17:1–17:7. ISBN: 978-1-4503-0414-6. DOI: 10.1145/1891903.1891926.

[127] Jeroen F. Lichtenauer, Emile A. Hendriks, and Marcel JT Reinders. "Sign language recognition by combining statistical DTW and independent classification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.11 (2008), pp. 2040–2046.

[128] Hsien-I Lin, Ming-Hsiang Hsu, and Wei-Kai Chen. "Human hand gesture recognition using a convolution neural network". In: IEEE, Aug. 2014, pp. 1038–1043. DOI: 10.1109/CoASE.2014.6899454.

[129] Jie Liu et al. "Training Conditional Random Fields Using Transfer Learning for Gesture Recognition". In: IEEE, Dec. 2010, pp. 314–323. ISBN: 978-1-4244-9131-5. DOI: 10.1109/ICDM.2010.31.

[130] Liwei Liu et al. "Hand posture recognition using finger geometric feature". In: *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, 2012, pp. 565–568.

[131] Yongjing Liu et al. "Image Processing and Recognition of Multiple Static Hand Gestures for Human-Computer Interaction". In: *Image and Graphics (ICIG), 2013 Seventh International Conference on*. IEEE, 2013, pp. 465–470.

[132] S. Madgwick. "An efficient orientation filter for inertial and inertial/magnetic sensor arrays". In: *Report x-io and University of Bristol (UK)* (2010).

[133] Nadia Magnenat-Thalmann, Jian J. Zhang, and David Dagan Feng. *Recent Advances in the 3D Physiological Human*. Springer, 2009.

[134] Sebastian Maier and Patrick van der Smagt. "Surface EMG suffices to classify the motion of each finger independently". In: *Proceedings of MOVIC*. 2008.

[135] João Luis Marins et al. "An extended Kalman filter for quaternion-based orientation estimation using MARG sensors". In: *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*. Vol. 4. IEEE. 2001, pp. 2003–2011.

[136] Pierre-François Marteau and Sylvie Gibet. "On Recursive Edit Distance Kernels with Application to Time Series Classification". In: *IEEE Trans. on Neural Networks and Learning Systems* (June 2014), pp. 1–14. DOI: 10.1109/TNNLS.2014.2333876. URL: http://hal.inria.fr/hal-00486916.

[137] Pierre-François Marteau, Sylvie Gibet, and Clément Reverdy. "Down-sampling Coupled to Elastic Kernel Machines for Efficient Recognition of Isolated Gestures". In: *Pattern Recognition (ICPR), 2014 22nd International Conference on*. Aug. 2014, pp. 363–368. DOI: 10.1109/ICPR.2014.71.

[138] André Maurer, Micha Hersch, and Aude G. Billard. "Extended Hopfield Network for Sequence Learning: Application to Gesture Recognition". In: *Artificial Neural Networks: Biological Inspirations – ICANN 2005*. Ed. by David Hutchison et al. Vol. 3696. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 493–498.

[139] Jérôme Maye and Oliver Brand. "Human Activity Analysis Using IMU/Microphone Sensors". In: (2012).

[140] W. W. Mayol and D. W. Murray. "Wearable hand activity recognition for event summarization". In: *International symposium on wearable computers*. Vol. 1. 2005.

[141] David McNeill. *Hand and mind: What gestures reveal about thought*. University of Chicago press, 1992.

[142] Microsoft Research. *Digits*. Dec. 2012. URL: https://www.microsoft.com/en-us/research/project/digits/ (visited on 11/19/2016).

[143] *MINDFLUX - Essential Reality - P5 Glove*. URL: http://mindflux.com.au/products/essentialreality/p5glove.html (visited on 11/17/2016).

[144] Ministère de la Défense, Etat-Major de l'Armée de Terre, and COFAT. *TTA 150 - Titre IV: Le Combat*. 2001.

[145] Ministère de la Défense, Etat-Major de l'Armée de Terre, and COFAT. *TTA 150 - Titre IV: Le Combat Proterre en Milieu Ouvert*. 2008.

[146] Sushmita Mitra and Tinku Acharya. "Gesture recognition: A survey". In: *IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS - PART C* 37.3 (2007), pp. 311–324.

[147] Arash Mokhber, Catherine Achard, and Maurice Milgram. "Recognition of human behavior by space-time silhouette characterization". In: *Pattern Recognition Letters* 29.1 (2008), pp. 81–89.

[148] Pavlo Molchanov et al. "Hand gesture recognition with 3D convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2015, pp. 1–7.

[149] S. Muller, Stefan Eickeler, and Gerhard Rigoll. "Crane gesture recognition using pseudo 3-d hidden markov models". In: *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*. IEEE, 2000, pp. 398–402.

[150] "Multi-feature criteria with fuzzy logic pattern recognition for hand gesture classification". In: *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*. Aug. 2015, pp. 1–4. DOI: 10.1109/MWSCAS.2015.7282130.

[151] Kazuya Murao and Tsutomu Terada. "A motion recognition method by constancy-decision". In: *International Symposium on Wearable Computers (ISWC) 2010*. IEEE, 2010, pp. 1–4.

[152] Jawad Nagi et al. "Max-pooling convolutional neural networks for vision-based hand gesture recognition". In: IEEE, Nov. 2011, pp. 342–347. DOI: 10.1109/ICSIPA.2011.6144164.

[153] Ghazi Al-Naymat, Sanjay Chawla, and Javid Taheri. "SparseDTW: A Novel Approach to Speed Up Dynamic Time Warping". In: *Proceedings of the Eighth Australasian Data Mining Conference - Volume 101*. AusDM '09. Melbourne, Australia, 2009, pp. 117–127. ISBN: 978-1-920682-82-8.

[154] Natalia Neverova et al. "A multi-scale approach to gesture detection and recognition". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2013, pp. 484–491.

[155] Natalia Neverova et al. "Multi-scale Deep Learning for Gesture Detection and Localization". In: *Computer Vision - ECCV 2014 Workshops*. Ed. by Lourdes Agapito, Michael M. Bronstein, and Carsten Rother. Vol. 8925. Cham: Springer International Publishing, 2015, pp. 474–490.

[156] Juan Carlos Niebles, Hongcheng Wang, and Li Fei-Fei. "Unsupervised learning of human action categories using spatial-temporal words". In: *International journal of computer vision* 79.3 (2008), pp. 299–318.

[157] *Norme FD E52-401: Gestes de commandement des appareils de levage - Gestes recommandés pour les manutentions courantes dans les usines et chantiers*. Sept. 2016.

[158] Kenton O'Hara et al. "Touchless interaction in surgery". In: *Communications of the ACM* 57.1 (2014), pp. 70–77.

[159] Eshed Ohn-Bar and Mohan Trivedi. "The power is in your hands: 3D analysis of hand gestures in naturalistic video". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2013, pp. 912–917.

[160] J. Arturo Olvera-López et al. "A Review of Instance Selection Methods". In: *Artif. Intell. Rev.* 34.2 (Aug. 2010), pp. 133–143. ISSN: 0269-2821.

[161] G. Orengo et al. "Characterization of piezoresistive sensors for goniometric glove in hand prostheses". In: *1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009. Wireless VITAE 2009*. 2009, pp. 684–687. DOI: 10.1109/WIRELESSVITAE.2009.5172530.

[162] Hocine Ouhaddi and Patrick Horain. "Conception et ajustement d'un modèle 3D articulé de la main". In: *Actes des 6èmes Journées de Travail du GT Réalité Virtuelle* (1998), pp. 83–90.

[163] Andrew K. Palmer and Frederick W. Werner. "Biomechanics of the distal radioulnar joint." In: *Clinical orthopaedics and related research* 187 (1984), pp. 26–35.

[164] Farid Parvini et al. "An Approach to Glove-Based Gesture Recognition". In: *Proceedings of the 13th International Conference on Human-Computer Interaction. Part II: Novel Interaction Methods and Techniques*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 236–245. ISBN: 978-3-642-02576-1. DOI: 10.1007/978-3-642-02577-8_26.

[165] Pranav Patel et al. "Mining Motifs in Massive Time Series Databases". In: *In Proceedings of IEEE International Conference on Data Mining (ICDM'02*. 2002, pp. 370–377.

[166] Vladimir I. Pavlovic, Rajeev Sharma, and Thomas S. Huang. "Visual interpretation of hand gestures for human-computer interaction: A review". In: *IEEE Transactions on pattern analysis and machine intelligence* 19.7 (1997), pp. 677–695.

[167] Fabrizio Pedersoli et al. "XKin: an open source framework for hand pose and gesture recognition using kinect". In: *The Visual Computer* 30.10 (2014), pp. 1107–1122.

[168] Krzysztof Pietrusewicz and Karol Miadlicki. *Gestures can control cranes*. Feb. 2014.

[169] Lionel Pigou et al. "Sign Language Recognition Using Convolutional Neural Networks". In: *Computer Vision - ECCV 2014 Workshops*. Ed. by Lourdes Agapito, Michael M. Bronstein, and Carsten Rother. Vol. 8925. Cham: Springer International Publishing, 2015, pp. 572–578.

[170] Ednaldo B. Pizzolato, Mauro dos Santos Anjo, and Guilherme C. Pedroso. "Automatic Recognition of Finger Spelling for LIBRAS Based on a Two-layer Architecture". In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. SAC '10. New York, NY, USA: ACM, 2010, pp. 969–973. ISBN: 978-1-60558-639-7. DOI: 10.1145/1774088.1774290.

[171] Andres Jessé Porfirio et al. "LIBRAS Sign Language Hand Configuration Recognition Based on 3D Meshes". In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2013, pp. 1588–1593.

[172] Leigh Ellen Potter, Jake Araullo, and Lewis Carter. "The leap motion controller: a view on sign language". In: *Proceedings of the 25th Australian computer-human interaction conference: augmentation, application, innovation, collaboration*. ACM, 2013, pp. 175–178.

[173] William Premerlani and Paul Bizard. "Direction Cosine Matrix IMU: Theory". In: *DIY DRONE: USA* (2009), pp. 13–15.

[174] Nicolas Pugeault and Richard Bowden. "Spelling it out: Real-time ASL fingerspelling recognition". In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1114–1119.

[175] L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (Feb. 1989), pp. 257–286. ISSN: 0018-9219. DOI: 10.1109/5.18626.

[176] Giuseppe Raffa et al. "Don't slow me down: Bringing energy efficiency to continuous gesture recognition". In: *International Symposium on Wearable Computers (ISWC) 2010*. IEEE, 2010, pp. 1–8.

[177] Aditya Ramamoorthy et al. "Recognition of dynamic hand gestures". In: *Pattern Recognition* 36.9 (2003), pp. 2069–2081.

[178] Aditya Ramamoorthy et al. "Recognition of dynamic hand gestures". In: *Pattern Recognition* 36.9 (2003), pp. 2069–2081.

[179] Siddharth S. Rautaray and Anupam Agrawal. "Vision based hand gesture recognition for human computer interaction: a survey". In: *Artificial Intelligence Review* 43.1 (2015), pp. 1–54.

[180] Holger Regenbrecht, Jonny Collins, and Simon Hoermann. "A Leap-supported, hybrid AR interface approach". In: *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*. ACM, 2013, pp. 281–284.

[181] Miguel Reyes, Gabriel Domínguez, and Sergio Escalera. "Featureweighting in dynamic timewarping for gesture recognition in depth data". In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1182–1188.

[182] Marco Roccetti, Gustavo Marfia, and Angelo Semeraro. "Playing into the wild: A gesture-based interface for gaming in public spaces". In: *Journal of Visual Communication and Image Representation* 23.3 (2012), pp. 426–440.

[183] Simon Ruffieux, Denis Lalanne, and Elena Mugellini. "ChAirGest: a challenge for multimodal mid-air gesture recognition for close HCI". In: ACM Press, 2013, pp. 483–488. ISBN: 978-1-4503-2129-7. DOI: 10.1145/2522848.2532590.

[184] Giovanni Saggio. "A novel array of flex sensors for a goniometric glove". In: *Sensors and Actuators A: Physical* 205 (Jan. 2014), pp. 119–125. ISSN: 0924-4247. DOI: 10.1016/j.sna.2013.10.030.

[185] Giovanni Saggio. "Mechanical model of flex sensors used to sense finger movements". In: *Sensors and Actuators A: Physical* 185 (Oct. 2012), pp. 53–58. ISSN: 0924-4247. DOI: 10.1016/j.sna.2012.07.023.

[186] Giovanni Saggio et al. "A novel application method for wearable bend sensors". In: *Applied Sciences in Biomedical and Communication Technologies, 2009. ISABEL 2009. 2nd International Symposium on*. IEEE, 2009, pp. 1–3.

[187] S. Saha et al. “A novel gesture recognition system based on fuzzy logic for healthcare applications”. In: *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. July 2016, pp. 634–641. DOI: 10.1109/FUZZ-IEEE.2016.7737746.

[188] H. Sakoe and S. Chiba. “A dynamic programming approach to continuous speech recognition”. In: *Proceedings of the 7th International Congress of Acoustic*. 1971, pp. 65–68.

[189] Hiroaki Sakoe and Seibi Chiba. “A dynamic programming approach to continuous speech recognition”. In: *Proceedings of the seventh international congress on acoustics*. Vol. 3. Budapest, Hungary, 1971, pp. 65–69.

[190] Yasushi Sakurai, Christos Faloutsos, and Masashi Yamamuro. “Stream Monitoring under the Time Warping Distance”. In: IEEE, 2007, pp. 1046–1055. ISBN: 978-1-4244-0802-3. DOI: 10.1109/ICDE.2007.368963.

[191] Yasushi Sakurai, Masatoshi Yoshikawa, and Christos Faloutsos. “FTW: Fast Similarity Search Under the Time Warping Distance”. In: *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS ’05. Baltimore, Maryland, 2005, pp. 326–337. ISBN: 1-59593-062-0. DOI: 10.1145/1065167.1065210. URL: http://doi.acm.org/10.1145/1065167.1065210.

[192] Stan Salvador and Philip Chan. “Toward Accurate Dynamic Time Warping in Linear Time and Space”. In: *Intell. Data Anal.* 11.5 (Oct. 2007), pp. 561–580. ISSN: 1088-467X. URL: http://dl.acm.org/citation.cfm?id=1367985.1367993.

[193] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural Networks* 61 (2015), pp. 85–117.

[194] Scuba Diving Fan Club. *Most Common Diving Signals*. 2016. URL: http://www.scubadivingfanclub.com/Diving_Signals.html (visited on 11/23/2016).

[195] Sensor Products Inc. *Tactilus | Compression Force Sensing Resistor (fsr) | Force Sensing Resistors | Force Sensing Resistors | Tactilus | Surface Pressure Indicator | Mapping | Force Sensing And Profiling*.

[196] T. B. Sheridan. “Space teleoperation through time delay: review and prognosis”. In: *IEEE Transactions on Robotics and Automation* 9.5 (Oct. 1993), pp. 592–606. ISSN: 1042-296X. DOI: 10.1109/70.258052.

[197] Yutao Shou, Nikos Mamoulis, and David W. Cheung. “Fast and Exact Warping of Time Series Using Adaptive Segmental Approximations”. In: *Mach. Learn.* 58.2-3 (Feb. 2005), pp. 231–267. ISSN: 0885-6125. DOI: 10.1007/s10994-005-5828-3.

[198] N. E. Sian et al. “Whole body teleoperation of a humanoid robot - development of a simple master device using joysticks”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*. Vol. 3. 2002, 2569–2574 vol.3. DOI: 10.1109/IRDS.2002.1041657.

[199] Lisa K. Simone and Derek G. Kamper. “Design considerations for a wearable monitor to measure finger posture”. In: *Journal of NeuroEngineering and Rehabilitation* 2.1 (Mar. 2005), p. 5. ISSN: 1743-0003. DOI: 10.1186/1743-0003-2-5.

[200] Lisa K. Simone et al. “A low cost instrumented glove for extended monitoring and functional hand assessment”. In: *Journal of Neuroscience Methods* 160.2 (Mar. 2007), pp. 335–348. ISSN: 0165-0270. DOI: 10.1016/j.jneumeth.2006.09.021.

[201] L.K. Simone et al. "Measuring Finger Flexion and Activity Trends over a 25 Hour Period using a Low Cost Wireless Device". In: *28th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2006. EMBS '06*. 2006, pp. 6281–6284. DOI: 10.1109/IEMBS.2006.260546.

[202] Salvatore Sorce, Vito Gentile, and Antonio Gentile. "Real-Time Hand Pose Recognition Based on a Neural Network Using Microsoft Kinect". In: *Broadband and Wireless Computing, Communication and Applications (BWCCA), 2013 Eighth International Conference on*. IEEE, 2013, pp. 344–350.

[203] Anthony Sorel. "Gestion de la variabilité morphologique pour la reconnaissance de gestes naturels à partir de données 3D". PhD thesis. Université Rennes 2, Dec. 2012.

[204] SparkFun Electronics. *DIYDrones ArduIMU+ V3 - DEV-11055 - SparkFun Electronics*.

[205] SparkFun Electronics. *SparkFun Triple Axis Accelerometer and Gyro Breakout - MPU-6050 - SEN-11028 - SparkFun Electronics*.

[206] Spectra Symbol. *FLEX SENSOR DATA SHEET 2014 - FlexSensor.pdf*.

[207] Spectra Symbol. *Flex Sensor Manufacturers | Spectra Symbol*.

[208] Spectra Symbol. *Spectra Symbol Flex Sensor*. URL: https://www.sparkfun.com/datasheets/Sensors/Flex/FLEXSENSOR(REVA1).pdf (visited on 04/17/2014).

[209] I. E. C. Standard. "60529". In: *Degrees of protection provided by enclosures* (1989).

[210] Thad Starner and Alex Pentland. "Real-time american sign language recognition from video using hidden markov models". In: *Motion-Based Recognition*. Springer, 1997, pp. 227–243.

[211] Harald Sternberg, Friedrich Keller, and Thomas Willemsen. "Precise indoor mapping as a basis for coarse indoor navigation". In: *Journal of Applied Geodesy* 7.4 (2013), pp. 231–246.

[212] William C. Stokoe. "Sign language structure: An outline of the visual communication systems of the American deaf". In: *Journal of deaf studies and deaf education* 10.1 (2005), pp. 3–37.

[213] R. L. Stratonovich. "Conditional Markov Processes". In: *Theory of Probability & Its Applications* 5.2 (Jan. 1960), pp. 156–178. ISSN: 0040-585X, 1095-7219. DOI: 10.1137/1105015.

[214] StretchSense Limited. *Products*. URL: http://stretchsense.com/sensor-kits/ (visited on 11/18/2016).

[215] StretchSense Limited. *Stretch Sensors*. 2016. URL: http://stretchsense.com/sensors/stretch/ (visited on 11/17/2016).

[216] Matt Strickland and Calvin Law. "Using a depth-sensing infrared camera system to access and manipulate medical imaging from within the sterile operating field". In: *Canadian Journal of Surgery* 56.3 (2013), E1.

[217] David J. Sturman and David Zeltzer. "A Survey of Glove-based Input". In: *IEEE Comput. Graph. Appl.* 14.1 (Jan. 1994), pp. 30–39. ISSN: 0272-1716. DOI: 10.1109/38.250916.

[218] Yanhua Sun, Noriaki Kuwahara, and Kazunari Morimoto. "Development of Recognition System of Japanese Sign Language Using 3D Image Sensor". In: *International Conference on Human-Computer Interaction*. Springer, 2013, pp. 286–290.

[219] Poonam Suryanarayan, Anbumani Subramanian, and Dinesh Mandalapu. "Dynamic hand pose recognition using depth data". In: *Pattern Recognition (ICPR), 2010 20th International Conference on*. IEEE, 2010, pp. 3105–3108.

[220] Tomoichi Takahashi and Fumio Kishino. "Hand Gesture Coding Based on Experiments Using a Hand Gesture Interface Device". In: *SIGCHI Bull.* 23.2 (Mar. 1991), pp. 67–74. ISSN: 0736-6906. DOI: 10.1145/122488.122499.

[221] Desney Tan et al. "Recognizing gestures from forearm emg signals". US20090327171 A1. Dec. 2009.

[222] Thalmic Labs. *Myo - Gesture control armband by Thalmic Labs*. URL: https://www.thalmic.com/en/myo/ (visited on 04/17/2014).

[223] Thalmic Labs. *Tech Specs | Myo Battery Life, Dimensions, Compatibility, and More*.

[224] Kai Ming Ting et al. "Defying the gravity of learning curve: a characteristic of nearest neighbour anomaly detectors". In: *Machine Learning* 106.1 (2017), pp. 55–91.

[225] A. S Tolba. "GloveSignature: A Virtual-Reality-Based System for Dynamic Signature Verification". In: *Digital Signal Processing* 9.4 (Oct. 1999), pp. 241–266. ISSN: 1051-2004. DOI: 10.1006/dspr.1999.0351.

[226] Pedro Trindade, Jorge Lobo, and Joao P. Barreto. "Hand gesture recognition using color and depth images enhanced with hand angular pose data". In: IEEE, Sept. 2012, pp. 71–76. DOI: 10.1109/MFI.2012.6343032.

[227] Eleni Tsironi, Pablo Barros, and Stefan Wermter. "Gesture Recognition with a Convolutional Long Short-Term Memory Recurrent Neural Network". In: *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. Bruges, Belgium, 2016, pp. 213–218.

[228] C. Tzafestas et al. "Gestural teleoperation of a mobile robot based on visual recognition of sign language static handshapes". In: *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*. Sept. 2009, pp. 1073–1079. DOI: 10.1109/ROMAN.2009.5326235.

[229] UCMP Bekerley. *The Electromagnetic Spectrum*. URL: http://www.ucmp.berkeley.edu/education/dynamic/session5/sess5_electromagnetic.htm (visited on 11/18/2016).

[230] Ali Unluturk, Omer Aydogdu, and Ufuk Guner. "Design and PID control of two wheeled autonomous balance robot". In: *Electronics, Computer and Computation (ICECCO), 2013 International Conference on*. IEEE, 2013, pp. 260–264.

[231] US Department of the Army. *FM 3-04.513 Battlefield Recovery and Evacuation of Aircraft*. Sept. 2000. URL: http://www.globalsecurity.org/military/library/policy/army/fm/3-04-513/index.html (visited on 11/23/2016).

[232] J. Van Vaerenbergh et al. "A neural network for recognizing movement patterns during repetitive self-paced movements of the fingers in opposition to the thumb". In: *Journal of Rehabilitation Medicine* 33.6 (Nov. 2001), pp. 256–259. ISSN: 1650-1977.

[233] Vladimir Naumovich Vapnik and Samuel Kotz. *Estimation of dependences based on empirical data*. Vol. 40. Springer-Verlag New York, 1982.

[234] Virtual Realities, Ltd. *Cyberglove | Data Glove*. URL: http://www.vrealities.com/products/data-gloves/cyberglove-ii (visited on 04/17/2014).

[235] Stefan Waldherr, Roseli Romero, and Sebastian Thrun. "A gesture based interface for human-robot interaction". In: *Autonomous Robots* 9.2 (2000), pp. 151–173.

[236] C. Wang and D. J. Cannon. "A virtual end-effector pointing system in point-and-direct robotics for inspection of surface flaws using a neural network based skeleton transform". In: *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE, 1993, pp. 784–789.

[237] Hanjie Wang, Qi Wang, and Xilin Chen. "Hand posture recognition from disparity cost map". In: *Asian Conference on Computer Vision*. Springer, 2012, pp. 722–733.

[238] Jeen-Shing Wang and Fang-Chen Chuang. "An Accelerometer-Based Digital Pen With a Trajectory Recognition Algorithm for Handwritten Digit and Gesture Recognition". In: *IEEE Transactions on Industrial Electronics* 59.7 (July 2012), pp. 2998–3007. ISSN: 0278-0046, 1557-9948. DOI: 10.1109/TIE.2011.2167895.

[239] Jiang Wang et al. "Robust 3d action recognition with random occupancy patterns". In: *Computer vision–ECCV 2012*. Springer, 2012, pp. 872–885.

[240] Frank Weichert et al. "Analysis of the accuracy and robustness of the leap motion controller". In: *Sensors* 13.5 (2013), pp. 6380–6393.

[241] S. Wise et al. "Evaluation of a fiber optic glove for semi-automated goniometric measurements". In: *Journal of rehabilitation research and development* 27.4 (1990), pp. 411–424. ISSN: 0748-7711.

[242] Di Wu and Ling Shao. "Deep Dynamic Neural Networks for Gesture Segmentation and Recognition". In: *Computer Vision - ECCV 2014 Workshops*. Ed. by Lourdes Agapito, Michael M. Bronstein, and Carsten Rother. Vol. 8925. Cham: Springer International Publishing, 2015, pp. 552–571.

[243] Yeh-Kuang Wu et al. "Using HMMs and Depth Information for Signer-Independent Sign Language Recognition". In: *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*. Springer, 2013, pp. 79–86.

[244] Dan Xu et al. "Real-time dynamic gesture recognition system based on depth perception for robot navigation". In: *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*. IEEE, 2012, pp. 689–694.

[245] Jie Yang and Yangsheng Xu. *Hidden markov model for gesture recognition*. Tech. rep. DTIC Document, 1994.

[246] Ruiduo Yang and Sudeep Sarkar. "Coupled grouping and matching for sign and gesture recognition". In: *Computer Vision and Image Understanding* 113.6 (2009), pp. 663–681.

[247] Paul M. Yanik et al. "Use of kinect depth data and growing neural gas for gesture based robot control". In: *2012 6th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth) and Workshops*. IEEE, 2012, pp. 283–290.

[248] M. Yeasin and S. Chaudhuri. "Visual understanding of dynamic hand gestures". In: *Pattern Recognition* 33.11 (Nov. 2000), pp. 1805–1817. ISSN: 0031-3203. DOI: 10.1016/S0031-3203(99)00175-2.

[249] Byoung-Kee Yi and Christos Faloutsos. "Fast Time Sequence Indexing for Arbitrary Lp Norms". In: *Proceedings of the 26th International Conference on Very Large Data Bases*. VLDB '00. San Francisco, CA, USA, 2000, pp. 385–394. ISBN: 1-55860-715-3.

[250] Xiaoming Yin and Ming Xie. "Finger identification and hand posture recognition for human–robot interaction". In: *Image and Vision Computing* 25.8 (2007), pp. 1291–1300.

[251] Xiaoming Yin and Xing Zhu. "Hand posture recognition in gesture-based human-robot interaction". In: *Industrial Electronics and Applications, 2006 1ST IEEE Conference on*. IEEE, 2006, pp. 1–6.

[252] Xenophon Zabulis, Haris Baltzakis, and Antonis Argyros. "Vision-based hand gesture recognition for human-computer interaction". In: *The Universal Access Handbook. LEA* (2009), pp. 34–1.

[253] L.A. Zadeh. "Fuzzy sets". In: *Information and Control* 8.3 (1965), pp. 338–353. ISSN: 0019-9958. DOI: http://dx.doi.org/10.1016/S0019-9958(65)90241-X.

[254] Chenyang Zhang and Yingli Tian. "Edge enhanced depth motion map for dynamic hand gesture recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2013, pp. 500–505.

[255] Zhengyou Zhang. "Microsoft kinect sensor and its effect". In: *IEEE multimedia* 19.2 (2012), pp. 4–10.

[256] Hong-Min Zhu and Chi-Man Pun. "Real-time hand gesture recognition from depth image sequences". In: *Computer Graphics, Imaging and Visualization (CGIV), 2012 Ninth International Conference on*. IEEE, 2012, pp. 49–52.

Abstract

Glove-based gesture recognition for real-time outdoors robot control

Although gesture recognition has been studied for several decades, much research stays in the realm of indoors laboratory experiments. In this thesis, we address the problem of designing a truly usable, real-world gesture recognition system, focusing mainly on the real-time control of an outdoors robot for use by military soldiers. The main contribution of this thesis is the development of a real-time gesture recognition pipeline, which can be taught in a few minutes with: very sparse input ("small data"); freely user-invented gestures; resilience to user mistakes during training; and low computation requirements. This is achieved thanks to two key innovations: first, a stream-enabled, DTW-inspired technique to compute distances between time series; and second, an efficient stream history analysis procedure to automatically determine model hyperparameters without user intervention. Additionally, a custom, hardened data glove was built and used to demonstrate successful gesture recognition and real-time robot control. We finally show this work's flexibility by furthermore using it beyond robot control to drive other kinds of controllable systems.

Reconnaissance gestuelle par gant de données pour le contrôle temps réel d'un robot mobile

Alors que les systèmes de reconnaissance gestuelle actuels privilégient souvent un usage intérieur, nous nous intéressons à la conception d'un système dont l'utilisation est possible en environnement extérieur et en mobilité. Notre objectif est le contrôle temps-réel d'un robot mobile dont l'usage est destiné aux fantassins débarqués. La contribution principale de cette thèse est le développement d'une chaîne de reconnaissance gestuelle temps réel, qui peut être entraînée en quelques minutes avec: un faible nombre d'exemples ("small data"); des gestes choisis par l'utilisateur; une résilience aux gestes mal réalisés; ainsi qu'une faible empreinte CPU. Ceci est possible grâce à deux innovations clés: d'une part, une technique pour calculer des distances entre séries temporelles en flux, basée sur DTW; d'autre part, une rétro-analyse efficace du flux d'apprentissage afin de déterminer les hyperparamètres du modèle sans intervention de l'utilisateur. D'autre part, nous avons construit notre propre gant de données et nous l'utilisons pour confirmer expérimentalement que la solution de reconnaissance gestuelle permet le contrôle temps réel d'un robot en mobilité. Enfin, nous montrons la flexibilité de notre technique en ce sens qu'elle permet de contrôler non seulement des robots, mais aussi des systèmes de natures différentes.