Développement d'algorithmes et d'outils logiciels pour l'assistance technique et le suivi en réadaptation

Mémoire

Frédéric Schweitzer

Maîtrise en génie mécanique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Développement d'algorithmes et d'outils logiciels pour l'assistance technique et le suivi en réadaptation

Mémoire

Frédéric Schweitzer

Sous la direction de:

Alexandre Campeau-Lecours, directeur de recherche
Laurent Bouyer, codirecteur de recherche

Résumé

Ce mémoire présente deux projets de développement portant sur des algorithmes et des outils logiciels offrant des solutions pratiques à des problématiques courantes rencontrées en réadaptation.

Le premier développement présenté est un algorithme de correspondance de séquence qui s'intègre à des interfaces de contrôle couramment utilisées en pratique. L'implémentation de cet algorithme offre une solution flexible pouvant s'adapter à n'importe quel utilisateur de technologies d'assistances. Le contrôle de tels appareils représente un défi de taille puisqu'ils ont, la plupart du temps, une dimensionnalité élevée (c-à-d. plusieurs degrés de liberté, modes ou commandes) et sont maniés à l'aide d'interfaces basées sur de capteurs de faible dimensionnalité offrant donc très peu de commandes physiques distinctes pour l'utilisateur. L'algorithme proposé se base donc sur de la reconnaissance de courts signaux temporels ayant la possibilité d'être agencés en séquences. L'éventail de combinaisons possibles augmente ainsi la dimensionnalité de l'interface. Deux applications de l'algorithme sont développées et testées. La première avec une interface de contrôle par le souffle pour un bras robotisé et la seconde pour une interface de gestes de la main pour le contrôle du clavier-souris d'un ordinateur.

Le second développement présenté dans ce mémoire porte plutôt sur la collecte et l'analyse de données en réadaptation. Que ce soit en milieux cliniques, au laboratoires ou au domicile, nombreuses sont les situations où l'on souhaite récolter des données. La solution pour cette problématique se présente sous la forme d'un écosystème d'applications connectées incluant serveur et applications web, mobiles et embarquée. Ces outils logiciels sont développés sur mesure et offrent un procédé unique, peu coûteux, léger et rapide pour la collecte, la visualisation et la récupération de données. Ce manuscrit détaille une première version en décrivant l'architecture employée, les technologies utilisées et les raisons qui ont mené à ces choix tout en guidant les futures itérations.

Abstract

This Master's thesis presents two development projects about algorithms and software tools providing practical solutions to commonly faced situations in rehabilitation context.

The first project is the development of a sequence matching algorithm that can be integrated to the most commonly used control interfaces. The implementation of this algorithm provides a flexible solution that can be adapted to any assistive technology user. The control of such devices represents a challenge since their dimensionality is high (i.e., many degrees of freedom, modes, commands) and they are controlled with interfaces based on low-dimensionality sensors. Thus, the number of actual physical commands that the user can perform is low. The proposed algorithm is based on short time signals that can be organized into sequences. The multiple possible combinations then contribute to increasing the dimensionality of the interface. Two applications of the algorithm have been developed and tested. The first is a sip-and-puff control interface for a robotic assistive arm and the second is a hand gesture interface for the control of a computer's mouse and keyboard.

The second project presented in this document addresses the issue of collecting and analyzing data. In a rehabilitation's clinical or laboratory environment, or at home, there are many situations that require gathering data. The proposed solution to this issue is a connected applications ecosystem that includes a web server and mobile, web and embedded applications. This custom-made software offers a unique, inexpensive, lightweight and fast workflow to visualize and retrieve data. The following document describes a first version by elaborating on the architecture, the technologies used, the reasons for those choices, and guide the next iterations.

Table des matières

Liste des tableaux

Liste des figures

De même que l'arbre produit,
chaque année, le même fruit,
mais qui pourtant est chaque fois
nouveau, les idées qui ont une
valeur permanente doivent être
créées de nouveau.

Albert Schweitzer

Remerciements

Je mentirais si je disais que mes études graduées ont été difficiles. Bien sûr, le chemin a été parsemé d'obstacles et la difficulté des projets était bel et bien au rendez-vous. Par contre, lorsque ceux-ci sont stimulants et que l'on travaille dans un environnement propice à l'épanouissement, surmonter tout cela se fait bien plus facilement.

Pour cela, je voudrais remercier mon directeur de maîtrise, Alexandre Campeau-Lecours, et mon co-directeur, Laurent Bouyer, pour leur support et leur confiance. Vous m'avez initié à la recherche et au développement dans un domaine stimulant où l'on peut faire la différence dans la vie des gens. Alexandre, tu as été un mentor extraordinaire. Tu as toujours su me pousser dans la bonne direction tout en me faisant confiance pour mener à bien mes projets à ma façon et absorber une multitude de connaissances au passage.

J'aimerais remercier tous les membres du laboratoire de robotique que j'ai côtoyés durant ces deux dernières années. Les discussions passionnées que nous avions n'ont pas toujours paru productives, mais ont certainement influencé positivement mes travaux.

Finalement, j'aimerais remercier spécialement mes parents et aussi Émilie, qui a partagé ma vie tout au long cette aventure, pour m'avoir offert tout le support dont j'ai eu besoin pour en arriver où j'en suis. Vos contributions sont imprégnées dans chacune des pièces de ce travail.

Avant-propos

Cet ouvrage est présenté par l'insertion de deux articles qui sont présentés dans le premier et le second chapitre. Ces articles sont étroitement liés et décrivent les itérations de développement qui ont permis l'élaboration, l'implémentation et la mise à l'essai d'algorithmes offrant une approche alternative pour le contrôle de technologies d'assistance aux personnes vivant avec un handicap.

Le premier article a été soumis en février 2020 à la conférence *Rehabilitation Engineering and Assistive Technology Society of North America (RESNA)* qui devait avoir lieu en juillet 2020 à Washington D.C., mais qui s'est finalement déroulée de façon virtuelle en septembre 2020. Celui-ci présente le développement et l'expérimentation d'une première application de l'interface de contrôle. L'étudiant est l'auteur principal, sous la supervision du directeur de recherche Alexandre Campeau-Lecours.

Le second article repose sur la preuve de concept établie par le premier article et pousse le projet encore plus loin avec le développement d'une autre itération. Encore une fois, les méthodes de développement et d'analyse y sont détaillées. Ce papier a été soumis en avril 2021 au journal en accès libre *Signals* du *Multidisciplinary Digital Publishing Institute (MDPI)*. L'étudiant est l'auteur principal, sous la supervision du directeur de recherche Alexandre Campeau-Lecours.

Le dernier chapitre, quant à lui, explore une autre problématique rencontrée en réadaptation, cette fois-ci au niveau de la collecte de données et du suivi (*monitoring*). Ce chapitre présente le développement d'une première version d'une suite d'applications facilitant la collecte sans fil et visualisation rapide de données. Cette dernière section du mémoire documente l'architecture et les technologies employées pour ces applications, ainsi que les raisons qui ont mené à ces choix.

Introduction

Une interface est un élément fascinant permettant à deux entités distinctes de travailler ensemble. On parle ici d'interactions entre humains et machines créant des écosystèmes et répondant à des fonctions précises dans divers domaines. Tandis que celles entre les humains sont étudiées depuis des siècles par les sciences sociales, les interactions humain-machine et machine-machine représentent des champs d'études plus récents qui sont de plus en plus intégrés en industrie et à la vie de tous les jours.

Les interfaces mettant en oeuvre l'Humain et la technologie sont généralement composées d'assemblages mécaniques, de capteurs électroniques et d'algorithmes intelligents. Que ce soit pour contrôler un robot, envoyer des commandes à un ordinateur ou faire fonctionner une maison intelligente, l'interface est vitale à l'interaction humain-machine. Il est évident qu'une interface bien conçue et bien adaptée donnera à son utilisateur une impression d'aisance et de confiance lors de son utilisation et ce dernier saura alors, intuitivement, comment interagir avec la technologie [1, 2]. À l'opposé, lorsque déficiente, elle entraînera un sentiment d'encombrement comme si l'utilisateur devait se concentrer davantage sur le fonctionnement de l'interface en tant que telle plutôt que l'action finale désirée au niveau de la technologie. Quand on utilise le clavier pour interagir avec l'ordinateur, écrire le prochain mot de cette phrase se fait naturellement. S'il fallait penser à comment utiliser l'interface pour écrire le mot, personne ne l'utiliserait. De même, utiliser des commandes vocales pour contrôler sa maison intelligente est bien pratique, mais si l'interface n'interprète pas correctement les mots prononcés, l'utilisateur perd confiance en celle-ci et cesse simplement de l'utiliser. Pour ces raisons, en ingénierie de la réadaptation, l'interface est souvent présentée comme étant le talon d'Achilles du système [3, 4]. Dans ce domaine, les cas de figure d'utilisation d'interface de contrôle sont nombreux. Par exemple, lorsqu'une personne vivant avec un handicap utilise une technologie d'assistance au quotidien. L'utilisateur a donc besoin d'une interface de contrôle intuitive et propre à ses capacités. Par conséquent, les notions de de confort et d'aisances d'utilisations sont d'autant plus importantes, mais encore plus complexes à concevoir et implémenter.

L'interaction entre appareils technologiques quant à elle est beaucoup plus «objective». Elle ne dépend pas d'émotions ou de perceptions. Ces appareils technologiques peuvent être des ordinateurs, des téléphones intelligents, des capteurs, des actionneurs ou n'importe quels appareils

pouvant être mis en réseau avec un autre [5]. Traditionnellement, le terme SCADA (*Supervisory Control And Data Acquisition*) désigne ces systèmes dont l'objectif principal est la collecte de données, le suivi et le contrôle [6] sans intervention humaine. Aujourd'hui, une interface machine-machine est de plus en plus synonyme d'Internet des objets (IoT) [7, 8, 9, 10, 11]. Ce nouveau paradigme dans l'industrie et dans plusieurs sphères de la société, telles que la santé et même la vie de tous les jours, représente une nouvelle approche pour le design de systèmes impliquant plusieurs appareils technologiques. L'IoT tire profit de l'interconnectivité entre ceux-ci pour offrir une solution abordable et fluide pour implémenter les systèmes SCADA.

Problématiques

Ce mémoire explore le développement et l'application d'interfaces en réadaptation. Spécifiquement, il tente d'apporter des solutions à deux problèmes fréquemment rencontrés dans ce domaine.

En premier lieu, on s'intéresse aux interfaces de contrôle. Une technologie d'assistance telle qu'un bras robotisé, un fauteuil roulant motorisé, un exosquelette actif ou simplement un ordinateur obligent souvent l'utilisateur à contrôler plusieurs degrés de libertés, modes ou commandes (dimensions). En soi, cela peut déjà s'avérer complexe, mais cela ne représente que la moitié du problème. Dans la plupart des cas, ces multiples dimensions doivent être contrôlées avec une interface basée sur des capteurs qui offre bien moins de commandes physiques qu'il ne peut y avoir de dimensions à contrôler. Par exemple, un bras robotisé qui a six degrés de liberté en plus des doigts et de trois ou quatre fonctions d'accès rapides peut être contrôlé avec une paille et un capteur de pression qui ne permettent à l'utilisateur que de souffler ou aspirer pour contrôler toutes ces dimensions.

Le second problème porte plutôt sur les interfaces d'échange de données. En effet, à des fins de suivi et d'analyse, en laboratoire ou en milieu clinique, il est souvent nécessaire de collecter des données pour pouvoir ensuite les analyser. Par exemple, lors d'une expérience en laboratoire, des capteurs doivent normalement être portés par le participant. En pratique, la procédure et les appareils utilisés pour la collecte varient. Les capteurs les plus coûteux viennent souvent avec leur propre mécanisme d'enregistrement, tandis que pour les autres, on a souvent recourt à un périphérique de stockage externe et les données doivent être par la suite transférées manuellement sur un ordinateur pour être analysées. De plus, cette dernière façon de faire implique généralement des systèmes encombrants pour le participant. Cela rend le processus très peu fluide.

Objectifs de maîtrise

En lien avec les deux problématiques énoncées, les objectifs de ce mémoire sont donc divisés de la même façon.

1. Le premier objectif en lien avec les interfaces de contrôle est donc de développer des algorithmes intelligents dans le but d'améliorer des systèmes de contrôle de technologies d'assistance couramment utilisées. Ces algorithmes visent principalement les interfaces de faible dimensionnalité, c'est-à-dire avec des commandes physiques limitées, pour le contrôle d'appareils de haute dimensionnalité.
2. Le second objectif portant sur les interfaces de collecte et d'échange de données est de développer un système d'applications connectées facilitant ces actions. Ces applications ont pour but de faciliter le flux de travail pour la collecte, le suivi et l'analyse de données dans un contexte de réadaptation en tirant avantage de l'interconnectivité entre les appareils technologiques et de paradigmes de développement modernes pour les interfaces machine-machine.

Ainsi, ces deux projets visent à améliorer la qualité de vie des utilisateurs d'aides techniques dans leurs activités quotidiennes, mais aussi de faciliter le développement et l'évaluation de ces technologies en fournissant des outils efficaces pour les chercheurs et les professionnels en réadaptation.

Plan du mémoire

Les projets sont développés à l'aide de méthodes itératives, interdisciplinaires et flexibles. Le chapitre 1 présente les premiers développements de l'algorithme général étudié dans le cadre de la maîtrise, ainsi qu'une première application à une interface de contrôle par le souffle. Ce chapitre se veut moins technique d'un point de vue d'ingénierie pour se concentrer plutôt sur des aspects plus appliqués et centrés sur l'utilisateur. Les développements et évaluations qui y sont décrits représentent une preuve de concept solide pour la suite du projet détaillé au chapitre 2. Ce dernier apporte alors une vision beaucoup plus mathématique et technique sur l'algorithme développé, appelé *algorithme de correspondance de séquences* (ou *sequence matching algorithm*). On y présente aussi une seconde application pour le contrôle d'un ordinateur avec des gestes de la main. Cette application y est implémentée et évaluée complétant la seconde itération du projet.

Le dernier chapitre change quelque peu de sujet et présente une première itération de solution proposée pour répondre au second objectif de la maîtrise. Le but de ce chapitre est autant de présenter les travaux réalisés que de guider ceux à venir.

Bibliographie

[1] A. Lebrasseur, J. Lettre, F. Routhier, P. S. Archambault, and A. Campeau-Lecours, "Assistive robotic arm : Evaluation of the performance of intelligent algorithms," *Assistive Technology*, 2019.

[2] V. Maheu, P. S. Archambault, J. Frappier, and F. Routhier, "Evaluation of the JACO robotic arm : Clinico-economic study for powered wheelchair users with upper-extremity disabilities," in *IEEE International Conference on Rehabilitation Robotics*, 2011.

[3] N. Friedman, A. Cuadra, R. Patel, S. Azenkot, J. Stein, and W. Ju, "Voice assistant strategies and opportunities for people with tetraplegia," in *ASSETS 2019 - 21st International ACM SIGACCESS Conference on Computers and Accessibility*, (New York, NY, USA), pp. 575–577, Association for Computing Machinery, Inc, oct 2019.

[4] L. V. Herlant, R. M. Holladay, and S. S. Srinivasa, "Assistive teleoperation of robot arms via automatic time-optimal mode switching," in *ACM/IEEE International Conference on Human-Robot Interaction*, vol. 2016-April, pp. 35–42, IEEE Computer Society, apr 2016.

[5] M. Chen, J. Wan, and F. Li, "Machine-to-machine communications : Architectures, standards and applications.," *Ksii transactions on internet & information systems*, vol. 6, no. 2, 2012.

[6] A. Daneels and W. Salter, "What is scada ?," 1999.

[7] D. Raposo, A. Rodrigues, S. Sinche, J. Sá Silva, and F. Boavida, "Industrial iot monitoring : Technologies and architecture proposal," *Sensors*, vol. 18, no. 10, p. 3568, 2018.

[8] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things : Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[9] J. Cheng, W. Chen, F. Tao, and C.-L. Lin, "Industrial iot in 5g environment towards smart manufacturing," *Journal of Industrial Information Integration*, vol. 10, pp. 10–19, 2018.

[10] M. S. Hossain and G. Muhammad, "Cloud-assisted industrial internet of things (iiot)–enabled framework for health monitoring," *Computer Networks*, vol. 101, pp. 192–202, 2016.

[11] Y. J. Fan, Y. H. Yin, L. Da Xu, Y. Zeng, and F. Wu, "Iot-based smart rehabilitation system," *IEEE transactions on industrial informatics*, vol. 10, no. 2, pp. 1568–1577, 2014.

Chapitre 1

Intuitive Sequence Matching Algorithm Applied to a Sip-and-Puff Control Interface for Robotic Assistive Devices

1.1 Résumé

Cet article présente le développement et la validation d'une interface de contrôle base sur un algorithme de correspondance de séquence (ACS). Un défi important dans le domaine des technologies d'assistance est de permettre aux utilisateurs de contrôler une technologie de haute dimensionnalité (ex. robot à plusieurs degrés de liberté) avec une interface de basse dimensionnalité (ex. quelques boutons). L'ACS consiste en la reconnaissance d'un patron obtenu de signaux de capteurs comparés à une banque de patrons prédéfinis. L'objectif est de permettre à l'utilisateur d'entrer plusieurs commandes avec l'interface de basse dimensionnalité (ex. Le code Morse définit plusieurs lettres avec un seul bouton). L'algorithme est utilisé ici pour le contrôle d'un bras robotisé d'aide et a été adapté à une interface de contrôle par le souffle. Une validation préliminaire avec des sujets sains a démontré que l'ACS rend le contrôle plus rapide, facile et confortable.

1.2 Abstract

This paper presents the development and preliminary validation of a control interface based on a sequence matching algorithm. An important challenge in the field of assistive technology is for users to control high dimensionality devices (e.g., assistive robot with several degrees of freedom, or computer) with low dimensionality control interfaces (e.g., a few switches). Sequence matching consists in the recognition of a pattern obtained from a sensor's signal

compared to a predefined pattern library. The objective is to allow the user to input several different commands with a low dimensionality interface (e.g., Morse code allowing inputting several letters with a single switch). In this paper, the algorithm is used in the context of the control of an assistive robotic arm and has been adapted to a sip-and-puff interface. A preliminary validation with healthy subjects has shown that sequence matching makes the control faster, easier and more comfortable.

1.3 Introduction

Interactions between human and technology is well documented and known as "human-robot interactions" in industrial robotics. When it comes to assistive technology (AT), the importance to develop around this aspect of engineering is high and the human part becomes predominant. On the one hand, a device like an assistive robotic arm can help a person gain more autonomy in their daily life [1]. However, the control interface (the link between the human operator and the AT) is often portrayed as the system's Achilles heel preventing users to efficiently control their AT [2, 3]. This is amplified by the fact that each user has his/her own abilities and may require specific control interface for his/her condition. For instance, controlling a six-degree-of-freedom robotic arm or a computer with a low dimensional interface is not always an easy task. Assistive robotic arm users need to control many degrees of freedom and components (translations, rotations, fingers, etc.) with only a two-axis joystick and switches [4]. Other users rely solely on a sip-and-puff control interface, which makes the control even more difficult since they only use two inputs (inhale and exhale) to control the many degrees of freedom. These users must thus navigate through mode selection to activate different control modes, which can be a time-consuming, demanding and sometimes impossible process.

In recent years, alternative control interfaces have emerged for ATs. Among the alternatives, assuming the user can speak clearly enough, voice control interfaces represent a valuable option. Commercial voice assistants can help people living with disabilities in various domestic tasks such as making phone calls, playing music, or handling lights [2]. Although these assistants require a constant Internet connection, it's also possible to develop offline interfaces [5]. Some people prefer control with IMUs. Users who can move their head and/or lower limbs easily can wear IMUs and control a robot [6] or a computer mouse [7]. Electromyography (EMG) interfaces may also be an option [8] in specific cases. Finally, sip-and-puff or tongue control devices [9] are widely used but are such low dimensional interface that they represent a big challenge for the user. These devices can be more efficiently interfaced with an AT using intelligent algorithms. To that effect, automatic mode switching techniques are in development [3], but may have their limitations for now. In addition, these algorithms are well suited for robotic arms, but the interface should also be adaptable to different ATs. Some of these emerging control interfaces are somewhat cumbersome in practice and may require

many calibrations by the user. This may explain why standard interfaces such as joysticks and sip-and-puff devices are predominant to this day in practice

1.4 Objectives

The objective of this project is to develop and evaluate an intuitive control interface based on currently used devices (e.g., switches and joysticks) and to combine them with intelligent algorithms to enhance the user's control interface and help people living with disabilities to control their assistive devices. This paper presents a sequence matching algorithm as a proof of concept, and shows that using low dimensionality sensors with intelligent signal recognition algorithms can improve the control of ATs

1.5 Sequence Matching Algorithm

This section presents the proposed sequence matching algorithm. The general idea behind the development of this algorithm is to detect a sequence of actions executed by the user and to map those sequences to the controls of assistive devices (e.g., mode selection of a robotic arm, keyboard shortcuts on a computer, home automation). For a sequence to be matched, it must correspond to a preset user-defined sequence (UDS). The sequence to match can vary from simple units (short or long presses) to more complex actions (e.g., comparing two continuous analog signals at 1 kHz). In the specific context of this paper, the algorithm was applied to a sip-and-puff sensor with four possible actions: short sip (inhaling), long sip (inhaling), short puff (exhaling), long puff (exhaling). The actions are defined in the software respectively as the following signals: 1, 2, -1 and -2. In this paper, the signals are used to control an assistive robotic arm with several degrees of freedom. The available modes are:

1. forward-backward translations,
2. left-right translations,
3. up-down translations,
4. x-axis rotation,
5. y-axis rotation,
6. z-axis rotation,
7. open-close fingers,
8. save a point,
9. go to a saved point.

Figure 1.1 is a visual representation of the general algorithm. In the following description, numbers in parentheses refer to a corresponding block in Figures 1.1, 1.2 and 1.3. The program starts in detection mode (1) and remains in that mode until the current sequence (CS) matches

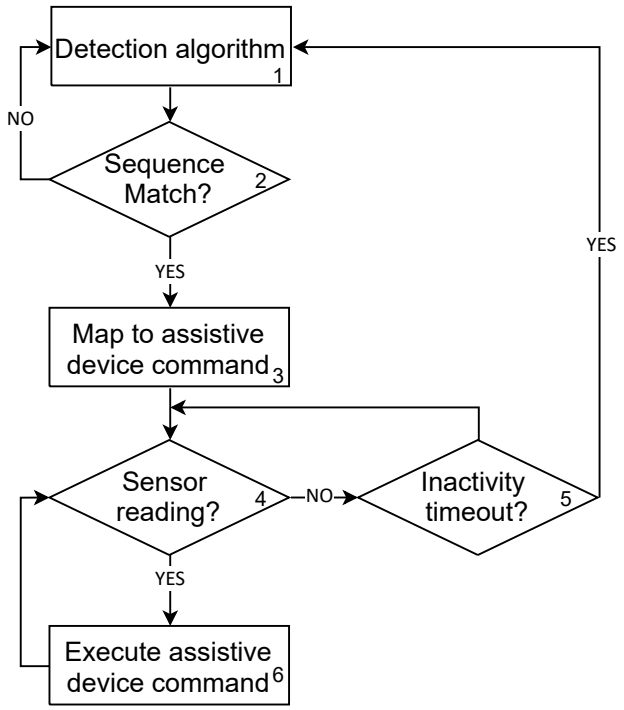


FIGURE 1.1 – General algorithm.

one of the user-defined sequences. When this happens, the robot is ready to receive a command (i.e., moving according to the selected mode) (3). For instance, if the mode forward-backward has been selected, the robot will move forward with an exhalation and backwards with an inhalation (6). The command mapping can also be user-defined, meaning that the user can tell the system which sequence corresponds to which mode. After a moment of inactivity (no inhale or exhale) (5), the algorithm returns to detection mode.

The mode detection algorithm is detailed in Figure 1.2. The algorithm acquires sensor data (8) to detect peaks and determine their nature (9). A peak can be short or long and up (exhale) or down (inhale). It is detected with thresholds on the 0-5 V signal. The time difference between the moment the signal goes up and the moment it comes down (or the other way around) determines if it is long or short. Detected peaks are stacked in a list (either as -2, -1, 2, 2) (10) until a UDS is matched (14) or the list is reset.

Figure 1.3 shows the detailed match algorithm which task is to determine what to do with the CS. For instance, given the UDSs:

$$S_1 = \langle 1, 2, -1 \rangle \tag{1.1}$$

$$S_2 = \langle 1, 2 \rangle \tag{1.2}$$

$$S_3 = \langle 2, 1 \rangle , \tag{1.3}$$

if the CS (15) inputted by the user is $S_0 = \langle 2 \rangle$, there is no confusion (16-17-19-22), but the program waits for a 1 to match S_3 (16-17-18-21). Otherwise, the CS is reset if the user takes

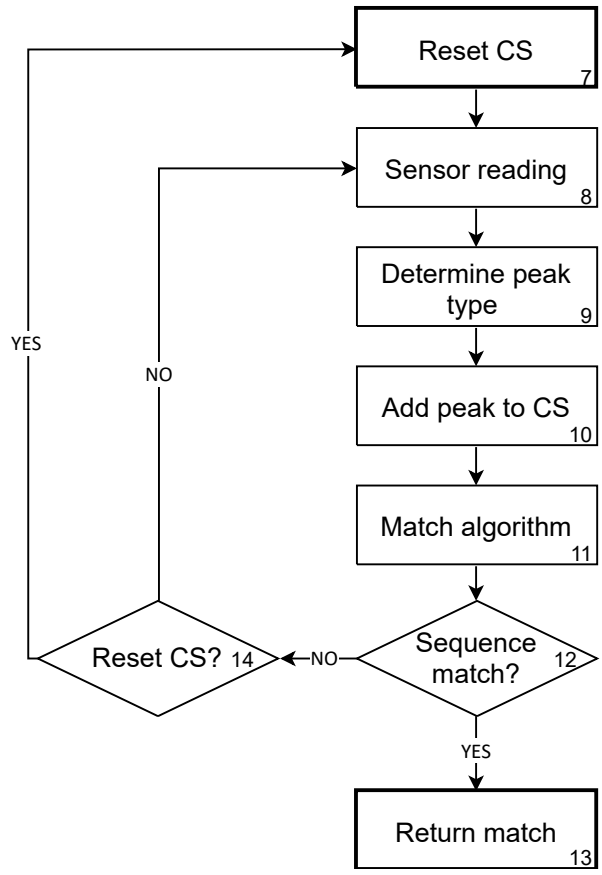


FIGURE 1.2 – Detailed sequence detection algorithm.

too much time to complete any UDS (16 -20-23). If $S_0 = \langle 1, 2 \rangle$ instead, there is confusion between S_1 and S_2 (16-17-18-22). The program has to wait to see if the user sends a new peak. If so, it shall correspond to -1 to match S_1 (16-17-18-21) otherwise the CS is reset (16-17-19-23) as it was not a viable command. If no new peaks are sent after $\langle 1, 2 \rangle$, S_2 is matched (16-20-21). This method ensures that the compatible sequences are automatically matched and the user does not lose any time waiting for his sequence to be matched. In addition, the waiting timers (5, 16) may be adjusted to the user to increase or decrease the waiting time. The whole algorithm was adapted to a sip-and-puff sensor, but the sensor part (8) can be replaced by any sensor that best suit the user's abilities.

1.6 Experimentation

As noted in the previous section, the algorithm has been adapted for a sip-and-puff interface for the control of an assistive robotic arm. In the experiments, a JACO assistive robotic arm [10] produced by Kinova Robotics was used. In order to assess the performance of the proposed control interface, three tasks have been performed. Each task was performed with both a classic sip-and-puff (named Basic Sip-and-Puff (BSP)) control for assistive robots and

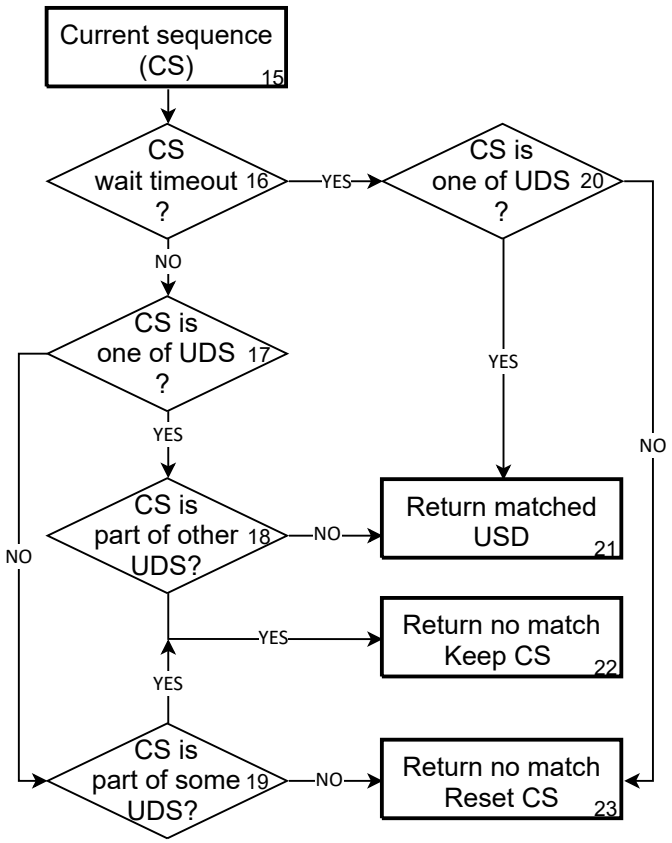


FIGURE 1.3 – Detailed match algorithm.

with proposed control algorithm (named Advanced Sip-and-Puff (ASP)). The BSP is the more common and commercialized sip-and-puff interface. It allows the user to view all modes auto-scrolling on the screen. When the desired mode is highlighted, the user exhales once to enter the mode and then inhales or exhales to go back and forth in this mode. The ASP shows a graphical interface where a sequence is displayed beside each corresponding mode (e.g., sip-sip-sip = forward-backward). Eight healthy participants (6 males and 2 females) aged between 23 and 33 years old took part in the experiments. Half the participants began with the ASP and the other half with the BSP.

The first task consisted in grabbing a jar on a shelf and to bring it and drop it on a table. The second task was to grab a plastic block holding a spoon from a shelf, take a spoonful of coffee and put it in a mug on a table. The third task consisted in taking a bottle of water on a shelf and pour water in a glass on a table. Those tasks are inspired by the work of Beaudoin et al. [11]. All tasks are within the reach of the robotic arm.

Before the experiment begins, the participants were allowed a ten-minute familiarization time with each control mode. The participants were timed for each task (full completion and moving time). Figure 4 shows all the task completion times of the participants. After the tests, the

participants were asked to answer a questionnaire, which contained questions adapted from the QUEAD of Schmidtler et al. [12], and leave their comments.

1.7 Results and Discussion

Figure 1.4, shows box plots, estimated probability densities and means, plus or minus the standard deviation (SD) of the times for the full completion of the three tasks by interface. Each point of different color on the plots corresponds to a distinct participant. The completion times with the ASP is significantly lower than the completion times with the BSP according to one-tailed, non-parametric Wilcoxon signed-rank tests (for paired data) for each task ($p-value = 5.86 \times 10^{-3} < 0.05$, for each task). Indeed, for tasks 1 to 3, the ASP completion times are respectively 35%, 30% and 42% lower than the BSP completion times. Since the tasks are the same for both interfaces, the moving time (the time the robot actually moves) should also be the same. This is confirmed by a two-tailed Wilcoxon test on the moving times (again for paired data). The test fails to reject the null hypothesis. Therefore there is no evidence to assume that the moving times are different between both interfaces ($[p-value = 0.433] > 0.05$, on ASP – BSP difference: $mean = 0.54$, $SD = 3.80$ seconds). The extra time from ASP to BSP is therefore wasted time during which the participants wait for the interface to keep up. This is consistent with the fact that the variances of estimated probability densities are higher for BSP than ASP on Figure 1.4. This strong variability shows that the participants made more mistakes (i.e., missing a mode during the auto-scroll) using BSP, and that those mistakes cost time. The results of the questionnaire are presented in Table 1.1.

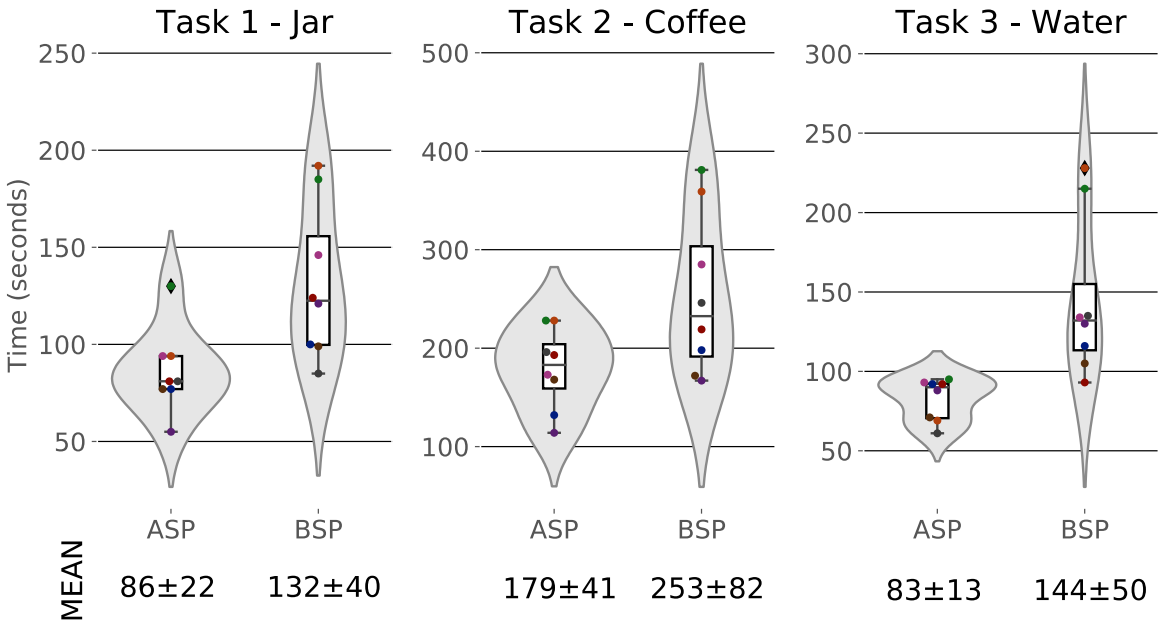


FIGURE 1.4 – Task completing time by interfaces.

TABLE 1.1 – Post experiment questionnaire.

	Completely disagree	Mostly disagree	Neutral	Mostly agree	Completely agree
Perceived usefulness					
The ASP interface is useful				1	7
The ASP interface enhances my performance				1	7
I could efficiently complete the tasks using the ASP interface				4	4
I could efficiently complete the tasks using the BSP interface		1	3	2	2
Perceived ease of use					
The ASP interface is easy to use				5	3
The ASP interface is easier to use than the BSP interface			2	1	5
The ASP interface feels cumbersome to use	3	3	1	1	
The BSP interface feels cumbersome to use		1		6	1
I didn't need concentration to use the ASP interface		3	2	3	
I didn't need concentration to use the BSP interface	1	2		5	
Using the ASP interface was intuitive		1	1	3	3
Emotions					
I prefer using the ASP interface rather than the BSP interface					8
Attitude					
In a long-term perspective, I think it would be faster to use the ASP interface					8
In a long-term perspective, I think it would be difficult to learn multiple command of the ASP interface	6	2			

In general, participants found the ASP more useful and easier to use compared to the BSP and they preferred using the ASP. On the other hand, the participants thought using ASP required more concentration than using BSP. However, they also mentioned that, with an adaptation and learning period, the ASP would become much more comfortable to use. This leads to conclude that the difference between both interfaces is the control mode selection, which is less time-consuming with the ASP.

1.8 Conclusion

Controlling assistive technologies of high dimensionality with a low dimension interface is time- and energy-consuming for many users. This paper has presented a sequence matching algorithm that was applied to the control of an assistive robotic arm with a sip-and-puff interface. The experiments conducted show that the algorithm made the control faster and the interface more comfortable. This preliminary validation with healthy subjects is encouraging. The next steps will consist in validating the algorithm with actual assistive robot users along as to develop a new version of the algorithm that can work with continuous signals obtained from different sensors such as inertial measurement units in order to detect custom movements.

1.9 Bibliographie

[1] V. Maheu, P. S. Archambault, J. Frappier, and F. Routhier, "Evaluation of the JACO robotic arm : Clinico-economic study for powered wheelchair users with upper-extremity disabilities," in *IEEE International Conference on Rehabilitation Robotics*, 2011.

[2] N. Friedman, A. Cuadra, R. Patel, S. Azenkot, J. Stein, and W. Ju, "Voice assistant strategies and opportunities for people with tetraplegia," in *ASSETS 2019 - 21st International ACM SIGACCESS Conference on Computers and Accessibility*, (New York, NY, USA), pp. 575–577, Association for Computing Machinery, Inc, oct 2019.

[3] L. V. Herlant, R. M. Holladay, and S. S. Srinivasa, "Assistive teleoperation of robot arms via automatic time-optimal mode switching," in *ACM/IEEE International Conference on Human-Robot Interaction*, vol. 2016-April, pp. 35–42, IEEE Computer Society, apr 2016.

[4] A. Lebrasseur, J. Lettre, F. Routhier, P. S. Archambault, and A. Campeau-Lecours, "Assistive robotic arm : Evaluation of the performance of intelligent algorithms," *Assistive Technology*, 2019.

[5] S. Poirier, F. Routhier, and A. Campeau-Lecours, "Voice control interface prototype for assistive robots for people living with upper limb disabilities," in *IEEE International Conference on Rehabilitation Robotics*, vol. 2019-June, pp. 46–52, IEEE Computer Society, jun 2019.

[6] C. L. Fall, P. Turgeon, A. Campeau-Lecours, V. Maheu, M. Boukadoum, S. Roy, D. Massicotte, C. Gosselin, and B. Gosselin, "Intuitive wireless control of a robotic arm for people living with an upper body disability," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, vol. 2015-November, pp. 4399–4402, Institute of Electrical and Electronics Engineers Inc., nov 2015.

[7] R. Raya, J. O. Roa, E. Rocon, R. Ceres, and J. L. Pons, "Wearable inertial mouse for children with physical and cognitive impairments," in *Sensors and Actuators, A : Physical*, vol. 162, pp. 248–259, Elsevier, aug 2010.

[8] C. L. Fall, G. Gagnon-Turcotte, J. F. Dube, J. S. Gagne, Y. Delisle, A. Campeau-Lecours, C. Gosselin, and B. Gosselin, "Wireless sEMG-Based Body-Machine Interface for Assistive Technology Devices," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, pp. 967–977, jul 2017.

[9] L. N. Andreasen Struijk, L. L. Egsgaard, R. Lontis, M. Gaihede, and B. Bentsen, "Wireless intraoral tongue control of an assistive robotic arm for individuals with tetraplegia," *Journal of NeuroEngineering and Rehabilitation*, vol. 14, pp. 1–8, nov 2017.

[10] A. Campeau-Lecours, H. Lamontagne, C. Simon Latour, C. Philippe Fauteux, C. Véronique Maheu, C. François Boucher, C. Charles Deguire, and C. L. Louis-Joseph Caron, "Kinova Modular Robot Arms for Service Robotics Applications," *International Journal of Robotics Applications and Technologies*, vol. 5, no. 2.

[11] M. Beaudoin, J. Lettre, F. Routhier, P. S. Archambault, M. Lemay, and I. Gélinas, "Long-term use of the JACO robotic arm : a case series," *Disability and Rehabilitation : Assistive Technology*, vol. 14, pp. 267–275, apr 2019.

[12] J. Schmidtler, K. Bengler, F. Dimeas, and A. Campeau-Lecours, "A questionnaire for the evaluation of physical assistive devices (quead) : Testing usability and acceptance in physical human-robot interaction," in *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*, vol. 2017-January, pp. 876–881, Institute of Electrical and Electronics Engineers Inc., nov 2017.

Chapitre 2

IMU-Based Hand Gesture Interface Implementing a Sequence-Matching Algorithm for the Control of Assistive Technologies

2.1 Résumé

Les technologies d'assistance ont souvent une dimensionnalité élevée (plusieurs degrés de liberté ou mode) tandis que les utilisateurs doivent les contrôler avec des interfaces basées sur des capteurs de faible dimensionnalité (ex. boutons). Cet article présente le développement d'une interface basée sur un algorithme de correspondance de séquences. Celui-ci permet à l'utilisateur d'exécuter plusieurs commandes à partir de capteur de faible dimensionnalité en reconnaissant non seulement l'action instantanée, mais un profil séquentiel du signal dans le temps comme un code Morse. L'algorithme est appliqué à la reconnaissance de gestes de la main captés avec des centrales inertielles portée par l'utilisateur. Un classifieur SVM, robuste sur de petits jeux de données est développé pour reconnaitre les gestes en temps réel. L'interface est utilisée pour le contrôle d'une souris et d'un clavier d'ordinateur. Elle a été comparée avec le logiciel de contrôle avec la tête AssystMouse et a démontré des résultats encourageants.

2.2 Abstract

Assistive technologies often have a high dimensionality of possible movements (e.g., degrees of freedom, modes) but the users have to control them with low-dimensionality sensors and interfaces (e.g., switches). This paper presents the development of an open-source interface based on a sequence-matching algorithm for the control of ATs. Sequence matching allows the user to input several different commands with low-dimensionality sensors by not only

recognizing their output, but also their sequential pattern through time, similarly to Morse code. The algorithm is applied to the recognition of hand gestures inputted using an inertial measurement unit worn by the user. An SVM-based algorithm that is aimed to be robust with small training sets is developed to recognize gestures in real time. The interface is applied to control a computer's mouse and keyboard. It was compared against and combined with the head-movement-based AssystMouse software. The hand gesture interface showed encouraging results for this application.

2.3 Introduction

Assistive technologies (ATs) are used all around the world by people living with all kinds of disabilities. For instance, the use of a robotic arm has been shown to help people living with upper limb disabilities with their daily tasks [1]. Another study on exoskeletons proved they have good potential for functional mobility in people with spinal cord injury [2]. While the emphasis is often put on the assistive device itself, such systems actually include three important parts: the user, the control interface and the device (e.g., robotic arm, exoskeleton). In fact, this is the case for any human-machine interaction and more often than not, the interface, which links the device to the user, seems to be the key element in the system [3, 4, 5]. So, when it comes to design, it is worth spending time on developing a valuable interface especially with ATs, on which the ability of a user to be independent in his or her daily living task may heavily rely on it. Besides, because of the specific abilities and needs being very scarce, even for the same diagnosis, the control interface needs to be either tailor-made or flexible and adaptable.

Control interfaces are built using various sensors that may be combined with intelligent algorithms. Recently, many interfaces using different technologies have emerged, and they are yet being improved. The simplest interfaces like sip-and-puff or tongue control [6, 7] are widely used in practice due to the absence of complexity in their implementation and usage. Another popular instance is voice control. It has been shown that commercial voice assistants can help people in their daily tasks by providing functionalities to interact with their phone [8], handle lights or simply play music [3]. Voice assistants often require an internet connection, but lighter versions can also be implemented offline [9]. Other options include electromyography (EMG) interfaces [10, 11, 12], which use electrical activity in muscles as input signals, and control via inertial measurement units (IMUs) [13, 14], that combine accelerometer and gyroscope signals. For instance, IMUs or EMGs can be positioned on a limb that a user can move easily (like the head, arms or legs) and act as an input mechanism to control a robotic arm or the mouse of a computer.

One major and obvious issue to address with almost any interface-AT pair is that many interfaces have limited actual physical commands to efficiently control all of the functions of their

corresponding AT. Indeed, controlling a high-dimensionality device with a low-dimensionality sensor (around which the interface is built) can be quite challenging. The dimensionality of components in this context can be interpreted as qualitative metrics to compare the work space of those components in terms of degrees of freedom (DoF), number of functions, or some other physical restrictions such as the number of actions a user can perform according to his or her abilities or the number of controls available. For instance, whereas a six-DoF assistive robotic arm may need 20 commands to operate (e.g., forward, backward, up, down, left, fingers, options, etc.), a computer may need 10 (mouse left, right, up, down, left click, right click, copy, paste, etc.). However, in both cases, the dimensionality of the control interface depends more on the abilities of the users (the actions they can perform and their reach). Therefore, many users only have access to basic devices such as switches to control the AT.

In any case, a common way to deal with the interface's lower dimensionality is often to arrange the AT's modes in sub-control groups. As shown in Figure 2.1, with this approach, accessing a mode can require a series of actions to browse through the nested groups [15, 16, 1]. Thus, a strategy needs to be implemented within the interface to either reduce the total number of modes, facilitate switching between them or somehow artificially augment the dimensionality of the interface itself.

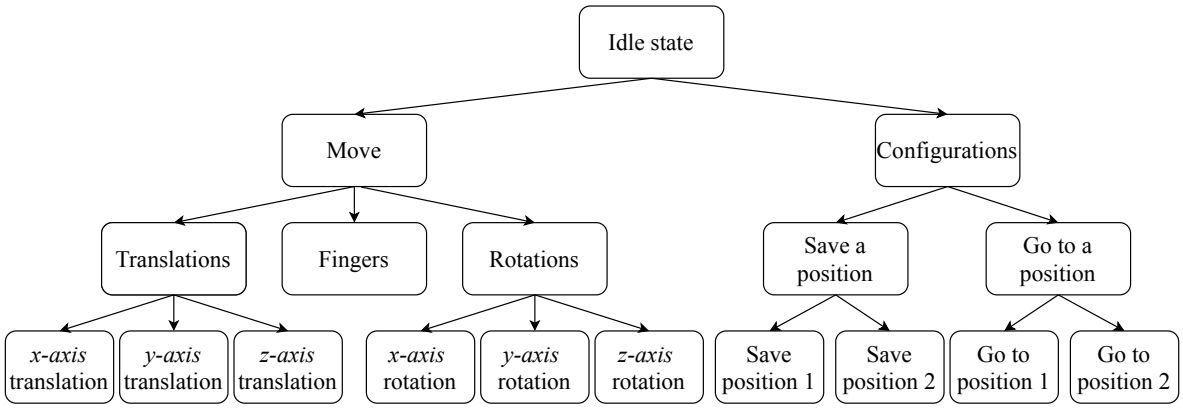


FIGURE 2.1 – Graph of possible sub-control groups for an assistive robotic arm.

The easiest and most flexible strategy to employ is to use a scanning menu panel [17, 18]. It can be implemented with or without a sub-groups architecture (as in Figure 2.1), by granting the user direct access to the modes. This may be used with a single switch interface and can accommodate virtually any number of modes. It also gives the developer the possibility to rearrange the modes in any architecture (number of sub-groups, number of modes by sub-group, height of the nested architecture, etc.). At each level, the modes are proposed to the user, one-by-one, and the user activates the switch when the mode they want is proposed. Although this strategy is quite robust (due to its simplicity) and thus commonly found in practice [19], it is time consuming and does not lead to satisfactory user experience [15]. In the past few years, many attempts (many of them successful) have been made to try to do

better. Among them, general ideas emerged such as level coding [11], dynamic switching, and even automatic mode switching. Level coding can work well with analog sensors, by dividing the range of the signals into multiple sub-ranges bounded by thresholds. Hence, what seemed to be a one-dimension interface (e.g., ON/OFF switch) actually has a higher dimensionality (e.g., sensing the pressure on the switch). The thresholds can also be applied on the time axis (short click, long click). Of course, the divisibility of the signal's magnitude depends on the user's dexterity and the sensor. Dynamic switching, on the other hand, integrates machine learning models to predict or suggest the next move to the user [20]. So, when comes the time to select a mode, this algorithm might suggest one of a few modes that the user is likely to select according to his habits. Much like the auto-completion suggestions on a smartphone keyboard, this can save the user a lot of time. On the other hand, training a classifier may require much data, and in the case of rehabilitation technology, this can be difficult. Indeed, to deploy such an interface, the users cannot be asked to build a large training dataset themselves. Thus, the development team should have access to such a dataset and make sure that the data reflect real-life situations (e.g., noise, imperfect inputs) which might prove difficult. Finally, another strategy is to focus on a complete trajectory to execute an action (e.g., pour a glass of water with a robotic arm). In this case, the algorithm is based on graph theory rather than machine learning [4, 21]. The aim is to find optimal trajectories for a given action using graph search algorithms such as Dijkstra's algorithm. This paradigm is robust for specific actions in closed environments, but the resulting interfaces may lack flexibility when it comes to real-life scenarios.

Recently, [18] proposed a novel approach to deal with low-dimensionality interfaces with a sequence matching algorithm (SMA). The idea is to arrange simple signals received from digital sensors (e.g. ON/OFF switch) into a time sequence similar to Morse code. The algorithm takes advantage of simple devices such as switches, which are already vastly used by people living with disabilities but to artificially increase the number of output states it can generate. While a single switch can only output two states (0 or 1), a switch with SMA can generate many outputs (e.g., 30), which enables the user to express more commands rapidly with a single switch. Instead of grouping the modes, a unique sequence is inputted by the user. Each sequence is mapped to a command (mode) of the AT (e.g., forward). This algorithm was applied to a sip-and-puff interface to control a robotic arm and has proven to be effective to augment the dimensionality of currently used basic interfaces, thus enhancing the user's control over the assistive device [18].

The objective of this paper is to build on the SMA in [18] by adapting the algorithm to analog signals rather than digital signals to further increase the output dimensionality, and to enable the use of analog sensors to lead to more intuitive interfaces. The proof of concept consists in the development, design and implementation of an intuitive and robust IMU-based hand gesture interface to control any assistive technology (e.g., robotic arm, mouse and

keyboard). To that effect, a classification pipeline has been developed to detect and recognize hand gestures from IMU signals. This pipeline was then embedded into the SMA. With that system, we aim to answer the previously stated issue of dimensionality difference between the interface and the assistive technology without relying on grouping the different modes, but rather by artificially augmenting the dimensionality of the interface itself. The goal is to help people living with upper or lower body incapacities in their daily life tasks.

This paper is structured as follows. First, the methods for the development of the gesture recognition algorithm are presented. This includes an offline version, in section 2.4, followed by a real-time version in section 2.5, and the integration of the latter to the SMA in section 2.6. Sections 2.7 and 2.8 describe the practical implementation of the interface and the experiments performed with it. Finally, the results are discussed in section 2.9.

2.4 Hand Gesture Recognition

The process of classifying hand gestures can be done with a trained machine learning model (classifier). Since artificial intelligence is now a blooming field of computer science and robotics, some classifiers have proven effective for specific applications. In the case of accelerometer and gyroscope signals for gesture recognition, the complexity of the models can go from a linear discriminant analysis (LDA) [12, 22] to a convolution neural network (CNN)[23].

Many case studies for the gesture-recognition problem combine surface EMG sensors with IMUs. In [24], LDA is proven to be accurate enough to classify signals from a wrist-worn sensor device. A smartwatch could eventually house the sensors and the lightweight classifier and be very convenient for the user. In [25], an SVM-based recognition system is used and applied to rehabilitation purposes. Other simple yet effective classification models like Bayesian models can also be used [26]. However, in the framework of this project, we aim to make use of only one IMU and focus on gestures that will take advantage of the accelerometers such as tapping on a hard surface or making a circle in the air rather than doing hand signs. Dynamic time warping is well known and often used with IMUs to classify hand gestures [27, 28, 29], but may not be suitable for a real-time application as it may not be fast enough [30].

Since the interface targets to users living with various incapacities, the classification models have to be fitted on each user's sample data individually. Therefore, the training dataset has to be small. Consequently, we will focus on simpler algorithms which are fast, robust and usable on an embedded system. Other than LDA, available options include support vector machines (SVM)[25], adaptive boosting (AdaBoost)[31] and k-nearest neighbors (kNN)[32].

The objective of this section is to propose an encapsulated procedure that takes a signal as an input, and returns a predicted gesture. In the particular case of the experiments in this paper, the signals are received from an IMU, and the hand gestures will serve as individual

actions to the SMA. This section presents the first of a twofold development. The first part is the offline development and the second part is the online implementation. For the offline development, the dataset is composed of 475 examples of 5 different gestures recorded at 100 Hz for 0.9 seconds from a single individual.

The classification pipeline includes multiple nodes. As seen in Figure 2.2, features must be extracted from an input signal. Then, the classification model must determine which class (gesture) those features are most likely to represent. Since it is established that the training dataset will have more dimensions than training examples, the dimensionality of the data (i.e., the number of features) must be reduced before they are presented to the classifier. The model has to be fast during evaluation, lightweight (for an implementation on an embedded system) and robust, even with a very small training dataset. The next subsections bring details on the development process and the final pipeline.

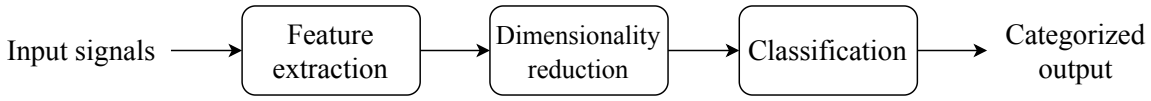


FIGURE 2.2 – Symbolic diagram of the classification process (pipeline).

2.4.1 Input Signals

In the experiments, the algorithm's inputs are obtained through an IMU and consist of a first-order low-pass filtered version of the acceleration norm ($a = \left\| \sqrt{a_x^2 + a_y^2 + a_z^2} \right\|$), and angular velocity norm ($\omega = \left\| \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \right\|$).

For the offline development of the classifier, five distinct hand gestures were studied: *hand tap* (on a table), *thumb tap* (on a table), *hand tilt*, *swipe left* and *swipe right*. The signals were chosen because they include completely distinct pairs (e.g., *hand tap* and *hand tilt*) as well as similar signals (e.g., *swipe left* and *swipe right*) that a simple conditional statement cannot classify.

The training set consists of five examples per class (25 examples total). The remaining of the balanced 475-examples dataset will be used to tune the models and evaluate the performance.

2.4.2 Feature Extraction

The examples in the dataset are just time series and cannot be inputted to the classifier in their raw format. From those time series' data, various metrics must be computed to "describe" their nature to the statistical model. This process is known as *feature extraction*. Identifying what features to extract from a signal first requires having a general idea of what it looks like. From Figure 2.3, it is possible to observe that the means, medians, standard deviations,

interquartile range, area under the curve, kurtosis, minima and maxima of accelerations and angular velocities are simple yet useful metrics that allow the classification of gestures. Those metrics can be computed for the entire signal but also for parts of it. Indeed, in the Figure 2.3 gyroscope signals, the two main peaks are supposed to have a different time lapse between them, so by dividing the signals into two equal parts, the peaks will be distinctly distributed among the parts. Thus, the previously stated metrics applied to the signal's parts are distinct

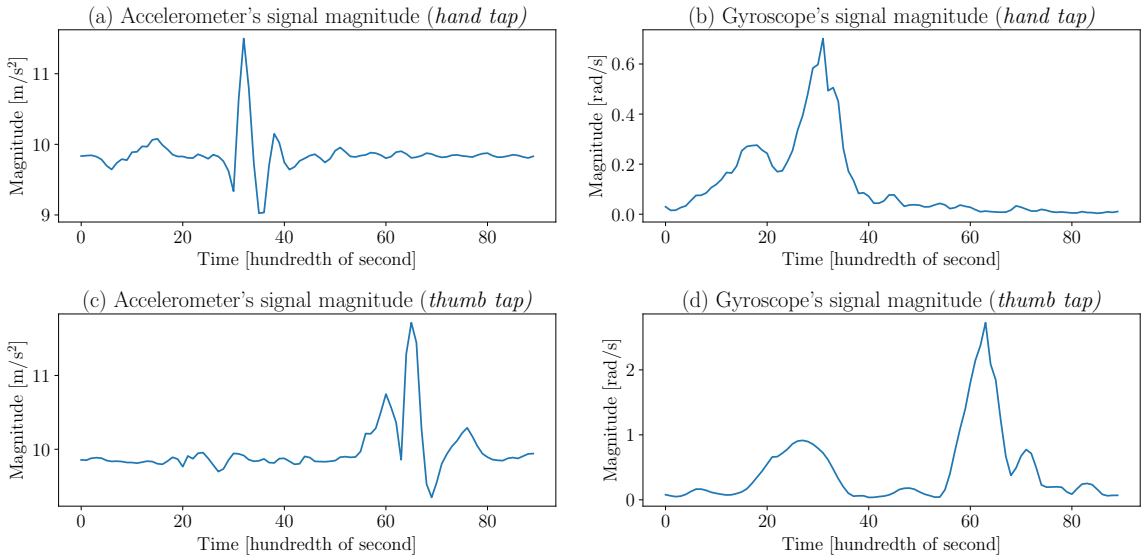


FIGURE 2.3 – Input signal examples for (a) a *hand tap* acceleration, (b) a *hand tap* angular velocity, (c) a *thumb tap* acceleration and (d) a *thumb tap* angular velocity.

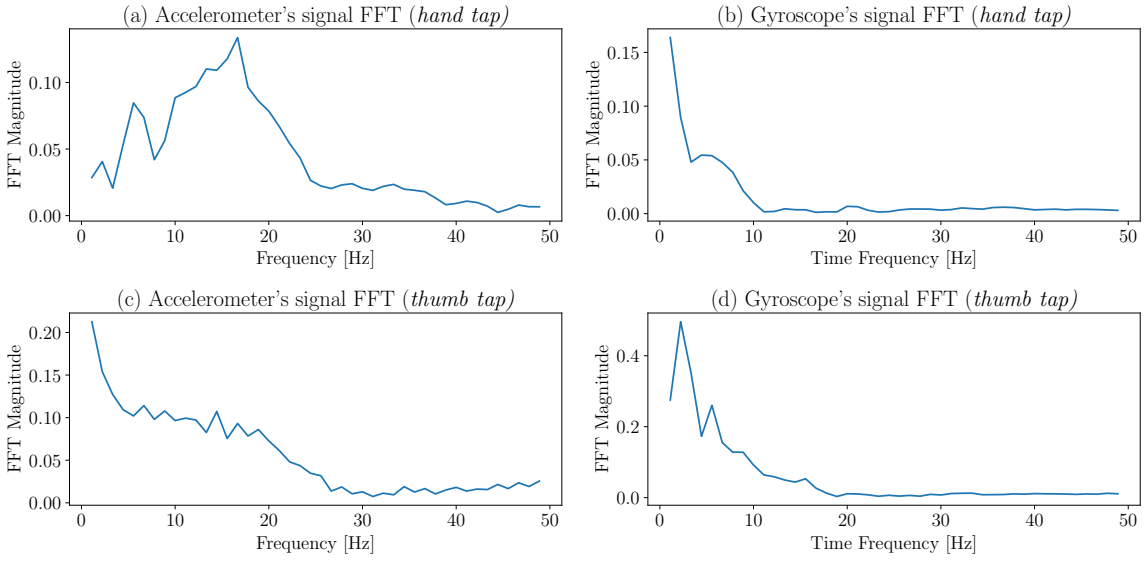


FIGURE 2.4 – Fast Fourier transform examples for (a) a *hand tap* acceleration, (b) a *hand tap* angular velocity, (c) a *thumb tap* acceleration and (d) a *thumb tap* angular velocity.

and offer valuable information to the classifier. It is determined empirically that dividing the signals into three parts offers the best results.

In addition to the metrics in the time domain, others can be extracted from the frequency domain. Figure 2.4 shows a fast Fourier transform of the Figure 2.3 time signals. Even though those curves are not the clearest (due to the 100 Hz sample rate) they still offer clear patterns and the median, maxima and energy values of the FFTs are valuable for the classifier.

The final metrics extracted from the signals are the coefficients of a third-order autoregressive model [33, 34]. Finally, a total of 81 features were extracted, and all of them are scaled in a $[-1, 1]$ range.

2.4.3 Dimensionality Reduction

Let X be the n by d matrix containing all the feature data extracted from the train set, where n is the sample size (the number of examples) and d is the dimension (the number of features),

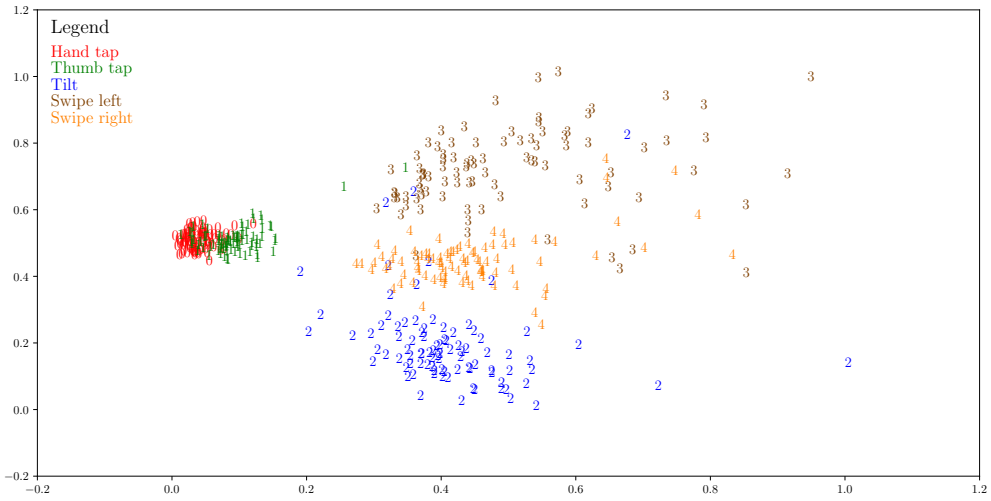
$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ x_{2,1} & x_{2,2} & \dots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,d} \end{bmatrix}. \tag{2.1}$$

We have

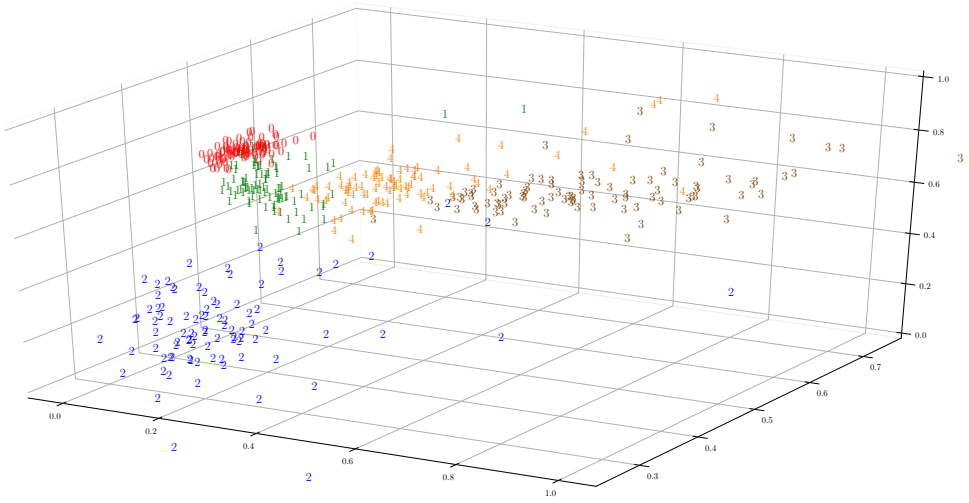
$$d = 81 \tag{2.2}$$

$$n = 25, \tag{2.3}$$

since $n < d$, the rows of X (the examples) are collinear. This means that X contains redundant information and unnecessarily increases the dimensionality d of the dataset. So, most of the dimensions are not seeded to describe the dataset. Furthermore, n is typically small, so the dimensionality must be reduced to avoid issues related to the curse of dimensionality [35] and maintain the robustness of the system. In order to do so, the approaches would be to either select the most significant features or to compute new features via the linear projection of $X_{25\times 81}$ to $X^*_{25\times d^*}$, with $d^* \leq n$, the dimension of the new space. Here, the latter has been employed. The principal component analysis (PCA) offers a deterministic method to compute projected features from X in a lower-dimension space. The PCA prevents losing too much information while reducing d to d^*. This fact is illustrated in Figure 2.5. The graphs on the figure are obtained by first applying the PCA to the training set (25 data points, perfectly balanced in five classes). This computes the transformation to apply to what is left of the dataset (the validation set, 450 data points). Thus, the figure shows the distribution of the validation set in the two- and three-dimensional spaces after the transformation is applied. It



(a)



(b)

FIGURE 2.5 – PCA's linear projection of the dataset in (a) two dimensions and (b) three dimensions. The plots' axes represent the projected features.

is possible to observe that even with $d^* = 2$ or $d^* = 3$, and with the transformation computed on a very small dataset, the classes are still almost linearly separable making it clear that most of the relevant information is still present in X^*. In order to implement the PCA, the target dimension must not be more than n. In the context of this application, it has been observed that $d^* = n$ presents a good balance between loss of information and low dimensionality.

2.4.4 Classification

The selection of the classification model is the part of the design where the principal design requirements (fast, lightweight and robust with fewer examples) should really be taken into consideration. For the development and in the framework of this paper, *scikit-learn* [36] was used.

In recent years, deep learning algorithms have received a lot of attention. But one caveat is in order: they require vast amount of data. In our case, the fact that the quantity of data required to train such an algorithm is limited rules them out due to the specificity of the applications. Therefore, the selection process focuses on more classic machine learning algorithms.

TABLE 2.1 – Studied machine learning classifiers with their advantages.

Classifier	**Advantages**
Support vector machines (SVM)	Limits overfitting, robust
Linear discriminant analysis (LDA)	Fast training, low memory usage, very easy to implement
Adaptative boosting (AdaBoost)	Fast, works well on smaller datasets
K-nearest neighbors (KNN)	Fast training, very easy to implement

The studied models for this project are listed in Table 2.1 along with their advantages specific to the design requirements. In order to test each model and to optimize them at the same time, the selection process consists in a grid search with random data selection. Each model is tested on a dataset of 475 data points (95 per class, 5 classes) on i_{max} iterations. The models are tested by first tuning their hyper-parameters and then retaining their best performance. For each model, a set of hyper-parameters is determined with a testing range associated with each one. At each iteration $i = 0, 1, \ldots, i_{max} - 1$, five data points of every class are selected (total of 25) to form the training set, and the remaining data constitutes the validation set (450 data points). Each model is trained with this training set and tested with the validation set once for every possible arrangement of hyper-parameters. After the i_{max} iterations the performances are analyzed and the best model with the best arrangement of hyper-parameters is retained. Having the size of the validation test far greater than the size of the training set allows evaluating how the algorithm succeeds in generalizing and not to overfitting on the small set of data it has been given. Table 2.2 presents the selected hyper-parameters for each model, the best configuration and the best prediction score of the models associated with those configurations for $i_{max} = 10$.

The final choice for the prediction model is an SVM with a linear kernel. The satisfactory performance of the linear SVM implies that the data is almost linearly separable in the $d-$dimensions space even if similar gestures were chosen. This means that it should not be difficult to choose new gestures to add to the system. Figure 2.6a shows the normalized confusion

matrices for the optimized SVM with the dataset previously described.

TABLE 2.2 – Evaluations of the machine learning models in function of their hyper-parameters' tuning. The value $C > 0$ is a regularization parameter, ϕ is the kernel function of the SVM (can be linear, gaussian, polynomial or sigmoid). The solver of the LDA can be a singular value decomposition, least-square solution or eigenvalue decomposition. The AdaBoost algorithm uses decision trees as weak learners where $n_{estimators}$ is the number of trees, and max_{depth} is the depth of those trees. $k_{neighbors}$ is the number of points to compare for the KNN algorithm and $weights$ is the weighting of each neighbor and can be uniform or a function of the distances.

Classifier	**Hyper-parameters**	**Best Configuration**	**Best Score**
SVM	(C, ϕ)	$(0.1, \text{linear})$	95%
LDA	$(solver)$	(least square)	85%
AdaBoost	$(n_{estimators}, max_{depth})$	$(25, 100)$	85%
KNN	$(k_{neighbors}, weights)$	$(2, \text{distance})$	92%

Such matrices are obtained by choosing a training set at random (composed of 5 data points per class), similar to the optimization process, and by validating with the remaining 450 data points. This process is repeated 50 times. Finally, the results are inputted into the confusion matrix and normalized by row. Since most of the confusion results from the *Swipe-left* and *Swipe-right* gestures, it is possible to improve the performance of the system by simply removing the gestures that are too similar (and possibly add some other ones). The result of this operation is shown in the matrix (b) of Figure 2.6. Referring to Table 2.3, four different gestures with three gestures per sequence allows 84 possible sequences which is fairly sufficient to control an assistive technology.

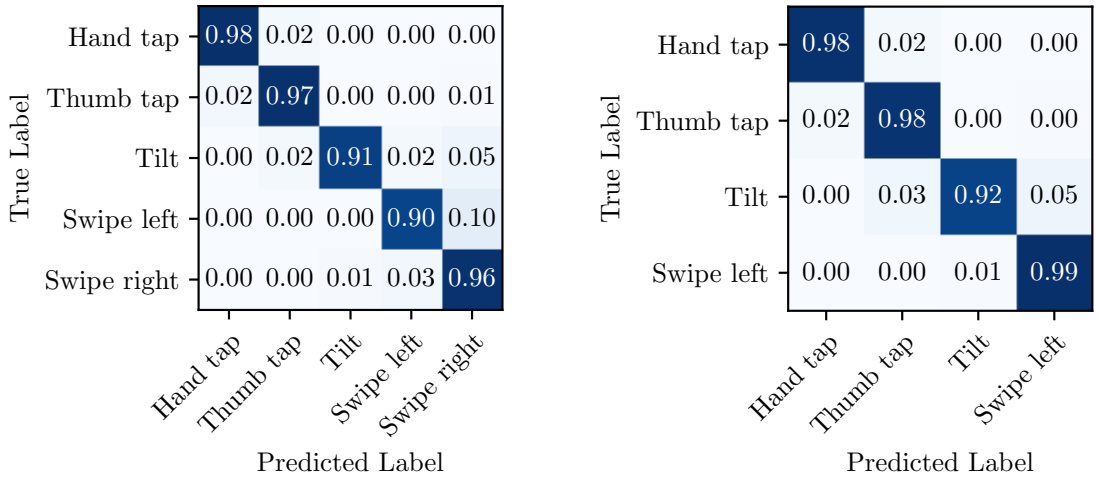


FIGURE 2.6 – Normalized confusion matrices resulting from the validation of the optimized SVM classifier for (a) five different gestures and (b) four different gestures.

2.4.5 Final Pipeline

The final classification pipeline is the prediction system that takes IMU signals as inputs and returns predicted gestures. This section establishes its components. Figure 2.7 shows the explicit diagram of the classification process.

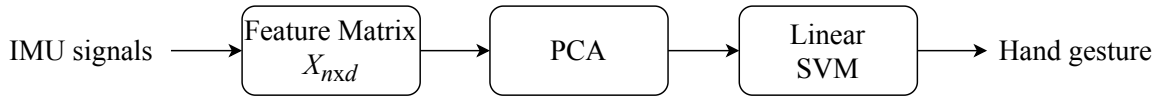


FIGURE 2.7 – Final diagram of the classification process (pipeline).

2.5 Real-Time Classification

The previous section proposed an offline machine learning model to recognize hand gestures from small fixed windows of IMU signals. To embed the Figure 2.7 classification pipeline in a real-time application, it first needs to be combined with an algorithm that will trigger the classification process. There are (at least) two options to do that. The first option is to have a fixed-size sliding window, and repeated classification routine at short fixed intervals on this window. When tested, this method performed weakly due to multiple reasons. First, different gestures take different lengths of time to complete. Even if the gestures are supposed to be the same, they can be executed at different rates. Also, to prevent the classifier from predicting the same given gesture multiple times over the fixed sliding window, or to make an early (and incorrect) prediction, there has to be an algorithm that determines the moment at which the gesture begins and ends (the solution could also be to increase the time lapse between the intervals of classifications, but this would greatly increase the risk of missing significant windows of signals).

The other option is to have a sliding window over the signal, and to use this window to determine the moment at which the gesture starts and ends. Only then, the classification process is triggered and a gesture is predicted from variable-length inputs. Indeed, none of the components in Figure 2.7 actually requires the input to always be of the same length. This procedure settles the previously stated issues (duplication of the prediction, variable length of the gestures) and allows the execution of sequences of gestures in a fluid and efficient manner. Algorithm 1 presents the definition of DETECTACTION, where the starting and ending points of the gestures are determined from the derivatives and the variances of the input signals.

The function takes r, a sensor reading, as an input. In the context of the experiments, r has two components, the accelerometer's magnitude signal and the gyroscope's magnitude signal. To generalize, this is not represented in the pseudo-code, but *window* would be a $2 \times l_w$ matrix

Algorithm 1 Definition of the Real Time Gesture Recognition Sub-routine.

function DETECTACTION(`SensorReading` r)
  ▷ `length`(r) should be 1
  ▷ $start = end =$ `NULL` for the first function call
  $window \leftarrow$ `concatenate`$(window[$`length`$(r)...], r)$ ▷ slide $window$
  if $\left|\frac{d}{dt}\left[window\left[-\delta\ldots\right]\right]\right| > threshold_d$ `AND` $\mathrm{Var}\left[window\left[-\delta\ldots\right]\right] > threshold_{var}$ **then**
    if $start ==$ `NULL` **then**
      $start \leftarrow$ `length`$(window) - \delta$
      $end \leftarrow$ `NULL`
    else
      $start \leftarrow start -$ `length`(r) ▷ slide with $window$
    end if
  else
    $end \leftarrow$ `length`$(window) - 1$
    if $start$ `NOT NULL` **then**
      $action =$ PREDICT$(window[start\ldots end])$
      $start \leftarrow$ `NULL`
    end if
  end if
  return $action$
end function

where l_w is the length of the window and each row represents one component. So the derivatives and the variances are computed from the rows of $window$. $threshold_d$ and $threshold_{var}$ are arbitrarily determined thresholds on the derivative and variance values respectively. Those thresholds represent the sensitivity of the algorithm to trigger the classification process. They can be modified manually for each user. Also, δ is an arbitrary offset on the sliding window to compensate the lag associated with the computation of the derivatives and variances on the tail of $window$ ($window[-\delta\ldots]$, which represents the latest δ points of the signals). This offset allows the algorithm to have a slightly larger window with a few stable points before and after the signals pass the thresholds. It ensures that all significant data is given to the classification pipeline. Finally, the PREDICT function is the implementation of the pipeline.

2.6 Sequence-Matching Algorithm Review

For this project, the point in developing a real-time gesture recognition algorithm is that it can be integrated into a more general algorithm that aims to help the control over higher dimensionality ATs. Indeed, the previous sections established a machine learning classifier based on five different inputs. Choosing to implement it directly in order for it to interface with an AT could result in a usable system, but there would only be five distinct physical commands. Instead, the classifier is embedded in a sequence matching algorithm to artificially increase the dimensionality of the interface. This section presents this sequence matching algorithm.

2.6.1 General Idea

The main objective of the SMA is to allow the user to gain degrees of control without increasing the number of actual physical controls (e.g., buttons, gestures, sensors, etc.). Figure 2.8 presents a flowchart of the general idea of the algorithm. First, individual actions (gestures)

FIGURE 2.8 – Simplified representation of the general ideal behind the sequence matching algorithm.

are detected. They are stacked into a sequence. Then, the algorithm recognizes in real time the sequence of actions and seeks to match it with a sequence among a set of sequences defined by the user. This set of user-defined sequences maps to a set of modes of an assistive technology (each sequence corresponds to a mode). Finally, the user can execute the commands that the mode offers. In the context of this project, hand gestures are considered individual actions, so the user-defined sequences are short series of simple gestures. What follows is the mathematical formulation of the SMA using the previously developed gesture recognition with the five possible inputs. Let A be the set of all possible actions, then

$$A = \{0, 1, 2, 3, 4\} \tag{2.4}$$

where the value of an integer $a \in A$ is,

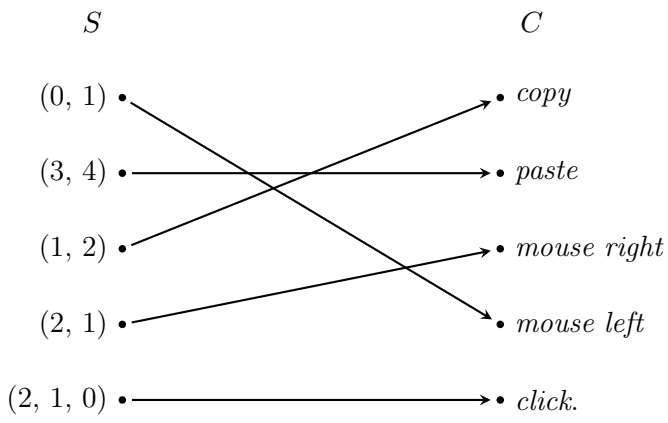
$$a = \begin{cases} 0, & \text{if hand tap} \\ 1, & \text{if thump tap} \\ 2, & \text{if tilt} \\ 3, & \text{if swipe left} \\ 4, & \text{if swipe right} \end{cases} \tag{2.5}$$

Let s be the ordered sequence of individual actions performed by the user and S the set possible sequences defined by the user. Finally, let C be the set of different modes for the robotic arm and R the relation that maps S to C. This relation, or rather the set S, is dynamic and will narrow itself as the user inputs actions and s is built. For instance, if a user wants to control five modes on a computer, then arbitrarily,

$$C = \{copy, paste, mouse\ right, mouse\ left, click\}, \tag{2.6}$$

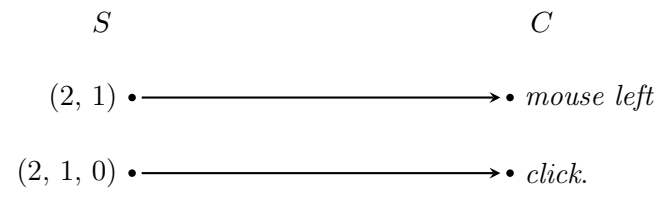
$$S = \{(0,1),(3,4),(1,2),(2,1),(2,1,0)\} \tag{2.7}$$

and the relation R with the initial S and C is represented by the following diagram,



(2.8)

If the user makes a *tilt*, the algorithm will detect it and set $s = (2)$. Simultaneously, the algorithm sorts out impossible sequences and the (2.8) becomes,



(2.9)

If, right after that, the user makes a *thumb tap*, then $s = (2,1)$, but the relation in (2.9) does not change since both sequences of S begin with s. At this point, the user must wait one second if he or she wants to enter the *mouse left* mode or make a *hand tap* (i.e. $a = -1$) for the *fingers* mode to be automatically selected.

In this example, C only has five elements, but in general, if the user chooses to have at most k actions in each sequence, then the cardinality of S

$$|S| = \sum_{i=1}^{k} (|A|)^i \tag{2.10}$$

with $|A|$ being the cardinality of A (i.e., the number of possible actions). If R is a bijection, then $|C| = |S|$. In practice, it is observed that $k = 3$ is suitable. Greater than that, the sequences become harder to execute correctly. Table 2.3 shows the values of $|S|$ in function of $|A|$ and k obtained with equation 2.10.

TABLE 2.3 – Number of possible sequences with respect to the number of distinct actions $|A|$ and the chosen maximum number of actions per sequence k.

	$k=1$	$k=2$	$k=3$
$\|A\|=2$	2	6	14
$\|A\|=3$	3	12	39
$\|A\|=4$	4	20	84
$\|A\|=5$	5	30	155

The last row of Table 2.3 illustrates the example above. This means that it would theoretically be possible to control a technology of 155 modes with a five-dimension interface. Of course, in practice, this number drops because the interface would become cumbersome with a number of controls too high, but still, $|S| \approx 15$ is feasible.

Algorithm 2 General Sequence Matching Algorithm

```
1:  function SequenceMatchingLoop
2:      t_action ← start timer
3:      loop
4:          s ← ()
5:          sequenceMatch ← False
6:          while NOT sequenceMatch do
7:              r ← sensor reading
8:              a ← DetectAction(r)
9:              if a ≠ NULL then
10:                 append a to s
11:                 reset(t_action)
12:             end if
13:             (sequenceMatch, s) ← CheckCurrentSequence(s, t_action)
14:         end while
15:         t_exec ← start timer
16:         S ← Initial State
17:         c ← R(s)
18:         r ← sensor reading
19:         while r ≠ 0 OR t_exec NOT timeout do
20:             if r ≠ 0 then
21:                 reset(t_exec)
22:                 execute(c)
23:             end if
24:         end while
25:     end loop
26: end function
```

2.6.2 Pseudo-code for the SMA

Now that the mathematical definitions have been established, the SMA can be presented in a more pragmatic form. The scheme written in Algorithm 2 presents the general algorithm

of the SMA. It requires S, C and R, as defined in the previous subsections. A look at the algorithm quickly reveals the five principal parts presented in Figure 2.8. The *Detect* step is conducted by the DetectAction subroutine. The *Recognize* and the *Match* steps are processed inside the CheckCurrentSequence subroutine, the function to *Map* is R. Finally, *Execute* represents the execution of the command c (`execute`(c)) when found. Algorithm 3 details the procedure to ensure that the correct sequence is matched as fast as possible to a sequence in the library (S). Given the current sequence inputted by the user, it returns a boolean value ($sequenceMatch$) indicating whether or not the current sequence s is an element of the set S. The function also returns the current sequence itself, which may have been emptied if there was no possibility that it might lead to one of the sequences of S.

The DetectAction subroutine is defined in Algorithm 1. It takes a sensor's signal as an input and returns an integer a indicating the detected action. If this action is not in A (i.e., no action performed by the user), a will be given a zero value.

Algorithm 3 Dynamic Matching Sub-routine of the SMA

```
1:  function CheckCurrentSequence(Sequence s, Timer t)
2:      if t NOT timeout then
3:          n_S ← length(S)
4:          for element in S do
5:              if element[0 : n_s] ≠ s then
6:                  S.remove(element)
7:              end if
8:          end for
9:          if n_S == 0 then
10:             return False, ()
11:         else if n_S == 1 then
12:             sequenceMatch ← (s ∈ S)
13:             return sequenceMatch, s
14:         else
15:             return False, s
16:         end if
17:     else
18:         sequenceMatch ← (s ∈ S)
19:         return sequenceMatch, s
20:     end if
21: end function                    ▷ boolean sequenceMatch, Sequence s
```

2.7 Experimental Implementation

Section 2.3 identified three parts in human-machine interactions: the user, the interface and the device This section will describe how the hand-gesture recognition is embedded in the SMA in real-time as well as an actual implementation. This system allows a user wearing an IMU to control a computer mouse and keyboard.

2.7.1 System Design

This subsection will elaborate on the entire system design, choices of technology and implementation. In order to maintain the modularity of the system, the latter has been divided into four parts. The first part manages the sensors (IMU), the second part is the actual interface, the third part manages the controlled device (mouse and keyboard) and finally, those three elements are brought together by a local server on the PC that manages the information exchange among them. Figure 2.9 presents this architecture.

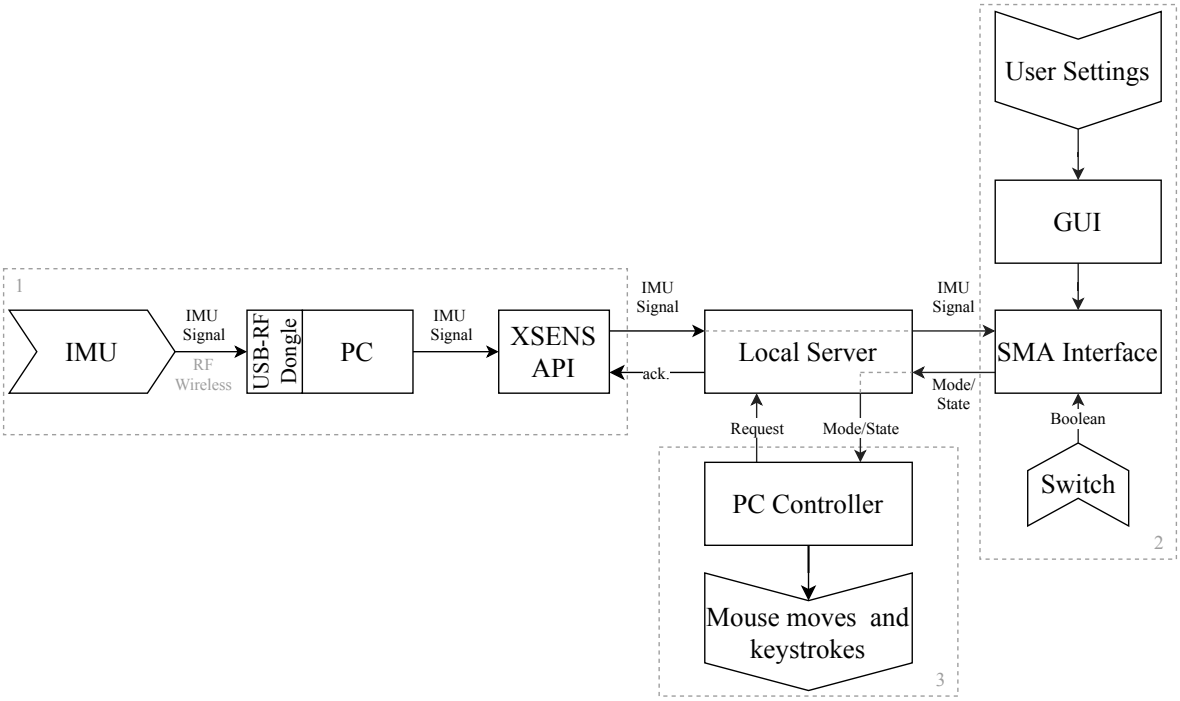


FIGURE 2.9 – System architecture for implementing an SMA interface for the control of a computer mouse and keyboard. The dotted boxes represent the modular parts of the system. The chevron-shaped boxes represent the inputs and output of the system.

Sensors

The sensors used in the implementation are the XSENS MTw Awinda wireless IMU. The data is read at 100 Hz and sent to the server.

Interface

The second piece of the system is the actual implementation of the SMA interface with the addition of a graphical user interface (GUI) that allows the user to set the system up. Through this GUI, the user can record, edit or delete gestures (defining A), modes (defining C) or sequences, and map the sequences to the modes of the controlled AT (defining R). The raw IMU signals dispatched by the local server are processed according to Algorithm 2. This

module takes a third input which is a boolean value coming from a switch. The switch is simply an external button and is used to send the signal to execute a mode. It also acts as a safety feature since nothing can be executed unless the switch is pressed. Indeed, if the AT were a robotic arm, this switch would be essential. It can be disabled for certain static modes (e.g., click, copy, paste) if the user wishes to automatically execute when the sequence is executed. The module then returns the selected mode and switch state to the server.

Controlled Device

In this application the controlled devices are the mouse and keyboard of a computer, so the modes are keyboard shortcuts (e.g., CTRL+C, CTRL+V, ATL+TAB, Arrow keys, etc.), mouse direction and mouse clicks. A simple mouse-and-keyboard controller has been developed. It takes as inputs a mode and the state of the switch. This third module manages the final output of the system.

Local Server

Although the first three parts of the system would work independently, they would not offer much functionality without a central piece to bring them together. The local server is a TCP server socket, and the three previous modules are client sockets. The server is programmed so that at each reception of data, it sends a response to the client. The response differs according to the sender. This way, every client-server communication is a synchronous exchange of information, but the clients can be asynchronous from one another. So, if one part of the system crashes, it will not stop the others. Only the failing part will need to restart, reconnect to the server and resume where the other parts are.

The server also allows the components to be linked easily even if they are not written in the same language. For example, the API for the sensor and the controlled device could be written in C++ and the interface in Python. In a final production environment, the modules can be merged into a single application, but the server's presence for the development enhances the versatility and fluidity of the development.

2.7.2 Usage

The user's experience with the system starts with the graphical interface. The GUI allows the user to set up the program and helps them familiarize themselves with the operation. The user's first steps with the interface (with the GUI) are as follows:

1. Record a first gesture (i.e., execute the gesture five times to train the algorithm),
2. Define a first sequence,
3. Map the sequence with a mode of the computer's keyboard or mouse.

After these three steps, the user is ready to control the selected modes of the computer. To do so they:

1. Perform the sequence associated with the desired mode,
2. When in the mode, use the switch to execute the mode (e.g., mode: mouse-up implies that the cursor will go up on activation of the switch),
3. When done, wait two seconds. The mode is left,
4. Perform a new sequence.

With the graphical interface, at any time, the user can also:

— Record or delete gestures,

— Create, edit or delete sequences,

— Edit the mapping between the sequences and the modes,

— Adjust the system sensitivity (the thresholds that trigger the gesture recognition algorithm as discussed in Section 2.5),

— View the current mapping,

— Have visual feedback of the current sequence and the individual actions performed.

Not having a large dataset has many disadvantages, but it allows the algorithm to train very quickly. This advantage is used every time the gestures or sequences change. When new gestures are added or removed, the recognition model is trained again in a matter of seconds. Also, the algorithm verifies which recorded gestures are used by the mapped sequences, and only trains or re-trains the model on relevant gestures. This allows the classifier's training to be optimal on the significant data.

2.7.3 Availability

The software developed for this project is open-source. It is available at [37].

2.8 Experiments

Following the development and the implementation presented in the previous sections, a series of tasks involving the control of a computer mouse and keyboard has been designed to test the system in a real-life scenario. The hand gesture interface was compared against and combined with the AssystMouse [38], a head-movement-based software to control the mouse using a webcam. The first objective of the tests was to assess the performance of the hand gesture interface by first comparing it to the AssystMouse software. Then, the tests aimed to evaluate the usage of the hand gesture interface when combined with the AssystMouse software. In this case, AssystMouse was used to control the mouse and the hand gestures provided quick keyboard shortcuts (e.g., *Tap* to copy and *tap-tap* to paste). The experimental setup is presented in Figure 2.10. It includes an IMU to input gestures, a webcam required by

the AssystMouse, and display monitors showing the graphical interface (displaying the modes, sequences and options) and the actual workspace where the tasks are performed.

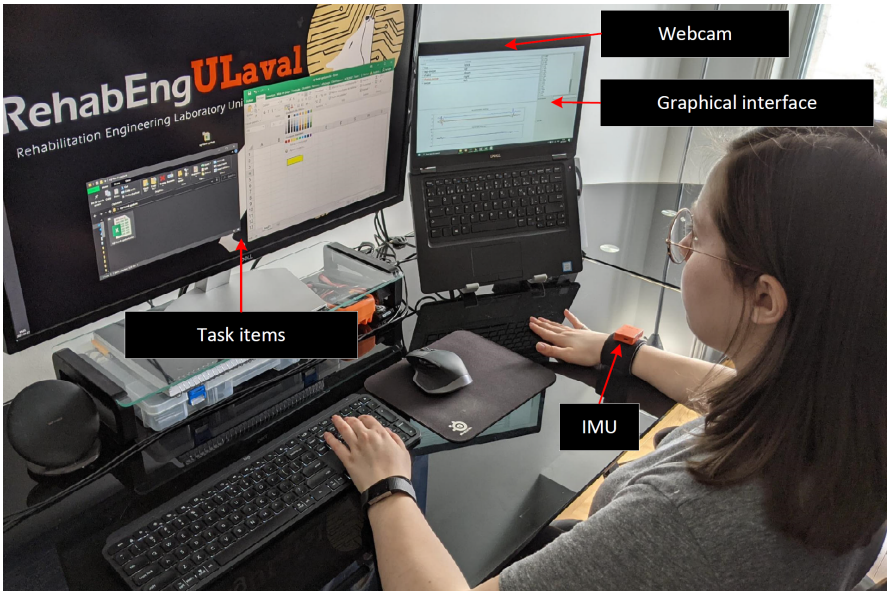


FIGURE 2.10 – Experimental setup for the hand gesture interface evaluation.

The experiments have been performed by five healthy participants (2 male, 3 female) between 24 and 35 years old. In order to assess the performance of the hand gesture interface, they had to execute two tasks. To set up the experiment, each participant had to record his or her own gestures, then build sequences and map them to specific control modes of the mouse and keyboard. They first recorded three distinct hand gestures:

1. *Tap*: a simple tap on the table,
2. *Swipe*: a swipe on or above the table,
3. *Shake*: wrist shake in the air.

To do so, the participants had to execute each gesture ten times.

The first task, shown in Figure 2.11, was to open a folder on the PC desktop, then open the spreadsheet in the folder and change the color of the yellow cell to red. This task required the participant to be precise with the mouse as well as to execute single and double clicks. The participants had to complete this task using first the hand gesture interface and then the AssystMouse and they were timed for both interfaces. With the gestures, five sequences were created : *Tap*, *Tap-Swipe*,*Shake*, *Shake-Swipe* and *Swipe*. Those sequences were mapped to the mouse modes left-click, up, down, left and right. For this task, a button was used to activate the mode once in it. For instance, if the participant executed *Tap-Swipe* to enter the mouse-up mode, they had to press the button to actually make the cursor go up. Before the experiment, the participants had ten minutes to familiarize themselves with each interface.

The second task was performed in the spreadsheet. The participants first had to copy and paste a cell three times with the AssystMouse only. Then, they had to do the same thing but using both the Assystmouse and the hand gesture interface. The first three sequences were remapped to the modes click, copy and paste. Thus, the gestures could be used as shortcuts and the mouse was still controlled with the AssystMouse. This aimed to evaluate the suitability of the hand gesture interface when used as a complement to other available AT control systems.

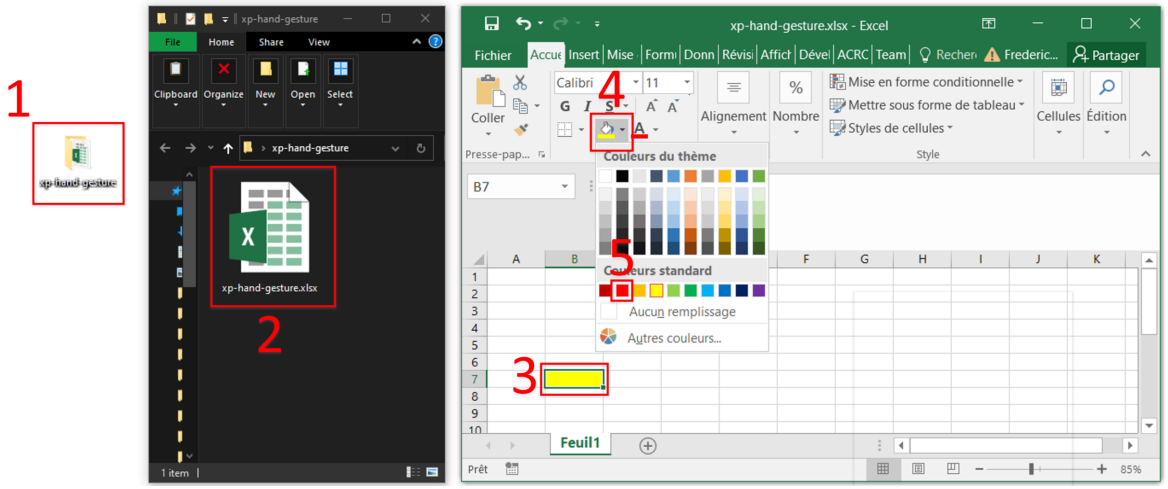


FIGURE 2.11 – Five steps to execute with both the hand gesture interface and the AssystMouse to complete the first task.

After the tests, the participants were asked to answer a questionnaire, which contained questions adapted from the QUEAD [39], and to leave their comments.

2.9 Results and Discussion

For each task and each interface presented in the previous section, the completion times were noted. The distributions of this data is shown by the plots in Figure 2.12.

Figure 2.12 presents the data by task (mouse control and copy-paste) and by interface (hand gesture and AssystMouse) shown in a boxplot. The boxplots are wrapped in a violin-like area that shows the estimated probability densities. The means of the completion times plus or minus the standard deviations are also displayed in the figure.

The distributions show that a webcam-based system for the control of the mouse is still much faster. However, the standard deviation of the times for the hand gesture mouse control task being relatively low indicates that the system is reliable and robust since all the participants were able to complete the task in similar times. In other words, it does not seem to require any specific skills to work.

On the other hand, the copy-paste task's data shows that the integration of the hand gesture interface with the AssystMouse consistently improved the participants' time by 77% on average. In addition, a one-tailed, non-parametric Wilcoxon signed-rank test also confirmed that the usage of the hand gesture interface significantly decreases the time to perform the second task ($p - value = 0.043 < 0.050$).

Another metric to look at for the experiments is the setup time for the first usage of the hand gesture interface. In general, the average time to record the three gestures was 99 seconds, and the time to set up the five sequences and to map them to the five modes was about 69 seconds. These actions are only executed the first time that a participant uses the interface. The participant's data is then kept in memory. Therefore, it might be cumbersome to use the hand gesture solution the first time. However, because these actions are only required once at the installation of the hand gesture interface and that it is kept in memory from day to day, the payback increases rapidly with time.

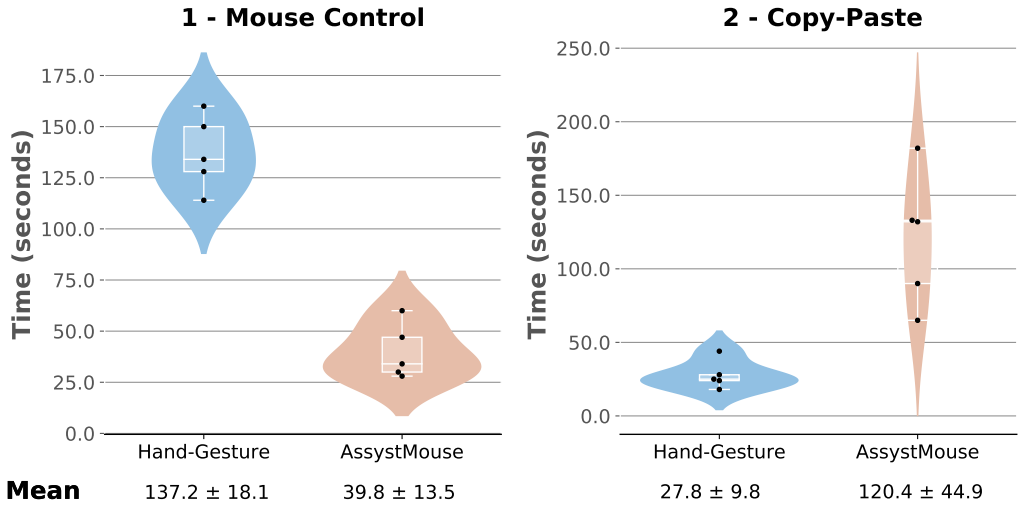
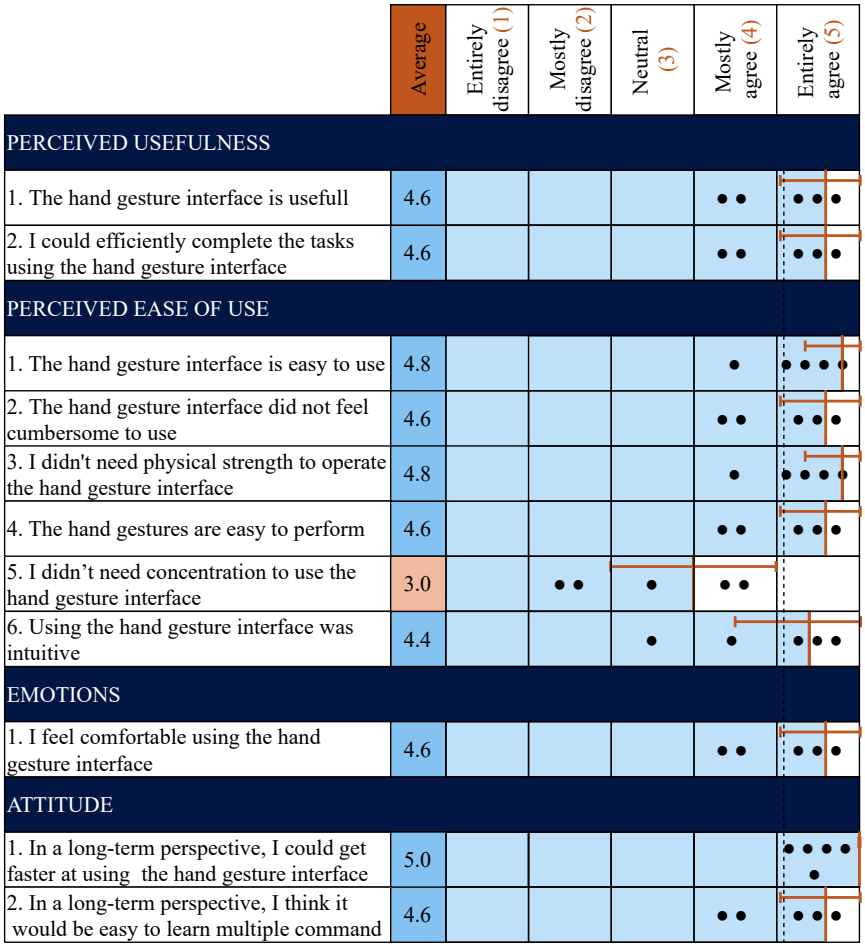


FIGURE 2.12 – Distributions of the total completion times by task and by interfaces.

Finally, some qualitative data has been gathered from the experiments as the participants answered a questionnaire after completing the tasks. This allowed the participants to rate their appreciation of the system. The results of this questionnaire are presented in Table 2.4.

The results show that the participants appreciated using the hand gesture interface. They felt that it would get easier to use with time, but still needed a fair level of concentration when using it. However, they did not find it cumbersome, counterintuitive or difficult to use. They also noted in the comments that they especially liked using the hand gesture interface in combination with the AssystMouse as it gave them quick access to shortcuts that were easy to execute.

TABLE 2.4 – Post experiment questionnaire results.



2.10 Conclusion

This paper has presented the development of an interface to control an assistive technology efficiently and intuitively. The goal was to help people living with upper or lower limb disabilities in their daily tasks. More specifically, the objective was to develop an interface to easily control high-dimensionality ATs with low-dimensionality devices. To do so, a second application of the sequence matching algorithm presented in [18] has been developed. The main advantage of the SMA is to navigate quickly among different modes of an AT without having to cluster them into multiple sub-groups while maintaining a robust system. The idea of the SMA was to combine simple actions into a sequence similar to Morse code.

For this project, the SMA has been applied to hand gestures recognized from IMU signals. This brought the problem of having to train a model to recognize those gestures with a dataset of a size significantly lower than that of a typical machine learning implementation. To answer the problem, four lightweight yet robust classification algorithms have been identified and tested against a five-class dataset. The classifiers were Support Vector Machines, Linear Discriminant

Analysis, Adaptative Boosting and K-Nearest Neighbors. The best performing classifier was the linear SVM with 95% of correct predictions on the offline data. Since the size of the training dataset was small, this score could be increased by simply having fewer classes (different gestures) or having gestures that are qualitatively more different. Fortunately, this can be done since it is the principal advantage of the SMA to give many different instructions with limited physical inputs. The SVM algorithm was preceded by the extraction of the features on the raw accelerometer and gyroscope signals, and by a PCA to reduce the dimensionality of the features. These components established the classification pipeline that predicts gestures from inputted IMU signals. The pipeline has then been wrapped in a triggering algorithm and implemented in real time so that it would output a prediction quickly on any valid input.

In addition to the elaboration of the gesture recognition routine, a complete actual application integrating the IMU, the SMA interface and a controlled device has been developed. The architecture of the system consisted of these three components (sensor, interface and device) implemented as independent modules organized together with a local server as a central piece. In the development environment, this configuration has proven versatile and efficient. The sensor module was a wireless commercial IMU and the device was a computer's mouse and keyboard (moves and shortcuts). Also, the user could interact with the system with a GUI to edit settings, gestures, sequences, and mappings and to have visual feedback of the state of the actions performed.

The implementation has been tested in two scenarios. For the control of the computer mouse, when the hand gesture interface is compared against a head-movement-based interface using a webcam, the latter seems superior. However, when combined, the hand gestures help participants increase their performance by 77% by offering quick shortcuts accessible with the gestures' sequences. Besides, even if the head movement interface offers better performance for the first half of the experiments, both interfaces may be better adapted to different populations and different scenarios.

Although the focus here is on an application of SMA with hand gestures, the interface is not limited to hands or to a computer as an AT. It could use head gestures (tilting) or lower limb moves (leg raise, foot stamps) as well. The application would not require any modifications. If the user wanted to control a robotic arm instead of a computer, only the controlled-device module (section 2.7.1) would need to be modified in the system. Future work would be to continue to develop more intelligent algorithms to increase the robustness of the system and to test the application with people living with disabilities.

2.11 Bibliographie

[1] V. Maheu, P. S. Archambault, J. Frappier, and F. Routhier, "Evaluation of the jaco robotic arm : Clinico-economic study for powered wheelchair users with upper-extremity disabilities," in *2011 IEEE International Conference on Rehabilitation Robotics*, pp. 1–5, IEEE, 2011.

[2] V. Lajeunesse, C. Vincent, F. Routhier, E. Careau, and F. Michaud, "Exoskeletons' design and usefulness evidence according to a systematic review of lower limb exoskeletons used for functional mobility by people with spinal cord injury," *Disability and rehabilitation : Assistive technology*, vol. 11, no. 7, pp. 535–547, 2016.

[3] N. Friedman, A. Cuadra, R. Patel, S. Azenkot, J. Stein, and W. Ju, "Voice assistant strategies and opportunities for people with tetraplegia," in *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, pp. 575–577, 2019.

[4] L. V. Herlant, R. M. Holladay, and S. S. Srinivasa, "Assistive teleoperation of robot arms via automatic time-optimal mode switching," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 35–42, IEEE, 2016.

[5] L. V. Herlant, "Algorithms, implementation, and studies on eating with a shared control robot arm," 2018.

[6] L. N. A. Struijk, L. L. Egsgaard, R. Lontis, M. Gaihede, and B. Bentsen, "Wireless intraoral tongue control of an assistive robotic arm for individuals with tetraplegia," *Journal of neuroengineering and rehabilitation*, vol. 14, no. 1, p. 110, 2017.

[7] D. Johansen, C. Cipriani, D. B. Popović, and L. N. Struijk, "Control of a robotic hand using a tongue control system—a prosthesis application," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 7, pp. 1368–1376, 2016.

[8] S. Azenkot and N. B. Lee, "Exploring the use of speech input by blind people on mobile devices," in *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, pp. 1–8, 2013.

[9] S. Poirier, F. Routhier, and A. Campeau-Lecours, "Voice control interface prototype for assistive robots for people living with upper limb disabilities," in *2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR)*, pp. 46–52, IEEE, 2019.

[10] C. L. Fall, G. Gagnon-Turcotte, J.-F. Dubé, J. S. Gagné, Y. Delisle, A. Campeau-Lecours, C. Gosselin, and B. Gosselin, "Wireless semg-based body–machine interface for assistive technology devices," *IEEE journal of biomedical and health informatics*, vol. 21, no. 4, pp. 967–977, 2016.

[11] D. Farina, N. Jiang, H. Rehbaum, A. Holobar, B. Graimann, H. Dietl, and O. C. Aszmann, "The extraction of neural information from the surface emg for the control of upper-limb prostheses : emerging avenues and challenges," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 4, pp. 797–809, 2014.

[12] E. Scheme and K. Englehart, "Electromyogram pattern recognition for control of powered upper-limb prostheses : state of the art and challenges for clinical use.," *Journal of Rehabilitation Research & Development*, vol. 48, no. 6, 2011.

[13] R. Raya, J. Roa, E. Rocon, R. Ceres, and J. L. Pons, "Wearable inertial mouse for children with physical and cognitive impairments," *Sensors and Actuators A : Physical*, vol. 162, no. 2, pp. 248–259, 2010.

[14] C. L. Fall, P. Turgeon, A. Campeau-Lecours, V. Maheu, M. Boukadoum, S. Roy, D. Massicotte, C. Gosselin, and B. Gosselin, "Intuitive wireless control of a robotic arm for people living with an upper body disability," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 4399–4402, IEEE, 2015.

[15] A. Lebrasseur, J. Lettre, F. Routhier, P. S. Archambault, and A. Campeau-Lecours, "Assistive robotic arm : Evaluation of the performance of intelligent algorithms," *Assistive Technology*, pp. 1–10, 2019.

[16] D.-S. Vu, U. C. Allard, C. Gosselin, F. Routhier, B. Gosselin, and A. Campeau-Lecours, "Intuitive adaptive orientation control of assistive robots for people living with upper limb disabilities," in *2017 International Conference on Rehabilitation Robotics (ICORR)*, pp. 795–800, IEEE, 2017.

[17] T. Simpson, C. Broughton, M. J. Gauthier, and A. Prochazka, "Tooth-click control of a hands-free computer interface," *IEEE Transactions on Biomedical Engineering*, vol. 55, no. 8, pp. 2050–2056, 2008.

[18] F. Schweitzer and A. Campeau-Lecours, "Intuitive sequence matching algorithm applied to a sip-and-puff control interface for robotic assistive devices," *arXiv preprint arXiv :2010.07449*, 2020.

[19] Y. Wakita, N. Yamanobe, K. Nagata, and M. Clerc, "Customize function of single switch user interface for robot arm to help a daily life," in *2008 IEEE International Conference on Robotics and Biomimetics*, pp. 294–299, IEEE, 2009.

[20] P. M. Pilarski, M. R. Dawson, T. Degris, J. P. Carey, and R. S. Sutton, "Dynamic switching and real-time machine learning for improved human control of assistive biomedical robots," in *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pp. 296–302, IEEE, 2012.

[21] P. Bevilacqua, M. Frego, E. Bertolazzi, D. Fontanelli, L. Palopoli, and F. Biral, "Path planning maximising human comfort for assistive robots," in *2016 IEEE Conference on Control Applications (CCA)*, pp. 1421–1427, IEEE, 2016.

[22] I. Kyranou, A. Krasoulis, M. S. Erden, K. Nazarpour, and S. Vijayakumar, "Real-time classification of multi-modal sensory data for prosthetic hand control," in *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pp. 536–541, IEEE, 2016.

[23] W. Qi, H. Su, C. Yang, G. Ferrigno, E. De Momi, and A. Aliverti, "A fast and robust deep convolutional neural networks for complex human activity recognition using smartphone," *Sensors*, vol. 19, no. 17, p. 3731, 2019.

[24] S. Jiang, B. Lv, W. Guo, C. Zhang, H. Wang, X. Sheng, and P. B. Shull, "Feasibility of wrist-worn, real-time hand, and surface gesture recognition via semg and imu sensing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3376–3385, 2017.

[25] A. S. Kundu, O. Mazumder, P. K. Lenka, and S. Bhaumik, "Hand gesture recognition based omnidirectional wheelchair control using imu and emg sensors," *Journal of Intelligent & Robotic Systems*, vol. 91, no. 3-4, pp. 529–541, 2018.

[26] Z. Lu, X. Chen, Q. Li, X. Zhang, and P. Zhou, "A hand gesture recognition framework and wearable gesture-based interaction prototype for mobile devices," *IEEE transactions on human-machine systems*, vol. 44, no. 2, pp. 293–299, 2014.

[27] R. Srivastava and P. Sinha, "Hand movements and gestures characterization using quaternion dynamic time warping technique," *IEEE Sensors Journal*, vol. 16, no. 5, pp. 1333–1341, 2015.

[28] Y.-L. Hsu, C.-L. Chu, Y.-J. Tsai, and J.-S. Wang, "An inertial pen with dynamic time warping recognizer for handwriting and gesture recognition," *IEEE Sensors Journal*, vol. 15, no. 1, pp. 154–163, 2014.

[29] A. Akl, C. Feng, and S. Valaee, "A novel accelerometer-based gesture recognition system," *IEEE Transactions on Signal Processing*, vol. 59, no. 12, pp. 6197–6205, 2011.

[30] M. Kim, J. Cho, S. Lee, and Y. Jung, "Imu sensor-based hand gesture recognition for human-machine interfaces," *Sensors*, vol. 19, no. 18, p. 3827, 2019.

[31] X. Wang, M. Xia, H. Cai, Y. Gao, and C. Cattani, "Hidden-markov-models-based dynamic hand gesture recognition," *Mathematical Problems in Engineering*, vol. 2012, 2012.

[32] C. McCall, K. K. Reddy, and M. Shah, "Macro-class selection for hierarchical k-nn classification of inertial sensor data.," in *PECCS*, pp. 106–114, 2012.

[33] A. M. Khan, Y.-K. Lee, and T.-S. Kim, "Accelerometer signal-based human activity recognition using augmented autoregressive model coefficients and artificial neural nets," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5172–5175, IEEE, 2008.

[34] E. Estrada, H. Nazeran, P. Nava, K. Behbehani, J. Burk, and E. Lucas, "Eeg feature extraction for classification of sleep stages," in *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 1, pp. 196–199, IEEE, 2004.

[35] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *International work-conference on artificial neural networks*, pp. 758–770, Springer, 2005.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn : Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[37] Frederic Schweitzer, "TeamAT IMU-hand-gestures." https://github.com/team-ingreadaptulaval/TeamAT_IMU-hand-gestures, 2021.

[38] Assistyv, "AssystMouse." https://www.assistyv.com/. Online ; accessed 2021-03-15.

[39] J. Schmidtler, K. Bengler, F. Dimeas, and A. Campeau-Lecours, "A questionnaire for the evaluation of physical assistive devices (quead) : Testing usability and acceptance in physical human-robot interaction," in *2017 ieee international conference on systems, man, and cybernetics (smc)*, pp. 876–881, IEEE, 2017.

Chapitre 3

Développement d'un système d'applications connectées pour la collecte de données et le suivi en réadaptation

3.1 Résumé

L'interconnectivité entre les appareils intelligents, souvent reconnue comme Internet des objets, est un phénomène de plus en plus présent dans plusieurs sphères de la société. Cela permet une nouvelle approche simple et peu coûteuse pour la manipulation de données. Ce chapitre présente le développement d'un écosystème d'applications facilitant la collecte et l'analyse de données en milieux cliniques et en laboratoires dans un contexte de réadaptation. Des microcontrôleurs supportant des connexions sans fil WiFi et Bluetooth ont été utilisés en tant qu'appareils agissant comme point de collecte de données. Les données sont envoyées directement sur un serveur hébergeant une base de données et exposant une API REST comme point d'accès. Les données peuvent être récupérées ou visualisées à partir d'applications web et mobile. Les développements présentés dans ce chapitre sont appliqués en réadaptation, mais ne s'y limitent pas. D'autres domaines tels que la mécanique du bâtiment pourraient facilement en tirer parti.

3.2 Abstract

The interconnectivity among technological devices, often known as the Internet of things, is a growing trend in multiple areas of the society. It represents an inexpensive and simple approach to handle data. This chapter presents the development of an application ecosystem for the gathering and the analysis of data, in clinical environments and in laboratories, in a

rehabilitation context. ESP32 microcontroller modules supporting wireless WiFi and Bluetooth connections have been used as data collecting devices. The data are sent to a server hosting a database and exposing a REST API. The data can be downloaded and/or visualized from web and mobile applications. The developments presented in this chapter are applied to rehabilitation, but are not limited to the field. Other fields such as HVAC monitoring could easily take advantage of this application.

3.3 Introduction

Avec l'arrivée des téléphones intelligents au tournant de 2010, s'est amorcé un virage technologique axé sur la connectivité entre les appareils et leurs interactions à l'intérieur de divers écosystèmes. En effet, ces appareils intelligents ont su tirer avantage de mécanismes de communications sans fil (Bluetooth, WiFi, etc.) pour échanger entre eux des données et réagir de façon intelligente par le biais d'interfaces machine-machine. Un paradigme de plus en plus connu et appliqué pour le développement de telles interfaces est celui d'Internet des objets (*Internet of things*, IoT).

Une application majeure de l'IoT aujourd'hui est celle des maisons intelligentes. Différents appareils connectés tels que des interrupteurs, haut-parleurs intelligents avec assistant vocal, télévisions intelligentes et autres sont connectés au réseau WiFi d'un domicile et contrôlés à partir d'un simple téléphone intelligent à portée de main du propriétaire. Le même principe s'applique aussi en industrie. L'industrie 4.0 fait maintenant intervenir des processus intelligents et automatisés pour la maintenance [1] et la production [2] pour ne nommer qu'eux. Tous ces systèmes impliquent de l'acquisition de données et des interactions machine-machine qui peuvent s'intégrer dans un réseau de capteurs [3] de manière à créer des usines intelligentes [4].

Dans le domaine de la santé, on tire aussi avantage de l'Internet des objets [5, 6]. Que ce soit en milieu clinique ou au domicile, les systèmes connectés permettent de faciliter, entre autres, le suivi médicale et les mécanismes d'alerte d'urgence, lors d'une chute chez une personne âgée par exemple.

3.3.1 Problématique

En réadaptation, pour la recherche ou la pratique, nombreuses sont les situations où l'on requiert du suivi et de l'analyse de données sur des patients. On peut penser par exemple, lors de courtes séances en laboratoire pour l'analyse de la marche ou pour suivre le rythme cardiaque, ou encore sur de plus longues périodes pour l'analyse sportive ou la prévention de blessure chez les aînés ou les travailleurs. Par contre, enregistrer et récupérer ces données requiert souvent des appareils dispendieux ou encombrants. Aussi, en pratique, le processus de collecte de données peut être laborieux impliquant de d'abord enregistrer les données sur un

stockage externe comme une carte SD, puis de les transférer manuellement sur un ordinateur ou elles pourront être finalement analysées.

3.3.2 Objectif du projet

L'objectif de ce projet est de développer une solution flexible, efficace et rapide pour répondre à la problématique énoncée. Concrètement, cette solution consiste en un écosystème d'appareils intelligents et d'applications connectés, sur mesure, peu coûteux, léger et offrant un produit qui peut aussi bien être clé-en-main que prêt à programmer. Ce chapitre présente le développement d'une première itération d'un tel système implémentant les principales fonctionnalités nécessaires pour collecter et analyser des données facilement et sans fil en laboratoire et en milieu clinique.

3.3.3 Organisation du chapitre

Ce chapitre présente d'abord les systèmes de collecte de données déjà existants, ainsi que diverses implémentions de mécanismes IoT qui ont servi d'inspiration au projet. La section 3.5 présente une vue d'ensemble du système développé tandis que les sections suivantes détaillent chaque partie du système en présentant le choix des technologies, les développements techniques et les perspectives de travaux futurs spécifiques à chaque partie. De plus, la base de code pour l'application web est disponible sur demande en [7] et celle pour l'application mobile en [8].

3.4 Revue de produits commerciaux

On retrouve, sur le marché, plusieurs produits qui apportent une partie de la solution au projet. Cette section survole les produits électroniques et les plateformes logicielles utilisant diverses architectures et protocoles de communications ayant servi d'inspiration à l'élaboration de la solution présentée par après dans ce chapitre.

3.4.1 Plateformes *cloud* IoT

Les géants des technologies de l'information Google [1], Amazon [2], IBM [3] et Microsoft [4] offrent tous une plateforme IoT comprenant de multiples fonctionnalités [9]. Ces plateformes permettent aux entreprises de développer leurs applications IoT en leur offrant des solutions de stockage et d'analyse de données et des infrastructures propices à l'évolutivité de celles-ci. Les applications développées dans le but d'automatiser complètement les interactions entre les appareils, pour le suivi de même que pour le contrôle, peuvent tirer pleinement avantage

1. Google Cloud, `https://cloud.google.com/solutions/iot`
2. Amazon Web Services, `https://aws.amazon.com/iot/`
3. IBM Cloud, `https://www.ibm.com/cloud/internet-of-things`
4. Microsoft Azure, `https://azure.microsoft.com/en-ca/overview/iot/`

de ce genre de service. Dans ce cas, comme schématisé sur la figure 3.1, la plateforme permet de gérer toutes les interactions entre les appareils, ainsi qu'avec la plateforme IoT.

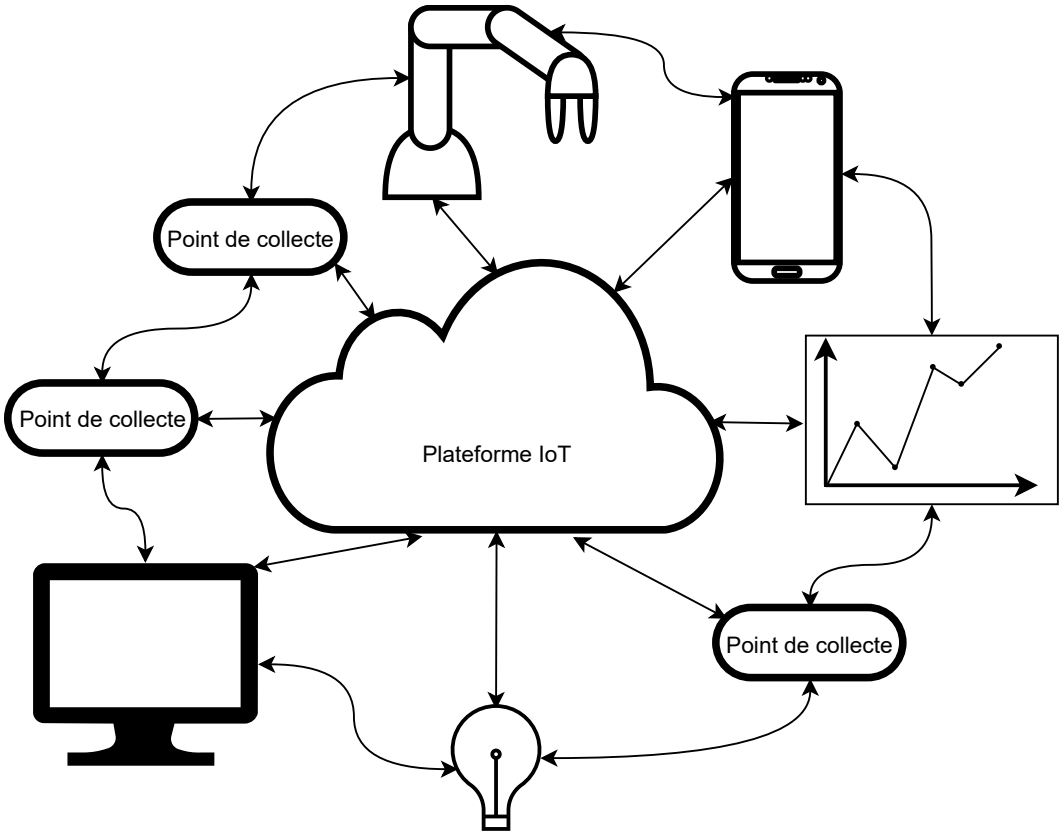


FIGURE 3.1 – Schéma d'interaction entre les appareils pour les services de plateforme IoT.

Dans le cadre de ce projet, pour l'utilisation en réadaptation, le contrôle est moins important et on s'intéresse plus simplement à envoyer des données à un serveur web pour être en mesure de les récupérer sur un ordinateur ou un téléphone intelligent plutôt que d'échanger des données entre les points de collecte. Le schéma d'interactions peut donc être simplifié (ordonné) tel qu'illustré sur la figure 3.2.

Plusieurs services commerciaux à plus petite échelle sont disponibles et mieux adaptés pour implémenter le schéma de la figure 3.2. Un exemple populaire, Thingspeak [5], est un service offert par MathWorks permettant de récolter des données et d'en faire le suivi. L'idée est d'utiliser un microcontrôleur Arduino supportant une connexion internet. Thingspeak propose une librairie Arduino permettant à un développeur de programmer le microcontrôleur afin d'envoyer les données au serveur. Ensuite, les données peuvent être directement lues sur le site web du service et analysées automatiquement à l'aide de codes Matlab. Celles-ci peuvent aussi être exportées en format *.csv* ou récupérées directement avec le logiciel Matlab. Ce type

5. Thingspeak, https://thingspeak.com/

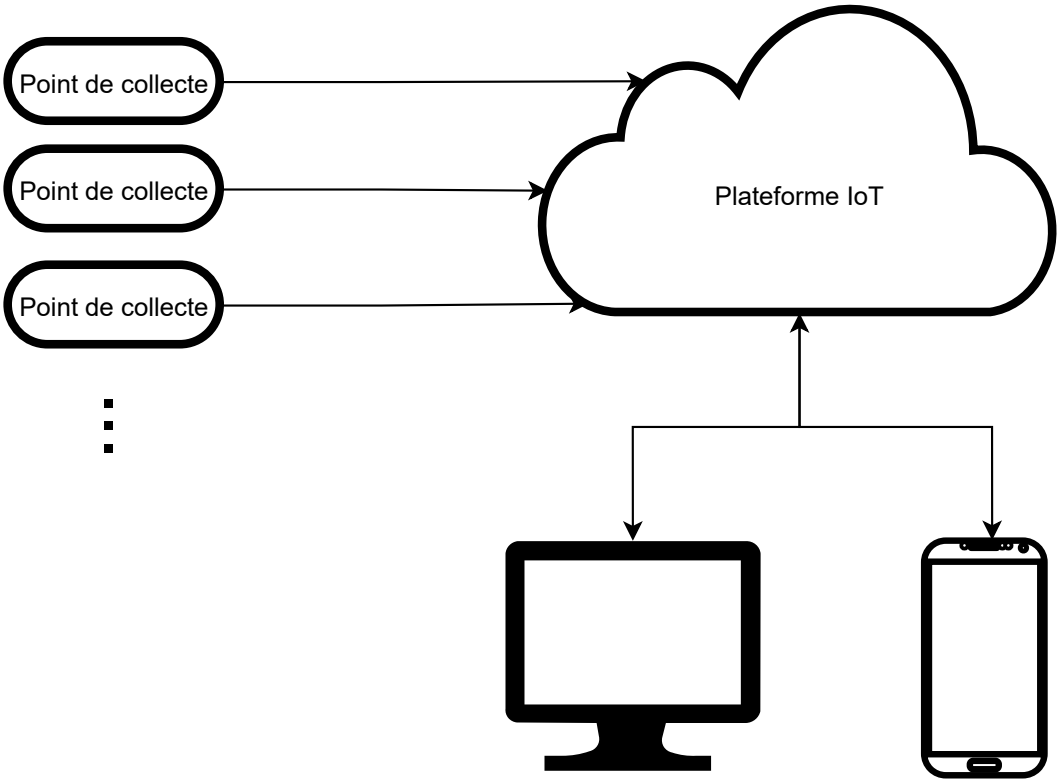


FIGURE 3.2 – Schéma d'interaction entre les appareils dans le cadre du projet.

de plateforme se prête très bien à la problématique du projet. Par contre, quelques éléments représentent des inconvénients :

— Le service devient coûteux lorsque les applications prennent de l'expansion,
— Le nombre d'envois au serveur est limité,
— Le service est bien conçu pour fonctionner avec Matlab, mais l'utilisation avec d'autres langages dépend de librairies indépendantes,
— Aucune application mobile disponible.

Thinger.io [6] et ThingsBoard [7] sont d'autres plateformes similaires offrant des fonctionnalités de collecte et de visualisation de données. Ces systèmes ont tous servi d'inspiration pour ce projet.

3.4.2 Appareils connectés

En plus du côté logiciel du projet, il y a aussi l'aspect électronique (*hardware*) à planifier. L'objectif est d'avoir un système électronique peu coûteux et simple d'utilisation qui se prête aussi bien à devenir un produit *plug-and-play* qu'un outil de développement.

6. Thinger.io, https://thinger.io/
7. ThingsBoard, https://thingsboard.io/

Maison intelligente

En analysant quelques produits connectés commerciaux, on retrouve quelques caractéristiques désirables au projet. Les appareils pour maisons intelligentes, tels que les interrupteurs et prises intelligents, assistants vocaux, télévisions intelligentes (figure 3.3) et autres, doivent être connectés au réseau WiFi de la maison pour bien fonctionner. Dans la plupart des cas, pour connecter ces appareils au réseau, l'utilisateur doit d'abord établir une connexion Bluetooth entre un appareil intelligent et son téléphone intelligent. Ce lien sans fil permet au téléphone de transmettre les authentifiants requis pour la connexion (ex. informations du WiFi, compte Google/Amazon). Ce processus de connexion est intéressant et peut être implémenté au projet pour l'appareil de type *plug-and-play* servant de point de collecte.

FIGURE 3.3 – Exemples d'appareils connectés pour maisons intelligentes. De gauche à droite : une prise intelligente Teckin Mini (amazon.ca, 2021) , un haut-parleur intelligent avec assistant vocal Amazon Echo Dot (amazon.ca, 2021) et un lecteur de flux multimédias intelligent Google Chromecast (Google store, 2021).

Montres intelligentes

D'autres technologies intéressantes sont les montres intelligentes. La manière dont elles échangent les données avec un téléphone intelligent est directement applicable à ce projet. En effet, la plupart tirent avantage du protocole *Bluetooth Low Energy* pour rester en communication constante avec le téléphone. Cela permet à la montre de gérer les notifications, les lecteurs multimédias et les appels par exemple, sans consommer trop d'énergie. D'autre part, les montres spécialisées pour le suivi d'exercices sportifs telles que les montres fitbit [8] de la figure 3.4, combinent cette méthode avec une application mobile permettant de synchroniser manuellement les données.

Ainsi, c'est l'utilisateur qui est responsable du déclenchement du transfert de données. Ce type de mécanisme peut être utilisé pour ce projet dans le cas ou l'analyse continue de données n'est

8. fitbit, https://www.fitbit.com/

FIGURE 3.4 – Montre (Fitbit, 2021) et application (ZDNet, 2019) d'entraînement Fitbit.

pas nécessaire. Dans ce cas, l'utilisateur pourrait seulement envoyer ses données au serveur au moment opportun. Sauvegardant ainsi l'énergie de l'appareil de collecte. L'application sert aussi à avoir un visuel rapide des données sous forme de graphiques.

Microcontrôleur IoT

Les types de produits présentés ci-haut sont clé en main. Aucune programmation n'est requise et ils sont prêts à brancher et à utiliser. Inévitablement, dans ces produits se trouve un microcontrôleur déjà programmé dont l'utilisateur n'a pas à se soucier. Pour ce projet par contre, un des objectifs est d'avoir un produit qui peut être potentiellement programmé par l'utilisateur. Le microcontrôleur est un élément très important pour le projet. Il représente le point de collecte des données et donc l'interface entre l'environnement physique et le système. À cet effet, plusieurs microcontrôleurs spécialisés pour les applications IoT sont disponibles sur le marché à faible coût.

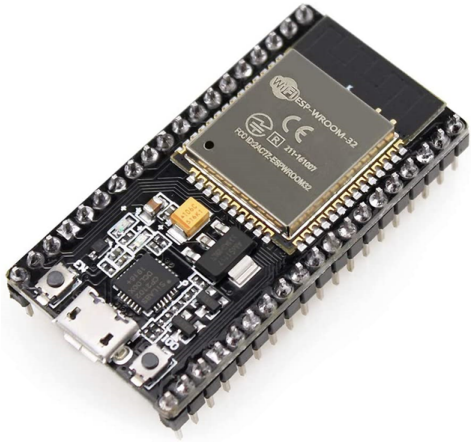


FIGURE 3.5 – Module de développement IoT ESP32 (amazon.ca, 2021).

Les modules de développement ESP sont très populaires pour la réalisation de projet IoT. L'ESP32, présenté à la figure 3.5, offre des connectivités WiFi et Bluetooth en plus de représenter un appareil plus qu'assez puissant pour la collecte de données et disponible à très faible coût (mois de 10$). Le microcontrôleur est compatible avec le framework Arduino, mais peut aussi être programmé directement en C.

3.5 Architecture

L'écosystème d'applications développé doit faire interagir quatre éléments principaux : une application web, une application mobile, un microcontrôleur WiFi (pour les points de collecte de données) et un serveur web qui est l'élément central. On vise ici d'avoir un système qui peut fonctionner de façon clé en main, mais qui est aussi flexible pour les développeurs, à la manière de *Thingspeak*, pour pouvoir s'adapter à plusieurs scénarios d'utilisation. Cette section étudie les différents paradigmes d'architecture et de communication propices à une telle application, puis présente l'architecture choisie.

3.5.1 Fonctionnement général

Tout d'abord, il est important de comprendre comment le système d'application fonctionne du point de vue de l'utilisateur. Précisément, comment faire pour collecter des données ?

L'utilisateur doit se munir d'un ordinateur, d'un microcontrôleur WiFi (comme un ESP32) et s'il le désire, d'un téléphone intelligent. Il se crée ensuite un compte sur le site web et ajoute un appareil (*device*). Le nouvel appareil est alors indiqué par un numéro d'identification (*ID*). Le microcontrôleur est responsable de faire la lecture d'un ou plusieurs capteurs. L'utilisateur programme maintenant le ESP32 avec la librairie disponible en indiquant dans le programme le ou les numéros d'identification créés sur le site web. Il y a un *ID* par capteur. Lors de la programmation du microcontrôleur, l'utilisateur peut indiquer ses authentifiants pour son compte utilisateur et pour son réseau WiFi. Il peut aussi choisir de ne pas les écrire dans le code. Dans ce cas, il devra les fournir depuis l'application mobile. À l'aide de celle-ci, il pourra établir une connexion Bluetooth avec le microcontrôleur et envoyer ses données d'authentification. Une fois ces données vérifiées et la connexion entre l'ESP32 et le serveur web établie, les données des capteurs sont envoyées. Ces données peuvent être visualisées et récupérées directement sur le site web ou l'application mobile.

L'utilisateur peut aussi choisir d'utiliser l'application mobile pour lire directement les données provenant du microcontrôleur sans passer par le réseau, mais plutôt via Bluetooth. Dans le cadre d'une expérience en laboratoire, par exemple, pour avoir un visuel en temps réel. Ou encore, si l'accès au réseau est impossible.

3.5.2 Internet des objets

La définition du paradigme d'IoT n'est pas unanime dans la littérature. La définition plus classique de l'IoT serait

> «un réseau ouvert et global d'objet intelligent ayant la capacité de s'organiser, partager de l'information, des données et des ressources, agissant et réagissant aux situations et changements de l'environnement» [10].

Ce projet ne pousse pas aussi loin l'implémentation. Néanmoins, on tire avantage de la connectivité entre les objets intelligents et le réseau pour faciliter les procédés. Une définition du paradigme IoT mieux adaptée à ce projet serait plutôt

> «une nouvelle approche de développement d'interface machine-machine exploitant l'interconnectivité des objets intelligents pour évaluer et suivre l'évolution de leur environnement physique.»

En effet, la première itération du projet présentée ici se concentre sur la collecte de données seulement, cette seconde définition est donc plus appropriée. Par contre, en ayant une approche IoT pour les développements, on laisse la porte ouverte à de futurs développements encore plus poussés.

3.5.3 Protocoles de communication

Pour ce projet, deux protocoles de communications qui sont régulièrement utilisés pour à des applications IoT ont été étudiés. Le protocole HTTP qui supporte la grande majorité des échanges de données de l'Internet et le protocole MQTT, plus léger et souvent appliqué pour des applications de messagerie entre les appareils sont les méthodes de communication d'intérêt.

HTTP : *Hypertext Transfer Protocol*

Le protocole HTTP est la base de la plupart des communications serveur-client via Internet. De façon générale, le client envoie une requête à un serveur et celui-ci lui répond. La réponse comprend entre autres, un statut (*status*) et un contenu (*body*). À partir de ces requêtes, le client demande au serveur d'effectuer des opérations sur sa base de données. Les principales opérations sont résumées à la table 3.1.

Table 3.1 – Requêtes principales CRUD pour les communications HTTP.

Requête	Opération	Action
POST	Créer (**C**reate)	Ajouter des éléments à la base de données
GET	Lire (**R**ead)	Extraire des éléments à la base de données
PUT	Mettre à jour (**U**pdate)	Modifier des éléments de la base de données
DELETE	Supprimer (**D**elete)	Retirer des éléments de la base de données

Le statut de la réponse correspond simplement un code accusant de la réception de la requête (ex. 404 not found, 200 OK, etc.). Le format du texte du contenu de la requête, ainsi que de la réponse dépendra de l'architecture de l'application. Par exemple, pour les requêtes envoyées à une API REST (section 3.5.4), les formats usuels sont XML et JSON, bien que ce dernier soit très largement dominant de nos jours.

MQTT : *Message Queuing Telemetry Transport*

À l'opposé du protocole HTTP et de son principe de requête créant des communications *one-to-one*, le protocole MQTT implémente plutôt une philosophie *publish-subscribe* favorisant les communications *one-to-many* et *many-to-one*. En effet, comme il est possible d'observer sur la figure 3.6, chaque client a accès aux données de tous les autres clients et vice-versa par l'entremise du serveur (qu'on appelle ici, *broker*).

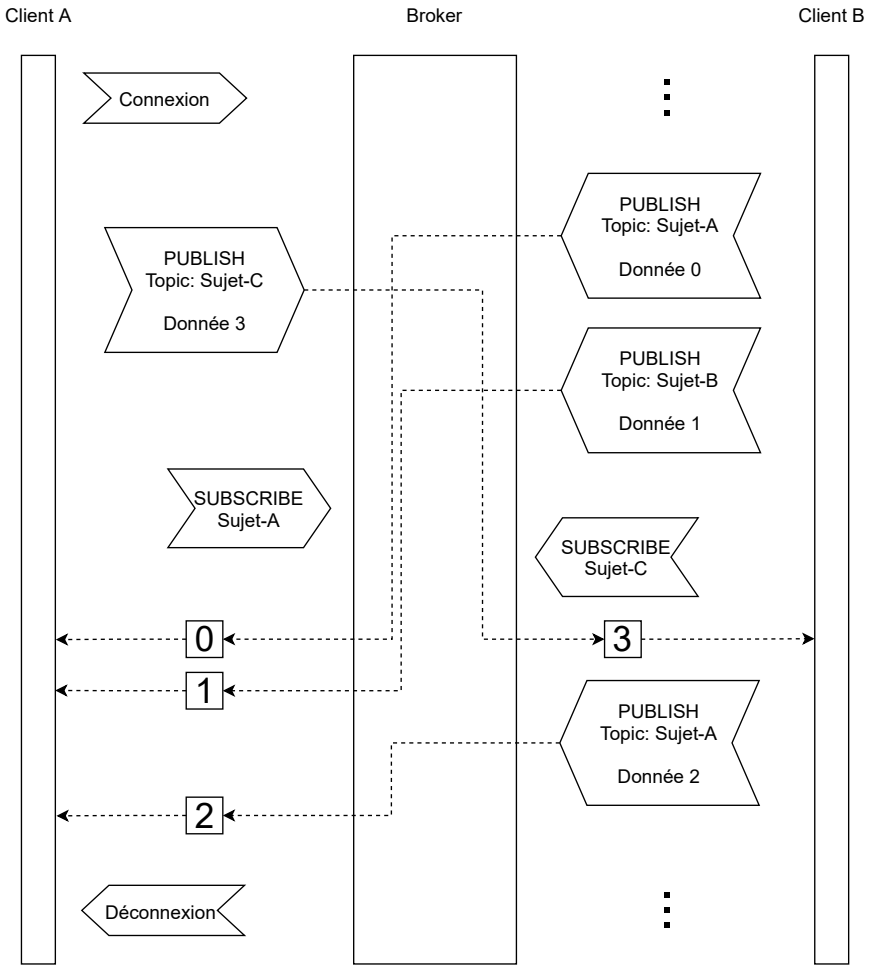


FIGURE 3.6 – Schématisation du processus d'échange de données *publish-subscribe* avec le protocole MQTT (adapté de Simon A. Eugster, Wikipedia, 2018).

Le *broker* gère un flux de données continu provenant de tous les clients. Chaque donnée est as-

sociée à un sujet (*topic*). Différents clients peuvent publier sur le ou les *topics* qu'ils souhaitent. Ils peuvent aussi souscrire aux *topics* d'intérêts. Ainsi, tous les clients sont interconnectés par le service de messagerie qu'est le *broker*.

On constate rapidement les avantages de travailler avec le *broker* MQTT dans le cadre d'applications IoT. Par contre, pour la première itération de ce projet, le besoin d'exploiter tous ces avantages n'est pas présent. En effet, le projet se concentre plus sur des communications *one-to-one* client-serveur. Pour cette raison et pour le niveau de complexité de l'implémentation pour la gestion des comptes utilisateurs, des permissions et des aspects sécurité, les développements autour du protocole MQTT ont donc été laissés de côté. Éventuellement, une approche simple serait d'implémenter un *broker* simple sur un microcontrôleur connecté au réseau local (comme un Raspberry Pi) avec un minimum de sécurité qui agirait comme un serveur local. De cette façon, ce produit serait prêt à installer avec un minimum de manipulations pour l'utilisateur et permettrait d'échanger des données entre les appareils.

Plusieurs implémentations commerciales ou open-source de *broker*/client MQTT sont disponibles. Un produit comme Mosquitto[11], supporté par la fondation Eclipse offre beaucoup de fonctionnalités et est très flexible tout en restant léger et sécuritaire. Il est supporté par plusieurs langages populaires tels que Python, C++, Java et JavaScript.

3.5.4 Architecture développée

L'architecture IoT est normalement représentée sous forme de couches [12, 13, 14, 15]. D'une implémentation à une autre, l'organisation des couches change, mais celles qui reviennent le plus souvent dans la littérature sont présentées à la table 3.2.

TABLE 3.2 – Couches de l'architecture IoT pour l'application de suivi en réadaptation.

Couche	Définition générale	Implémentation spécifique à l'application
Couche d'application	Interfaces utilisateurs, dispositifs de gestion de données	Applications web et mobiles
Couche intermédiaire (*middleware*)	Responsable du stockage, du traitement et de l'analyse des données	Serveur web et base de données
Couche réseau	Responsable de la transmission des données	WiFi et Bluetooth
Couche de perception	Capteurs et appareils physiques pour la collecte de données	Points de collecte, microcontrôleurs

Concrètement, la meilleure façon d'implémenter le schéma de la figure 3.2 est de développer une API REST (*Application Programming Interface, Representational state transfer*)[16]. L'architecture REST permet de manipuler des données entre un client (quel qu'il soit) et une base de données sous forme textuelle via des requêtes HTTP. En l'occurrence, le format JSON est utilisé. Le principe général est de demander au serveur un jeton (*token*) en s'authentifiant

en tant que client. Ce jeton devra par la suite être joint à chaque requête subséquente du client. Il permettra d'authentifier le client sans avoir besoin d'établir une session impliquant l'utilisation de *cookies* entre les deux entités. Le jeton a une durée de vie limitée, le client devra donc en demander un autre lors de l'expiration de celui-ci. Les étapes de la communication client-serveur sont les suivantes :

1. Le client envoie ses authentifiants au serveur en format JSON et reçoit ses informations et un jeton,
2. Le client fait des requêtes aux points d'accès de l'API en mettant le jeton en entête de la requête,
3. Lorsque le client à terminé, il peut se «déconnecter», ce qui entraînera la destruction du jeton.

De ce fait, en tout temps, le client et le serveur sont découplés. Cela permet à l'application de prendre facilement de l'expansion en plus d'échanger les données entre le serveur et un client toujours de la même façon, indifféremment de la nature ce dernier soit un navigateur web, une application mobile ou un microcontrôleur.

Indépendamment des interactions via Internet, il est aussi pertinent, en réadaptation d'avoir un visuel plus rapide des données récoltées. Pour ce faire, une API Bluetooth est aussi développée en parallèle entre les points de collecte et l'application mobile. Ainsi, les chercheurs en laboratoire peuvent voir les données des capteurs portés par un sujet sans s'encombrer de fils supplémentaires et à faibles coûts. La schématisation de l'architecture finale développée est présentée à la figure 3.7.

Les sections qui suivent détaillent chaque élément de l'architecture présentée ci-haut. Elles expliquent les raisonnements qui ont mené aux choix des différentes technologies et agissent à titre de documentation et d'outils de transfert de connaissances pour la continuation du projet.

3.6 Le backend

Le backend inclut la base de données et une API pour faire le lien entre cette dernière et le reste de l'écosystème. C'est le côté serveur de l'application. Plusieurs technologies sont disponibles pour chacun de ces deux éléments et celles-ci doivent être compatibles. En effet, certains frameworks pour développer l'API fonctionnent plus facilement ou plus efficacement avec certains systèmes de gestion de base de données spécifiques. Les choix du framework et du type de base de données vont donc de concert. Pour les raisons énoncées dans les sections qui suivent, le framework Django [9] du langage Python ainsi que le systèmes de gestion de base de données PostgreSQL [10] sont choisis pour construire le backend de l'application.

9. Django, https://www.djangoproject.com/
10. PostgreSQL, https://www.postgresql.org/

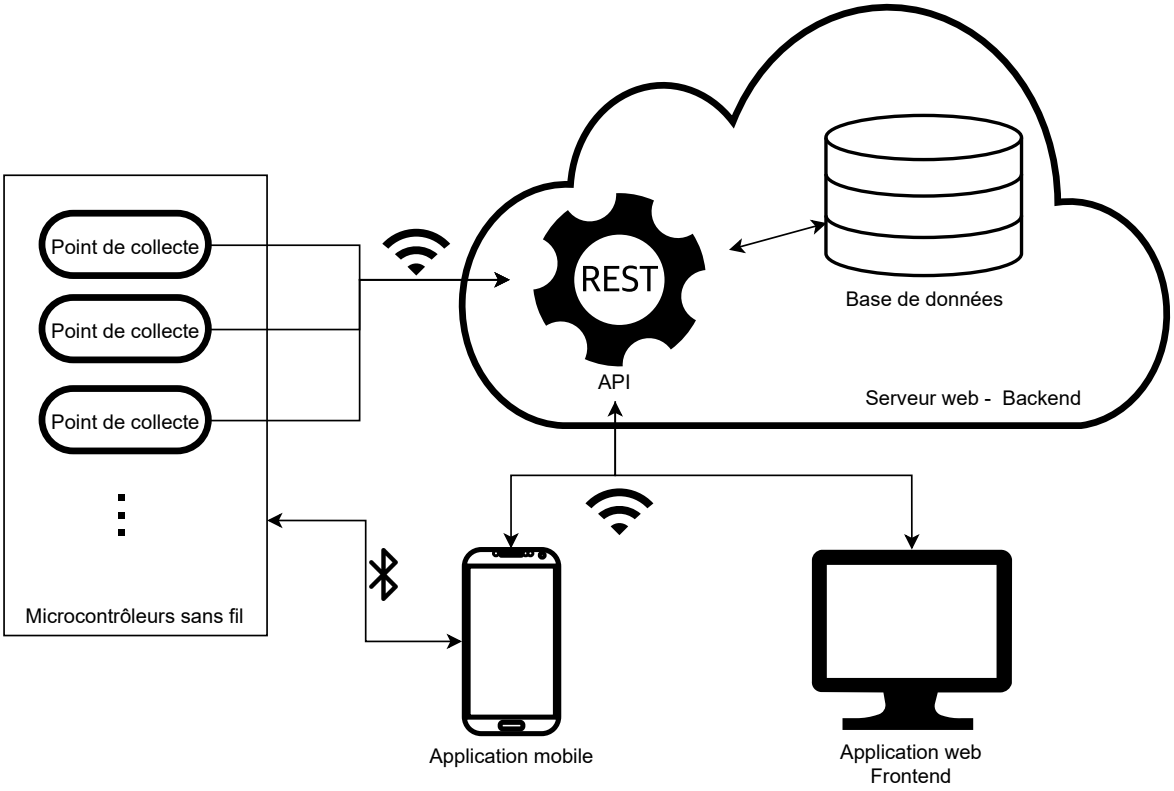


FIGURE 3.7 – Architecture de l'écosystème développé.

3.6.1 Base de données

Le choix d'un système de gestion de base de données est très important, surtout lorsque la manipulation et le stockage de séries de données sont au coeur du projet. Voici tout d'abord deux concepts à distinguer [17] :

— Base de données : une collection logique de données. Les deux types d'architecture à distinguer sont les bases de données relationnelles (SQL) et NoSQL (*Not Only SQL*),

— Systèmes de gestion de base de données (*Database management system*, DBMS) : un logiciel permettant de créer et administrer une base de données.

Le projet requiert de la base de données de contenir des informations d'authentification et un endroit pour enregistrer des données provenant de différents capteurs. Malgré la hausse en popularité des bases de données NoSQl, le paradigme SQL domine toujours le marché [18]. De plus, Django ne supporte officiellement aucune base de données NoSQL. Un des avantages d'opter pour un système NoSQL, le plus populaire étant MongoDB [11], aurait été la flexibilité dans l'organisation des données à mesure que le développement progresse. Or, les fonctionnalités de la base de données sont bien définies au départ et ne nécessitent pas cette flexibilité. De plus, un DBMS comme MongoDB permet à l'application de prendre de l'expansion plus

11. MongoDB, https://www.mongodb.com/

facilement étant donnée sa structure [19]. Cependant, pour une première version du projet, une base de données SQL plus classique fait amplement le travail [20]. Au final, le DBMS relationnel PostgreSQL est choisi pour sa compatibilité avec Django, l'accessibilité des ressources d'aide et parce qu'il est conçu pour l'évolutivité des applications et l'analyse complexe de données.

Les données de la base de données sont regroupées en tables en deux dimensions qui ont des interactions entre elles. La figure 3.8 présente un diagramme de tables nécessaires au fonctionnement de l'application. Par exemple, la table *Données* comprend les colonnes *ID*, *Donnée*, *Horodatage* et *Capteur-ID*. La contrainte *PK* (*primary key*, clé primaire) est appliquée sur la colonne *ID*. La clé primaire identifie de façon unique chaque ligne de la table. La contrainte *FK* (*foreign key*, clé étrangère) est appliqué sur la colonne *Capteur-ID*. La clé étrangère indique une relation avec une autre table en pointant depuis la clé primaire de cette

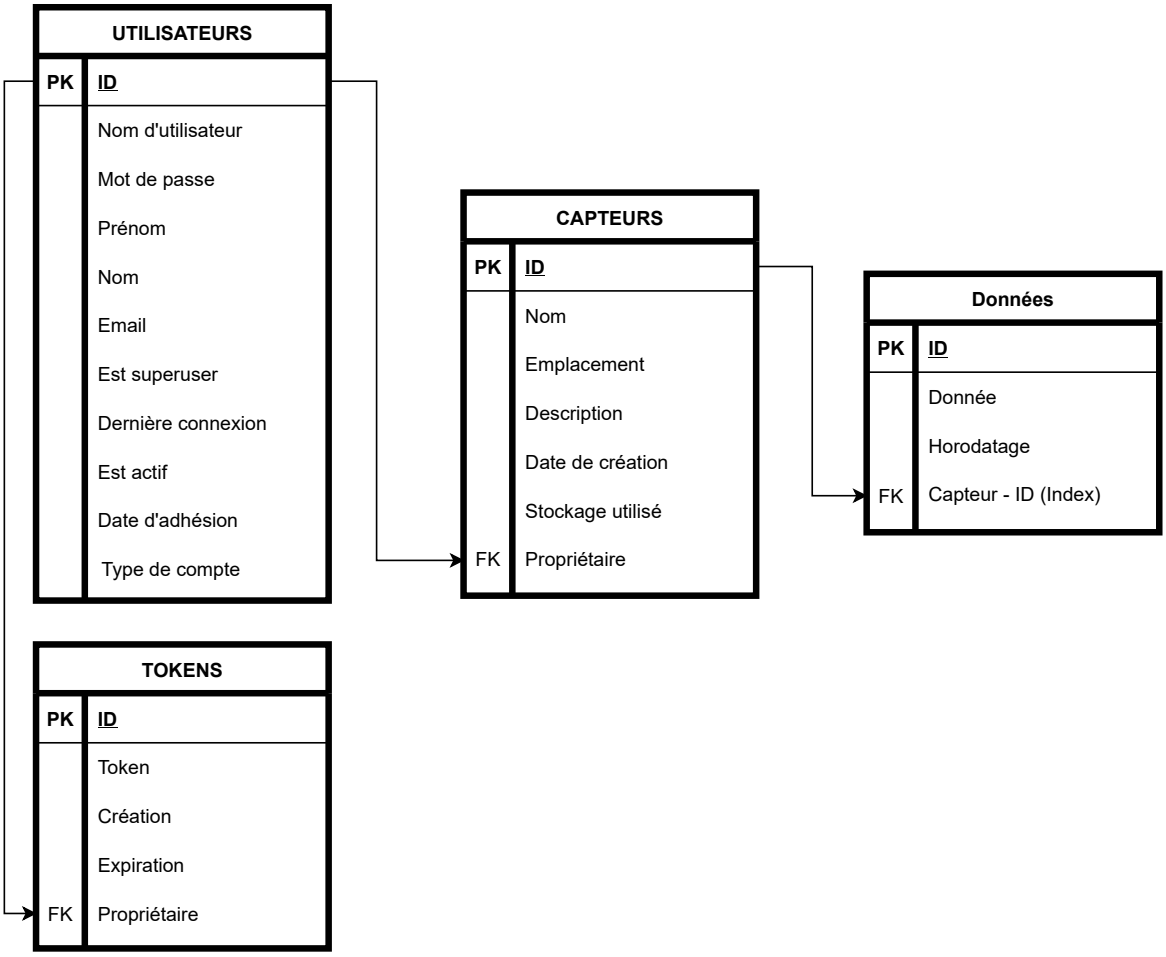


FIGURE 3.8 – Diagramme simplifié des tables de la base de données.

dernière. Finalement, une colonne marquée comme *index* permet le filtrage plus rapidement. Ainsi, en marquant la colonne *Capteur-ID* comme index, les données associées à un capteur en particulier sortiront plus rapidement.

3.6.2 API

Django est un framework de développement web écrit en Python. Les avantages de l'utiliser sont nombreux. Tout d'abord, Django permet de développer très rapidement et d'avoir un produit tangible avec un minimum de code. Sa philosophie très étroite et rigoureuse permet de construire des applications robustes et sécuritaires en proposant des mécanismes d'authentification et d'interactions avec la base de données déjà pré construits et rigoureusement éprouvés. De plus, une fois les principales fonctionnalités de l'application établies, créer une page web d'administrateur pour gérer le contenu de l'application se fait en quelques lignes de code.

Étant donné que l'architecture du backend est sous la forme d'une API REST, on doit travailler avec l'extension *Django REST framework*[12] pour les développements. Les principales utilités d'un tel framework est d'avoir des mécanismes d'authentification à l'aide de jetons, de sérialisation de données et de construction de points d'accès pour interagir avec la base de données selon les permissions des clients.

Manipulation de données

Chaque requête HTTP de la part d'un client entraîne une manipulation de données de la part du serveur. Le processus est toujours le même quelque soit la requête et il est représenté par la figure 3.9.

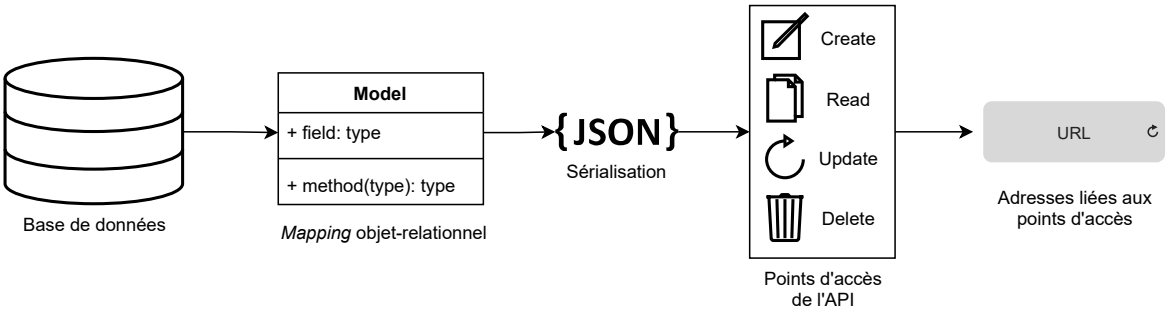


FIGURE 3.9 – Flux de manipulations de données à travers le backend de l'application.

Comme la plupart des frameworks de développement web bâtis sur des langages orientés objet, Django permet d'interagir avec la base de données de façon abstraite. C'est-à-dire que les tables de celle-ci sont modélisées comme des objets et offrent des méthodes permettant de les modifier. Cette abstraction est appelée *Mapping objet-relationnel*. Pour Django, les tables

12. Django REST framework, `https://www.django-rest-framework.org/`

de la base de données sont donc des classes héritant de la classe Model du framework. Ainsi, les champs des classes définissent les colonnes de la table et les relations et contraintes de ces champs peuvent être définies comme paramètres.

La seule partie exposée du backend, le dernier bloc de la figure 3.9, est l'adresse (URL). Chaque URL accessible aux clients est liée à un point d'accès de l'API servant à effectuer des opérations CRUD. Ces points d'accès analysent les permissions du client et envoient les données correspondantes à la requête qui ont été extraites de la base de données et sérialisées (transformées en chaîne de caractères, en format JSON).

Organisation du code

Un projet Django est organisé sous forme d'application. Dans le cas de ce projet, on inclut quatre applications (baseproject, accounts, devices, frontend) selon la disposition présentée à la figure 3.10.

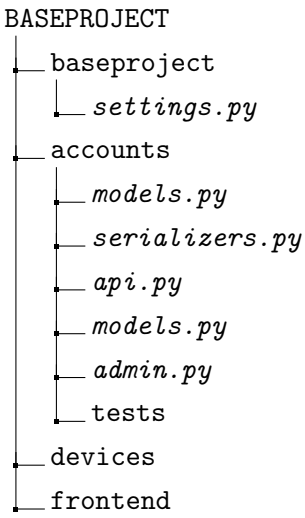


FIGURE 3.10 – Arborescence de projet Django et des quatre applications.

L'application `baseproject` contient les fichiers de configuration du projet et est créée par défaut par Django. Le fichier `settings.py` doit, entre autres, énumérer toutes les applications et les librairies dépendantes de Django utilisées par le projet. Aussi, ce fichier renferme les informations liées à la base de données. Les applications *devices* et *accounts* sont responsables des points de collectes (identification des microcontrôleurs, données) et des comptes usagers (inscriptions, authentification, gestion des jetons) respectivement. Ces deux applications sont organisées de façon similaire. Elles contiennent les fichiers `models.py`, `serializers.py`, `api.py` et `url.py` qui sont chacun homologue à un bloc de la figure 3.9. De plus, le fichier `admin.py` configure le site d'administration (gestion de contenu) généré automatiquement par Django. Finalement le sous répertoire `./tests` contient la librairie de tests unitaires pour tester chaque

point d'accès du serveur. La dernière application, *frontend* contient le code utile au client web et est détaillé à la section 3.7.

Authentification

Tel que mentionné précédemment, le mécanisme d'authentification de l'API REST repose sur le fait que chaque requête HTTP faite au serveur est accompagnée d'un jeton qui identifie le client et qui lui accorde les permissions auxquelles il a droit. Un tel mécanisme est ce qui permet facilement d'avoir un serveur central qui peut communiquer aussi bien avec une application web, mobile ou encore embarquée. La librairie *django-rest-knox*[13] est utilisée pour l'implémentation puisqu'elle est conçue justement à cet effet (communiquer avec des clients web ou mobile) et est sécuritaire.

Limite d'accès (*Throttling*)

Afin de prévoir l'expansion de l'application, des mécanismes doivent être implantés sur le serveur pour limiter l'accès en terme de nombre de requêtes maximales par période de temps. En effet, dépendamment du nombre d'utilisateurs des limites plus sévères devront être imposées pour éviter d'engorger le serveur. À cet effet, des classes ont été implémentées dans le projet sur l'application *devices* dans le fichier `throttles.py`. Ces classes peuvent être utilisées par les points d'accès de `api.py` afin de limiter l'accès selon le niveau (*membership*) de l'utilisateur. Ceci est dans l'optique de développements futurs où l'application offrirait un système d'abonnement pour l'utilisateur afin qu'il obtienne un taux de requête plus élevé (et une limite de stockage plus élevée).

Autorisations et permissions

Les permissions sont à distinguer des autorisations. L'autorisation dépend de la validité du jeton joint à la requête et donc de la validité des authentifiants de l'utilisateur. La permission dépend des droits de cet utilisateur sur la base de données. Une requête HTTP avec un jeton invalide entraînera un statut *401-Unauthorized* tandis qu'une action tentée sur des données qui n'appartiennent pas à l'utilisateur retournera un statut HTTP *403-Forbidden*.

Le processus d'autorisation est implémenté par *django-rest-framework* avec certains mécanismes de permissions de base. Par contre, on doit parfois implémenter des classes de permissions manuellement. Ceci est fait en héritant de la classe de base BasePermission de *django-rest-framework*. Pour ce projet, de telles classes ont été implémentées pour gérer l'accès aux *devices* et limiter le stockage par utilisateur.

13. django-rest-knox, `https://james1345.github.io/django-rest-knox/`

Points d'accès

Au final, la seule partie exposée du serveur est l'API et ses points d'accès. Ceux-ci sont présentés à l'annexe A. Les méthodes permises, ainsi que les formats attendus des requêtes et des réponses y sont documentés.

Librairie de test

La librairie de test contient des tests unitaires pour chacun des points d'accès de l'API REST. Dans un contexte de développement, les tests sont indispensables. Ils permettent de valider le bon fonctionnement du programme suite à des modifications. Ainsi, cette partie du développement est automatisée. Tous développements futurs sur le backend devraient commencer par l'écriture d'une série de tests. De cette façon, l'utilisation du nouveau point d'accès est bien définie au départ et lorsque les nouveaux développements sont testés, le reste de l'API est aussi testé. On s'assure alors que tout fonctionne toujours bien.

3.6.3 Améliorations futures

Voici finalement quelques avenues qui pourraient être intéressantes à étudier pour de futurs développements sur le backend de l'application.

1. Ajouter une table à la base de données dont la fonction est de regrouper les capteurs (*devices*). En réalité, plusieurs capteurs peuvent être branchés à un seul microcontrôleur, il est donc logique de regrouper ceux-ci.
2. Dans le cas ou l'application prendrait de l'expansion, évaluer la faisabilité d'utiliser une base de données NoSQL. Pour des projets IoT d'ampleur, il se peut que ce type de structure réponde mieux aux besoins de performances.
3. Combiner la base de données à un autre service de stockage telles que les Spaces de Digital Ocean ou S3 de AWS (voir section 3.10.2). Ces services sont moins coûteux pour stocker de grandes quantités de données. Par contre, ils sont mieux adaptés pour gérer des fichiers, une partie de l'architecture du backend serait donc à retravailler.

3.7 Le frontend

Tandis que le backend est responsable de toutes les manipulations de données en arrière-plan, le frontend de l'application permet à l'utilisateur d'interagir avec le système au moyen d'une page web. Pour le serveur, le frontend ne représente en fait qu'un client comme les autres. Pour l'utilisateur, la qualité de l'entièreté de son expérience dépend de la qualité du frontend.

Le frontend développé présente à l'utilisateur des interfaces graphiques telles que :

— une page de présentation,

— des formulaires pour la connexion et l'inscription,

— des interfaces pour gérer les appareils,

— des graphiques permettant de voir les données directement sur le site web,

— des interfaces pour récupérer des données en format .csv ou les supprimer.

Cette section présente les mécanismes utilisés pour pouvoir obtenir ces fonctionnalités.

3.7.1 Développement web

Traditionnellement, trois langages sont nécessaires pour construire une page web, HTML, CSS et JavaScript. De façon générale, HTML est en quelque sorte le squelette de la page, CSS permet d'appliquer des styles aux éléments et JavaScript est le langage de programmation qui rend la page dynamique. Ce qui signifie que celle-ci peut réagir aux entrées de l'utilisateur, par exemple en se remodelant, en redirigeant l'utilisateur ou en faisant des appels (requêtes) à l'API du backend. Comme pour le backend, l'utilisation d'un framework frontend est presque indispensable étant donné qu'il permet un développement beaucoup plus rapide en intégrant HTML, CSS et JavaScript.

Pour ce projet, React [14] est le framework utilisé pour le développement du frontend. React permet de développer une *single-page application* (SPA), donc toutes les ressources du frontend sont chargées par le navigateur web lors de l'ouverture de la page. Par la suite, cette page ne fait que changer de forme à mesure que l'utilisateur y navigue sans jamais charger d'autres fichiers, rendant son expérience extrêmement rapide et fluide. React qui est développé et entretenu par Facebook est l'un des fameworks les plus populaires à ce jour avec Angular [15] (Google) et Vue.js [16].

3.7.2 Intégration avec Django

Malgré le fait que le frontend soit développé pour être indépendant du backend, les ressources statiques (le code HTML/CSS/JS compilé) de la page web doivent quand même être hébergées et servies par ce dernier. Django prévoit par défaut un mécanisme de *Views* et *Templates* pour servir le rendu des pages web à la demande du client (le navigateur web). Par contre, pour tirer avantage de l'architecture REST développée et produire une SPA rapide et fluide, on doit intégrer le frontend différemment avec Django. Plutôt que d'avoir des *Views* et *Templates* pour chaque application (on parle ici d'applications Django, voir section 3.6.2), une application indépendante dédiée uniquement au frontend est implémentée. L'arborescence des principaux fichiers de l'application est donc celle de la figure 3.11

Le fichier `url.py` présente le point d'entrée de l'application avec un chemin vide. Cela indique donc la racine de l'adresse du site web (c.-à-d. le domaine `teamatsensors.com` ou l'adresse IP

14. React, `https://reactjs.org/`
15. Angular, `https://angular.io/`
16. Vue.js, `https://vuejs.org/`

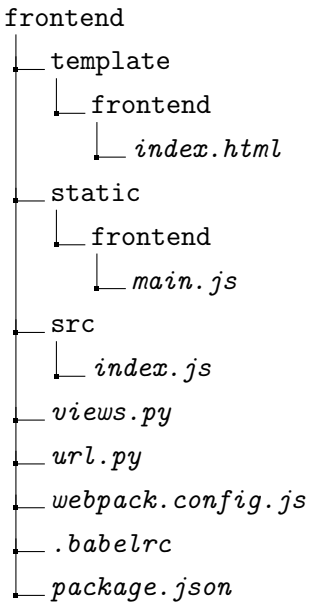


FIGURE 3.11 – Arborescence générale de l'application.

du serveur). Cette URL pointe vers une *View* définie dans `views.py` dont la seule fonction est de produire le rendu du fichier `template/frontend/index.html`. Or, ce dernier ne contient qu'un seul élément, l'application React compilée écrite dans `static/frontend/main.js`.

Le développement du code JavaScript se fait dans `frontend/src`. Le point d'entrée de ce programme est le fichier `frontend/src/index.js`. Pour compiler ce code, deux éléments sont utilisés. Le premier élément est le *bundler* statique webpack [17]. Le *bundler* prend le code dynamique contenu dans `frontend/src` et le convertit en un seul fichier d'instructions dans un format interprétable pour un navigateur web. Le deuxième élément est Babel [18]. Ce dernier n'est pas absolument essentiel, mais permet d'éviter des problèmes d'incompatibilité avec les plus vieux navigateurs web en convertissant le code vers une version plus ancienne de JavaScript. Le flux de développement classique d'une application React automatise normalement le processus de compilation. Cependant, comme on l'intègre à Django, les fichiers de configuration webpack et Babel, ainsi que les scripts de compilation doivent être définis manuellement (respectivement dans `frontend/webpack.config.js`, `frontend/.babelrc` et `frontend/package.json`).

3.7.3 Design avec React et gestion d'état

Composants React

Tout comme les autres frameworks frontend les plus populaires, le paradigme de design de

17. webpack, https://webpack.js.org/
18. Babel, https://babeljs.io/

React repose sur l'utilisation de composants (*components*). En fait, on peut s'imaginer chaque interface graphique construite avec React comme un assemblage de composants réutilisables tel qu'illustré avec les encadrés sur l'interface utilisateur de la figure 3.12.

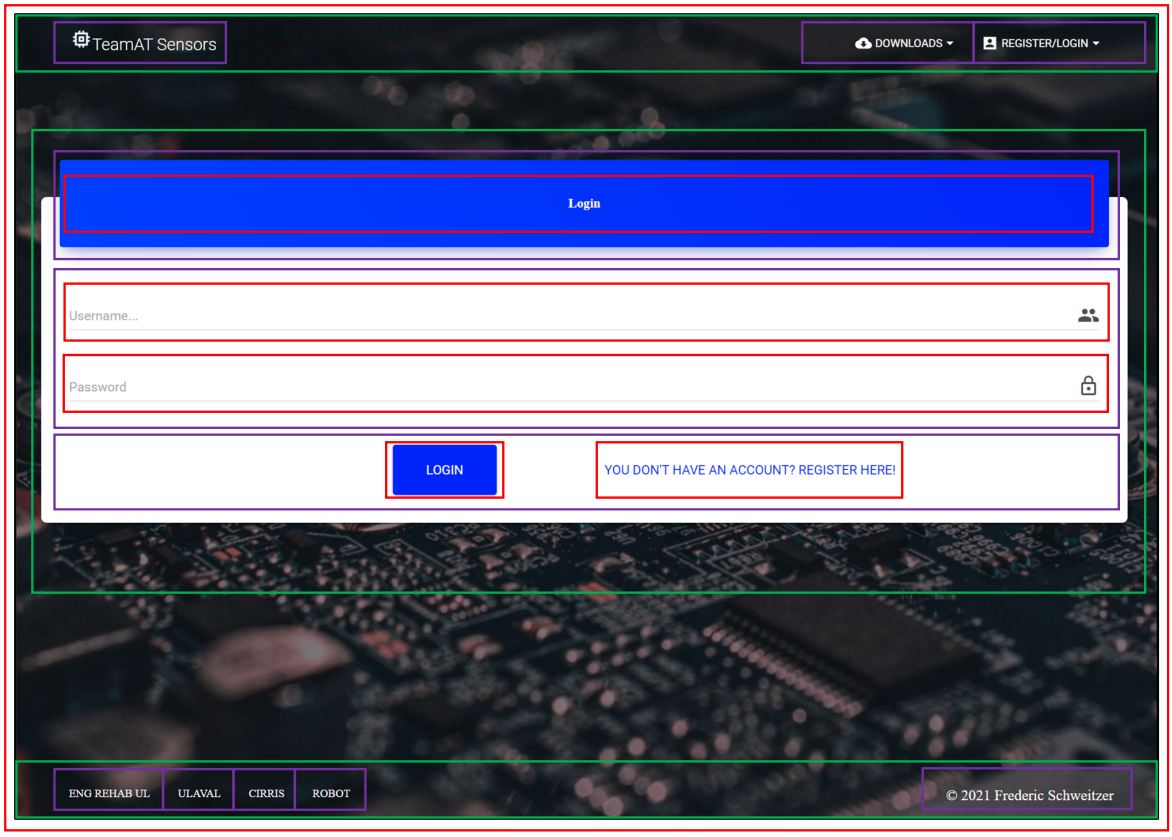


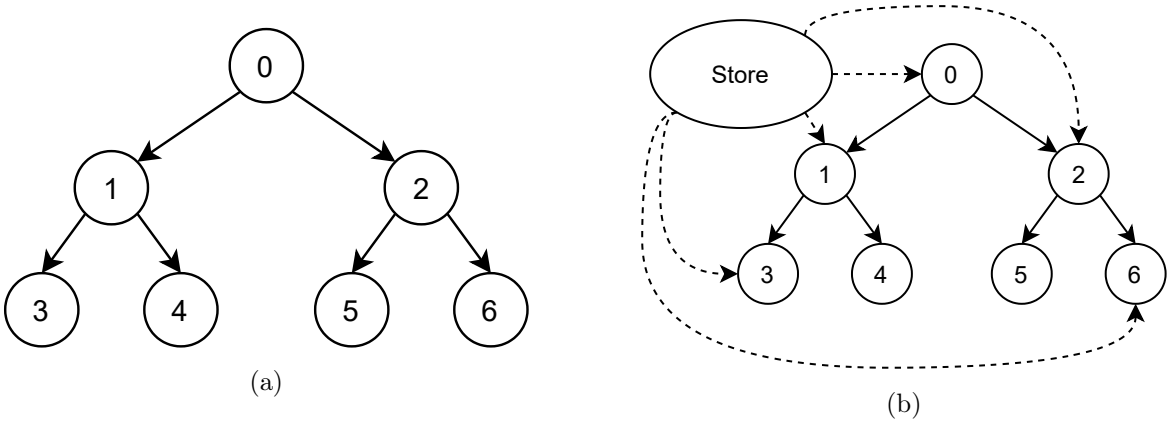
FIGURE 3.12 – Indication des composants React sur l'interface utilisateur de l'application frontend servant à connecter un usager.

Tout d'abord, cette page formulaire, utilisée pour connecter un utilisateur, correspond à un composant en elle-même. L'entête, la carte centrale et le pied de page sont des composants embarqués dans la page et comprennent eux-mêmes plusieurs autres composants comme les sections, les boutons et les zones de texte. Chaque composant a un état (*state*) et des propriétés (*props).* Les propriétés sont en quelque sorte, les arguments passés à un composant enfant lorsqu'il est créé par un composant parent. Par exemple, lorsque le pied de page de la carte instancie le bouton «LOGIN», il lui passe le texte, la forme et la couleur du bouton en propriétés. L'état d'un composant est l'ensemble de ses attributs globaux, ceux-ci sont homologues aux attributs d'un objet dans le cas d'une classe en programmation orientée objet.

Stratégies de gestion d'état

Pour plusieurs situations, les attributs d'un composant doivent être accessibles par un ou plusieurs autres composants. Généralement, il y a deux façons de rendre les états accessibles

entre les composants. La première façon est de passer les états en propriétés des composants créés. Cette façon est rapide et fonctionne bien, mais ne permet qu'aux enfants d'accéder aux états de leur parent. Tel qu'illustré à la figure 3.13a, le partage d'état ne peut se faire qu'en suivant l'arbre de création des composants. Il est possible que le composant (3) reçoive une instruction l'obligeant à changer une valeur. Par contre, si cette valeur est dans l'état du composant (2), alors le code devra être modifié et cette valeur devra être détenue à la place par le composant (0). Celui-ci la passera en propriété à (1) et (2), puis de (1) à (3). Ainsi (2) et (3) on accès à la valeur, mais seulement par l'entremise de (0).



(a) (b)

FIGURE 3.13 – Schéma de transfert d'état des composants dans une application React en passant les valeurs selon (a) l'arborescence des composants et (b) un gestionnaire d'état. Les bulles numérotées représentent les composants et les flèches représentent une possibilité de passer des valeurs de l'état d'un composant à un autre.

Une autre solution est d'utiliser un gestionnaire d'état (*state manager*). Ce gestionnaire change l'architecture de l'application pour créer un magasin (*store*), tel qu'observé sur la figure 3.13b. Le *store* contient des valeurs qui peuvent être accessibles par tous les composants. Ainsi, les états partagés ne sont plus stockés directement dans les composants, mais dans le magasin global. Le gestionnaire d'état utilisé pour cette application est Redux [19]. Pour la même situation que pour l'exemple ci-haut, Redux permet à l'application de suivre le flux de données de la figure 3.14. Les interactions avec l'interface utilisateur (GUI) (3) entraînent des actions. Les actions Redux sont en fait des fonctions qui permettent de modifier l'état du *store*. Les actions sont envoyées au *store* et le reducer s'occupe de modifier l'état selon les actions reçues (par dispatch(action)). L'état peut ensuite être récupéré par n'importe quel composant (par exemple (2)).

3.7.4 Navigation

Tel que mentionné précédemment, les applications construites avec React sont des *single-page app* ce qui signifie que toute la navigation (*Routing*) et les adresses web (URL) sont contrôlées

19. Redux, `https://react-redux.js.org/`

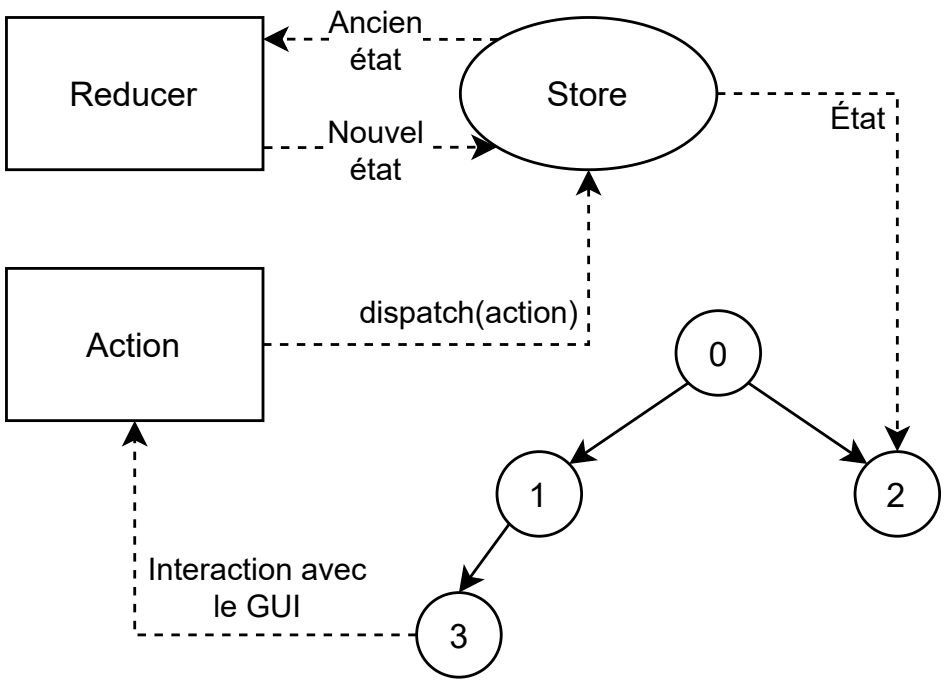


FIGURE 3.14 – Flux de données de l'application avec Redux.

par le frontend. En effet, à partir du point d'entrée défini par Django (voir section 3.7.2) tous les changements de pages subséquents sont commandés par le *HashRouter* de la librairie *React Router*[20]. La table 3.3 contient un résumé des gestionnaires de navigation selon le chemin de l'URL.

TABLE 3.3 – Gestion de la navigation selon les adresses.

Chemin	Description	Gestionnaire
/ ...	*Routing* de l'application frontend	React/*HashRouter*
/api/ ...	Points d'accès de l'API REST	django-rest-framework
/admin/ ...	*Routing* de la page d'administrateur de contenu	Django

3.7.5 Lignes directrices de design de l'interface graphique

L'introduction de ce mémoire fait mention de l'importance, pour tout développement impliquant des interactions humain-machine, de s'attarder sur l'interface si l'on souhaite que le produit ait du succès. Le développement web représente une de ces situations. Au cours des dernières années, les méthodes et les philosophies de design des pages web ont beaucoup évoluées et on retrouve aujourd'hui plusieurs librairies ou gabarits servant à faciliter le développement dans le respect de ces bonnes pratiques.

20. React Router, https://reactrouter.com/

Le frontend de l'application web de ce projet est organisé selon système Material Design de Google. Ce système mise sur les textures, les couleurs et les mouvements de l'interface graphique pour ajouter de la valeur à l'expérience utilisateur [21]. Ces principes sont implémentés avec React par la librairie Material-UI [21]. C'est donc ce qui été utilisé pour guider le développement. De plus, un gabarit de départ a été utilisé [22]. Ce gabarit en accès libre contient plusieurs composants React écrits avec Matérial-UI. Cela a servi à avoir une structure initiale pour le frontend et à accélérer le développement. Plusieurs des composants fournis ont été modifiés pour les besoins de la cause. Développer en utilisant de tels outils présente aussi un avantage supplémentaire. En effet, les composants sont conçus pour être *responsives*, c'est-à-dire qu'il s'adaptent bien à l'affichage web sur ordinateur autant que sur mobile.

3.7.6 Améliorations futures

Bien entendu, le frontend d'une application web est toujours à retravailler. Puisque c'est l'interface par laquelle les utilisateurs interagissent avec l'application, les améliorations futures de celle-ci devront refléter leurs besoins. On peut par exemple penser à plus de fonctionnalités pour personnaliser la visualisation des données avec les graphiques ou bien une interface plus intuitive pour pouvoir récupérer les données désirées.

3.8 Développement embarqué

Le développement embarqué pour ce projet s'est fait sur le microcontrôleur ESP32. Les détails techniques du module de développement utilisé sont disponibles en [22]. Une librairie Arduino a été développée [23] pour communiquer avec le serveur. Le fait de développer à l'aide du framework Arduino accélère grandement le développement et permet à plus de développeurs désirant améliorer le produit, ou simplement y avoir accès pour le modifier et le rendre sur mesure à leur projet, de le faire.

3.8.1 Électronique

Le microcontrôleur ESP32 a été choisi pour le développement, car il intègre des fonctionnalités de connexion sans fil (WiFi et Bluetooth) en plus d'avoir un processeur 32-bits à deux coeurs 240 Mhz qui fournit amplement de puissance de calcul pour la collecte de données et plus encore (prétraitement, analyse de données directement sur le système embarqué). Le module ESP32 est normalement alimenté par une prise USB. Comme on souhaite l'utiliser pour des études en réadaptation, il est très probable qu'il soit porté par une personne plutôt que d'être sédentaire près d'une prise de courant murale. Pour remédier à ce problème, l'utilisation d'une pile rechargeable LiPo et d'un module intégré de chargement est requise. La figure 3.15 présente le schéma de branchement de ces composantes.

21. Material-UI, `https://material-ui.com/`
22. Gabarit Creative Tim, `https://github.com/creativetimofficial/material-kit`

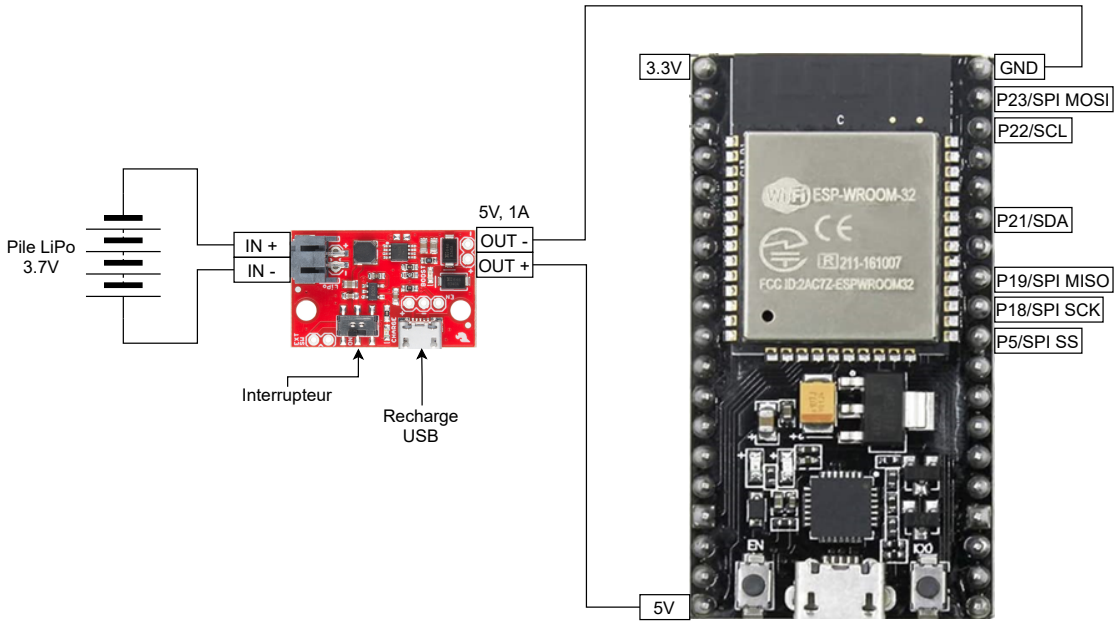


FIGURE 3.15 – Schéma de branchements de la pile LiPo pour alimenter le ESP32.

Le module de chargement choisi [23] agit aussi comme régulateur de tension pour l'ESP32 assurant d'avoir toujours une tension nominale de 5V à la sortie.

3.8.2 Librairie Arduino

La librairie Arduino développée dépend de deux autres librairies externes. L'une d'elles est l'interface développée par espressif, la compagnie responsable du développement des modules ESP. Elle est nécessaire pour utiliser l'ESP32 avec le framework Arduino. Ensuite, la librairie ArduinoJson [24] est utile pour la manipulation de données en format JSON.

Utilisation

La première étape pour utiliser la librairie TeamAT-Sensor est de l'installer et d'installer ses dépendances. Il suffit de suivre les instructions d'espressif pour configurer l'ESP32 pour pouvoir le programmer en Arduino. La librairie ArduinoJson s'installe de façon triviale. Il est aussi nécessaire d'avoir préalablement créé un *device* avec la page web de l'application. En faisant cela, on obtient un numéro d'identification *ID* du *device*. Cet *ID* doit être entré manuellement dans le code Arduino pour identifier les capteurs.

La librairie présente deux exemples. La différence entre les deux est la façon d'entrer les identifiants réseau et les informations de connexion pour le serveur. Pour un premier exemple, ces informations sont entrées manuellement dans le code

23. SparkFun LiPo Charger/Booster - 5V/1A, https://www.sparkfun.com/products/14411
24. ArduinoJson, https://github.com/bblanchon/ArduinoJson

```
logger.begin("YOUR_WIFI_SSID", "YOUR_WIFI_PWD");
logger.login("your_account_username", "your_account_password");
```

et pour l'autre, elles sont saisies avec une application mobile et transmise au ESP32 via Bluetooth.

```
logger.begin();
logger.login();
```

Pour envoyer des données au serveur, il fait créer un objet TatLogger prenant en argument le nombre de capteurs connectés au microcontrôleur

```
TatLogger logger(NBR_OF_SENSORS);
```

puis, on crée un tableau de ses capteurs, ceux-ci sont identifiés par leur *ID*.

```
TatSensor sensorArray[NBR_OF_SENSORS] =
    {TatSensor(FIRST_DEVICE_ID), TatSensor(SECOND_DEVICE_ID)};
```

L'envoi des données est géré par la méthode smartLog. Celle-ci envoie à la bonne fréquence (selon les configurations de *throttling* du serveur) les données temporairement enregistrées au serveur.

```
sensorArray[0].saveData(random(10), logger.getDatetime());
sensorArray[1].saveData(random(10), logger.getDatetime());
logger.smartLog(sensorArray);
```

Ici, saveData prend en argument la donnée et le point temporel de l'acquisition. L'horloge pour cette information est gérée par l'objet TatLogger.

Fonctionnement WiFi

La librairie Arduino est composée de quatre classes principales dont les responsabilités sont résumées à la table 3.4. La fonction principale de la librairie est d'envoyer les données des capteurs au serveur, bien qu'elle offre aussi la possibilité de diffuser ces données en temps réel à un téléphone intelligent via Bluetooth Low Energy (BLE).

En premier lieu, le Collecteur est responsable d'initier la connexion au réseau WiFi, ce mécanisme est implémenté dans la méthode begin. Si la méthode est appelée sans argument, le programme s'attendra à recevoir les informations de connexion via Bluetooth. Une fois la connexion établie, la méthode login du Collecteur permet à l'ESP32 de demander un jeton de connexion au serveur. Encore une fois, si cette méthode est appelée sans arguments, le programme en attendra par Bluetooth. Le jeton ainsi que toutes les informations requises pour les authentifications sont enregistrés dans une partition de mémoire non volatile EEPROM

TABLE 3.4 – Principales fonctionnalités des classes de la librairie Arduino.

Objet	Nom de la classe	Fonctions
Collecteur	TatLogger	Gère l'envoie de données au serveur ou à l'app. mobile
Capteur	TatSensor	Gère les données d'un seul capteur.
Horloge	TatSensorDateTime	Gère l'horloge et les points temporels des données
Serveur BLE	TatBTParser	Reçoit et envoie les données via Bluetooth Low Energy (BLE)

(comme la mémoire flash). L'ESP32 peut contenir 512 octets de mémoire EEPROM ne s'effaçant pas lorsque le module est débranché. Cela permet de récupérer le jeton à la mise sous tension de l'appareil ou encore de le renouveler facilement lorsqu'il s'invalide. La structure des données d'authentification conservée dans la mémoire EEPROM est présentée à la figure 3.16. Elle est simplement constituée de combinaisons répétées d'un octet, indiquant la longueur du mot qui le suit, puis du mot en question.

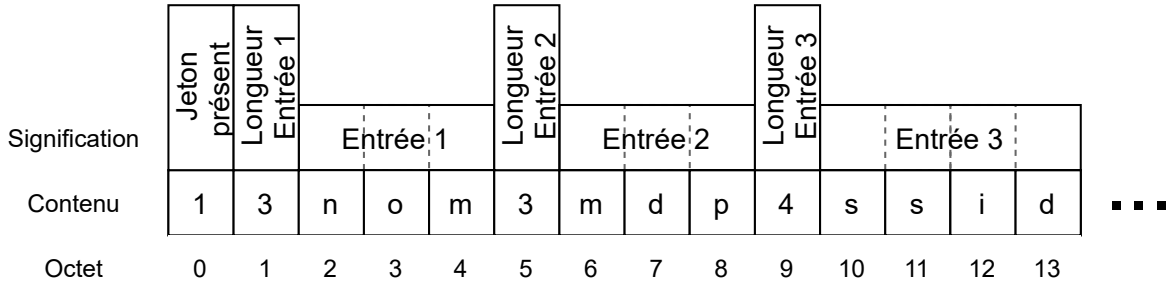


FIGURE 3.16 – Structures des octets pour l'enregistrement de données en mémoire flash.

Le Colecteur utilise plusieurs Capteurs regroupés sous forme de tableau dans un de ses attributs. Lorsque la méthode saveData du Capteur est appelée, une donnée jumelée avec son information temporelle est stockée dans un tableau. Celui-ci est propre à l'objet. Le Collecteur analyse ensuite les tableaux de données des Capteurs et décide quand les envoyer au serveur avec la méthode smartLog. En effet, si l'acquisition se fait à 10 Hz, il n'est pas avantageux ni pour le client ni pour le serveur de gérer 10 requêtes HTTP par seconde. L'approche à privilégier sera plutôt d'attendre et d'envoyer plusieurs données en même temps en une seule requête.

L'Horloge gère le temps dans le programme. Au démarrage du microcontrôleur, elle envoie une requête au serveur NTP `pool.ntp.org` pour synchroniser la date et l'heure puis le temps est par la suite calculé à partir de simples compteurs liés à des interruptions régulières du ESP32. Que ce soit pour assigner un point temporel à une donnée ou pour démarrer un chronomètre, c'est l'Horloge qui en est responsable.

Finalement le Serveur BLE est utilisé pour deux actions distinctes dans le programme,

1. recevoir les informations de connexion (waitForCredentials),
2. diffuser un flux de données pouvant être reçu par une application mobile par exemple (stream).

Lorsque le programme utilise une de ces deux fonctionnalités, le Collecteur utilise un Serveur BLE. Ce dernier s'occupe principalement des actions bas niveau pour établir les communications et déchiffrer les données reçues.

Bluetooth Low Energy

La communication Bluetooth conventionnelle n'est en quelque sorte qu'une communication sérielle ordinaire, mais sans fil. Deux appareils sont connectés ensemble et s'échangent des données. Le protocole BLE est légèrement différent. Dans ce cas, un appareil est un serveur et l'autre est un client. Le serveur diffuse des données à l'aide d'une structure de données GATT (*Generic Attribute Profile*). Celle-ci est illustrée à la figure 3.17.

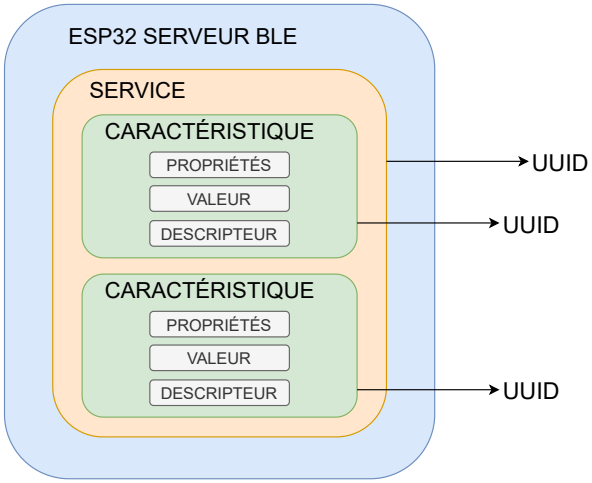


FIGURE 3.17 – Structure de données GATT pour la diffusion de données via BLE (adapté de [24]).

L'ESP32 agissant comme serveur BLE affiche un ou plusieurs services regroupant une ou plusieurs caractéristiques. Ces dernières sont en fait, les données qui peuvent être accompagnées de propriétés et de descripteurs. Chacun de ces services et de ces caractéristiques est identifié par un numéro d'identification universel unique (UUID). Le client (ex. application mobile) se connecte donc à un service d'un appareil pour en extraire les valeurs des caractéristiques. Celui-ci doit donc connaître ce numéro s'il veut obtenir les données. Ces numéros sont écrits dans le code du client et du serveur.

3.8.3 Quelques notes

Voici diverses précisions concernant la librairie embarquée.

1. Les librairies WiFi et Bluetooth (BT et BLE) sont lourdes. Si on souhaite les utiliser en même temps, on doit modifier les partitions de mémoire du ESP32. Pour ce faire, il suffit d'ouvrir l'IDE Arduino, d'aller dans le menu Tools, puis Partition Scheme et de sélectionner minimal SPIFFS. Cela assigne plus de mémoire pour le code (partitions app1, app0) au détriment de partition SPIFFS. Cette dernière correspond à une portion de mémoire FLASH externe utilisée pour stocker des fichiers.
2. Il est laborieux de faire fonctionner le WiFi et le Bluetooth du microcontrôleur simultanément puisque ceux-ci utilisent la même antenne. Pour envoyer une requête HTTP au serveur, les fonctionnalités Bluetooth doivent être désactivées.
3. La communication HTTPS (cryptée) est possible avec l'ESP32. Comme ce dernier est un client du serveur, il ne semble pas nécessaire d'ajouter le certificat SSL (voir section 3.10.2) au code. Ajouter le certificat permettrait au programme Arduino de valider son authenticité et donc de valider la source (le serveur). Il serait ainsi possible de vérifier que le serveur est vraiment celui souhaité et non pas un faux. Par contre, ce certificat s'invalide fréquemment. De plus amples recherches sur ce sujet pourraient être réalisées pour de futurs travaux.
4. Les objets Capteur renferment un tableau de données qui agit comme une zone tampon. Il pourrait être intéressant de substituer celle-ci avec un fichier sur une carte SD connectée au ESP32. De plus, une méthode similaire à smartLog pourrait être implémentée au niveau du Collecteur pour envoyer ses données au serveur selon une indication d'un utilisateur à la manière d'une montre de type Fitbit (voir section 3.4.2).
5. Bien que le protocole BLE soit utilisé pour transmettre les données d'authentification, une communication BT classique serait peut-être plus appropriée, et plus facile et logique à implémenter. À cet effet, des mécanismes de sécurité devraient être implémentés si l'on souhaite protéger les données transmises (ex. cryptage, limiter le nombre de connexions).

3.9 Application mobile

Jusqu'ici, les mécanismes développés permettent à un utilisateur d'utiliser un capteur quelconque, de le brancher sur un microcontrôleur ESP32, de récolter des données sans fil avec une connexion Wifi et de récupérer ou visualiser celles-ci à l'aide d'une application web. La valeur ajoutée de l'application mobile dans l'écosystème est premièrement d'avoir un aperçu des données à portée de main. Ensuite, elle permet de configurer un ESP32 pour qu'il puisse se connecter au réseau et au serveur. Finalement, l'application mobile comporte aussi une fonction pour recevoir des données directement de l'ESP32 via une diffusion Bluetooth Low Energy.

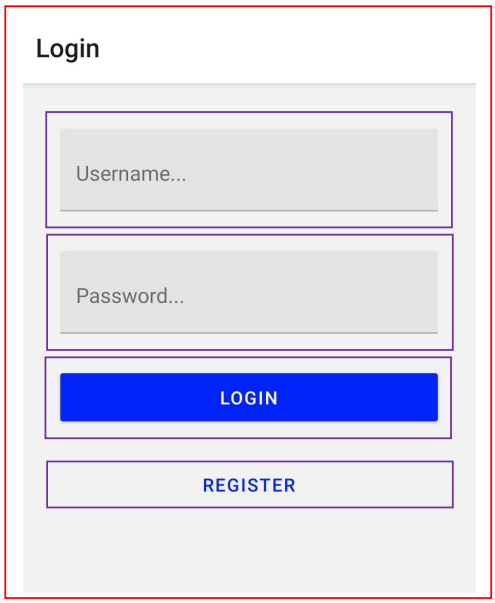


FIGURE 3.18 – Indication des composants React sur l'interface utilisateur de l'application mobile servant à connecter un usager.

3.9.1 Technologies utilisées

On retrouve trois paradigmes dans l'industrie pour le développement mobile. Premièrement, il y a bien sûr le développement natif. Ce qui signifie que la programmation des applications se fait directement avec les outils Google/Android (Kotlin) et Apple/iOS (Swift) et qu'il y a donc une base de code pour chaque plateforme. Pour sauver du temps, des frameworks multiplateformes peuvent être utilisés. Parmi les plus populaires en 2020, on retrouve React Native [25] et Flutter [26] [18]. Le dernier paradigme de développement est appelé *progressive web apps* [25]. Ces applications sont en fait des applications web répondant à certains critères qui font en sorte que l'utilisateur à réellement l'impression d'utiliser une application native en les utilisant avec son téléphone intelligent.

Le choix de technologie pour ce projet s'est orienté vers React Native. Ce framework utilise les bases de React pour construire des applications mobiles et est aussi supporté par Facebook. Étant donné que l'application web est déjà développée avec React, ce choix fut évident. Cela permet aussi de développer les applications Android et iOS avec une seule base de code.

Accès au serveur et structure

Le principe de composants React illustré à la figure 3.12 et le même pour React Native. L'application navigue d'une page à l'autre et affiche les composants qui y sont imbriqués. Ceux-ci renferment des valeurs d'état et des propriétés. Les styles changent, mais le principe

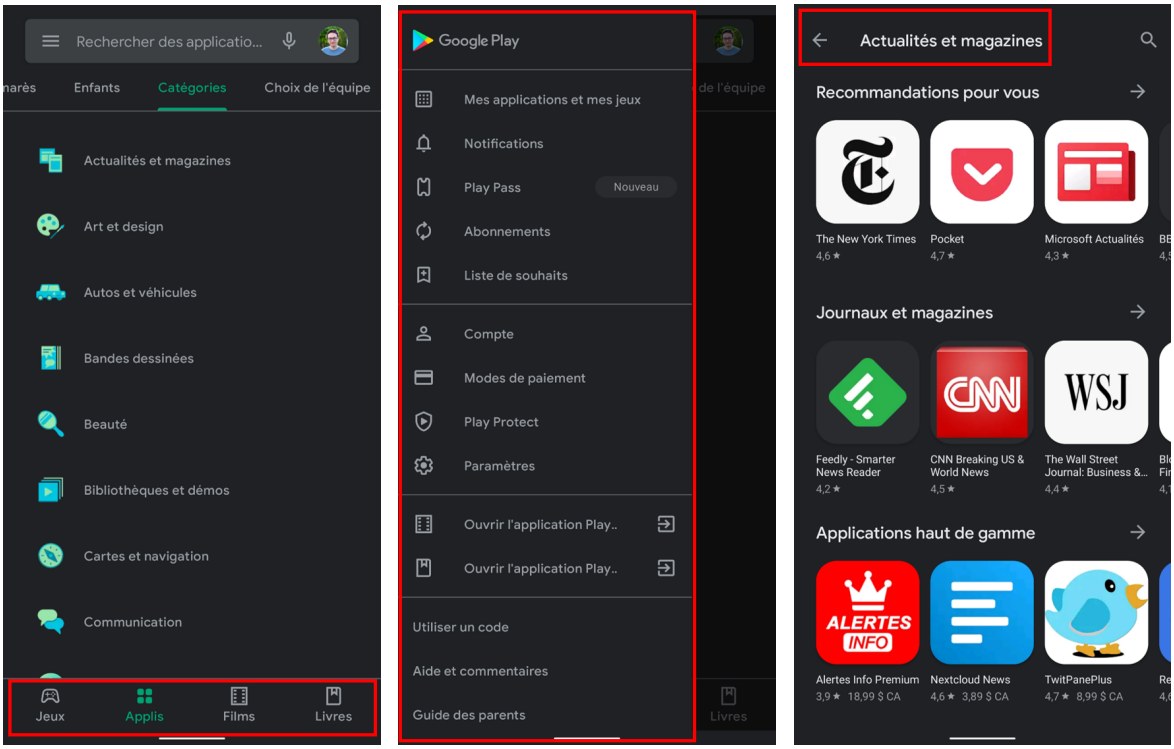
25. React Native, https://reactnative.dev/
26. Flutter, https://flutter.dev/

reste. La figure 3.18 présente à nouveau le formulaire de connexion en indiquant les composants, mais cette fois-ci pour l'application mobile.

Le serveur offrant une API REST, la procédure d'accès aux données serveur n'est pas tellement différente pour l'application mobile que pour l'application web. En effet, l'application envoie des requêtes HTTP au serveur et traite les réponses comme n'importe quel client. Pareillement, toute autre librairie JavaScript ne faisant pas intervenir la structure d'une page web (*Document Object Model*, DOM) peut être utilisée pour développer l'application mobile.

Navigation

La navigation pour une application mobile est un élément majeur qui diffère en comparaison à une application web. Plutôt que d'utiliser des URLs et des routes pour naviguer, une application mobile a plus tendance à faire usage d'onglets (*tabs*), de tiroirs (*drawers*) ou d'empiler les pages (*stack*) tels qu'observés sur la figure 3.19. Les onglets sont présentés au bas de l'écran et l'utilisateur peut naviguer d'une page à l'autre en les sélectionnant. Le tiroir est accédé à l'aide d'un bouton placé dans le coin supérieur gauche de l'écran ou en le «tirant» de la bordure de l'écran. Finalement, pour permettre à l'utilisateur d'accéder successivement à plusieurs pages suivant un flux logique, l'application empile celles-ci et présente seulement



(a) (b) (c)

FIGURE 3.19 – Exemples des éléments de navigations sur l'application Play Store de Google : les onglets en (a), le tiroir en (b) et la pile en (c).

à l'utilisateur, l'option de revenir en arrière (dépiler).

Pour l'application développée, la librairie React Navigation [27] est utilisé pour implémenter la navigation avec les onglets et les piles. Les nouvelles versions d'Android et iOS permettent à l'utilisateur de faire des *gestures* pour utiliser son téléphone (passer d'une application à l'autre, retour en arrière, afficher l'écran d'accueil, etc.), or celles-ci peuvent entrer en conflit avec l'action de «tirer» le tiroir. Ainsi, ce dernier n'a pas été utilisé. La figure 3.20 présente le schéma de navigation de l'application.

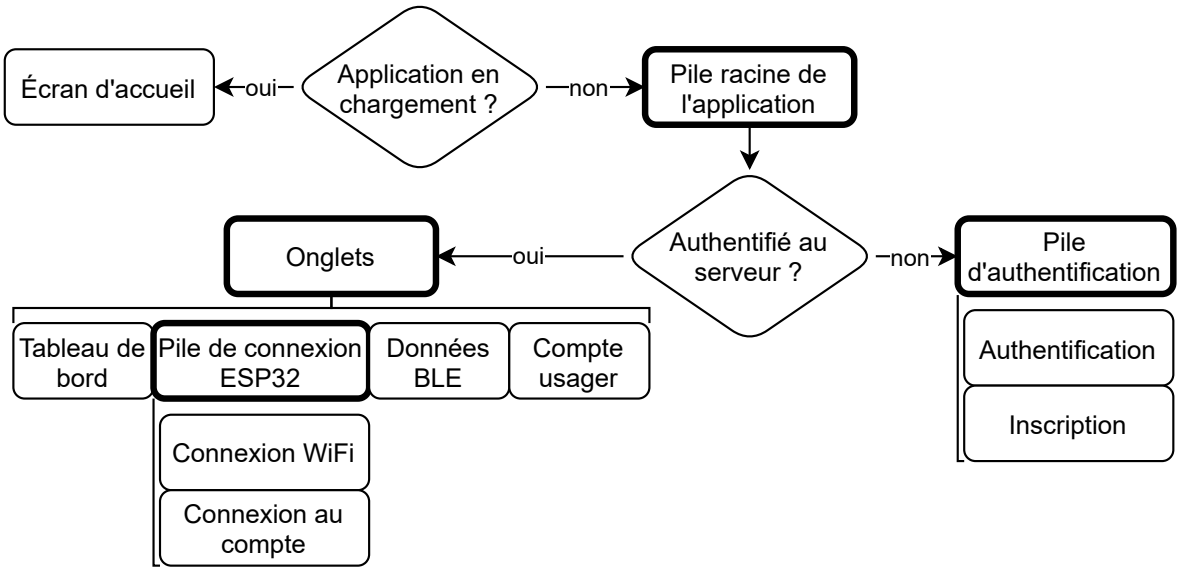


FIGURE 3.20 – Schéma de navigation de l'application mobile. Les encadrés en gras représentent des ensembles de vues, les encadrés simples sont les vues (pages) et les losanges sont des conditions à vérifier par le programme.

Le point d'entrée de l'application est l'écran d'accueil (*splash screen*). Celui-ci s'affiche pendant que l'application se charge. Une fois chargée, elle affiche le formulaire d'authentification si elle est incapable de trouver des authentifiants ou un jeton valide dans la mémoire du téléphone intelligent. Si la connexion est validée, les onglets s'affichent et l'utilisateur peut y naviguer.

Gestion d'état et de données persistantes

Tout comme pour l'application frontend les échanges de données entre les composants ont souvent besoin d'être gérés par un gestionnaire d'état. Dans ce cas-ci, on utilise la *Context API* incluse avec React plutôt que Redux. Celle-ci fonctionne de façon similaire, avec des Actions et Reducers. Cependant, la notion de Store est remplacée par celle de Context qui est moins globalisée (c-à-d qu'on utilise différents Contexts pour avoir accès à certaines Actions/Reducers au besoin plutôt que d'avoir un Store qui renferme tout).

27. React Navigation, https://reactnavigation.org/

De plus, certaines données ont besoin de rester en mémoire pour simplement faciliter l'utilisation et ces données sont souvent sensibles. Par exemple, lorsqu'un utilisateur ouvre l'application de son réseau social favori, il n'entre pas son nom et son mot de passe à toutes les fois. Pour stocker sécuritairement ces données Android et iOS prévoient des systèmes pour les crypter et les rendre accessibles pour l'application seulement avec une structure *key-value* (comme le format JSON), respectivement le Android Keystore et Keychain Sercives. Avec React Native, ces systèmes sont accessibles avec la librairie *react-native-sensitive-info*[28] et c'est ce qui a été utilisé à cet effet.

3.10 Déploiement, développement et opérations

L'ensemble de l'application web incluant le backend et le frontend peut fonctionner dans deux environnements différents.

1. Environnement de développement : fonctionne localement, utilisé pour développer et apporter des modifications à l'application.
2. Environnement de production : déploie une version stable de l'application sur une machine distante, fonctionne en continu pour que les utilisateurs y aient accès.

Les sections suivantes présentent la façon de faire pour configurer un environnement de développement, puis les étapes de déploiement de l'application dans un environnement de production sur un serveur Linux et finalement le flux de travail pour pouvoir pousser les modifications apportées en développement jusqu'à l'environnement de production.

3.10.1 Environnement de développement

L'idée générale de l'environnement de développement est d'avoir une version locale simplifiée de l'application pour pouvoir y apporter des modifications et de les tester rapidement. En effet, certains éléments sont différents par rapport à la version de production. Premièrement, le DBMS est différent. Comme la performance n'est pas un enjeu pour le développement, plutôt que d'utiliser PostgreSQL, on utilise SQLite. Utiliser PostgreSQL nécessiterait une installation supplémentaire sur l'ordinateur local en plus de devoir gérer celle-ci indépendamment (faire fonctionner en arrière-plan, avoir les bonnes tables, les bons authentifiants, etc.). À l'opposé, SQLite vient par défaut avec Django et est intégrée dans un seul fichier, créé automatiquement avec l'initialisation de l'environnement.

Le fichier `settings.py` permet de configurer l'environnement. Ce fichier contient les variables booléennes REMOTE et DEBUG. Les valeurs `False` et `True` doivent être assignées respectivement à ces deux variables pour signifier le mode développement. Le fichier de variables d'environnement `.env` est un fichier qui contient les valeurs contextuelles de ces variable, ainsi que des données sensibles telles que les mots de passe. Utiliser des variables d'environnement

28. react-native-sensitive-info, https://mcodex.dev/react-native-sensitive-info/

facilite le passage d'un environnement de travail à un autre (développement/production) et protège les données, puisqu'elles ne sont présentes que localement.

On utilise aussi un serveur local plutôt que distant. Django inclut le fichier `manage.py`. Celui-ci contient des scripts utilisés pour certaines manipulations de l'application. Par exemple, il permet de synchroniser la base de données, créer des administrateurs, démarrer le serveur de développement et faire d'autres actions utiles pour le développement et le déploiement. À l'aide d'un terminal ouvert dans le dossier contenant `manage.py`, les commandes

```
$ py manage.py makemigrations
$ py manage.py migrate
```

permettront de synchroniser la base de données et les *Models* Django. Si le fichier `sqlite3.db`, qui contient la base de données, n'est pas déjà présent dans le projet, il sera créé automatiquement. Ensuite, le serveur de développement peut être démarré avec la commande suivante.

```
$ py manage.py runserver 0.0.0.0:8000
```

L'adresse 0.0.0.0 est l'adresse de l'ordinateur sur le réseau local et 8000 représente le port sur lequel l'application est accessible. Utiliser 0.0.0.0 plutôt que *localhost* ou 127.0.0.1 permet aux appareils connectés au même réseau local (LAN), comme un microcontrôleur ESP32, d'avoir accès au serveur.

On peut donc accéder à l'application à l'adresse `localhost:8000` sur tous services fonctionnant sur l'ordinateur (comme un navigateur web) et à l'adresse `ip-hote:8000` sur tout autre service connecté au LAN (comme un ESP32). La valeur *ip-hote* représente l'adresse IP de l'ordinateur sur lequel roule le serveur.

Le dernier élément requis est de faire rouler le script de développement pour le frontend. La commande

```
$ npm run dev
```

à partir d'un terminal ouvert dans le fichier d'application `frontend`, appelle le script *dev* définit dans `package.json`. Ceci permet à la fois d'afficher les messages d'erreur à la console et de rafraîchir automatiquement l'application. En effet, lorsqu'une modification sera apportée au programme du frontend, celui-ci sera recompilé à chaque sauvegarde et il sera possible de la visualiser en actualisant le navigateur web.

3.10.2 Déploiement sur serveur Linux

Le déploiement d'une application web telle que celle développée dans ce chapitre passe par plusieurs étapes et tout comme pour les autres éléments, plusieurs choix s'offrent pour ce qui est des technologies à utiliser. On doit d'abord choisir un fournisseur de service *cloud*

pour héberger l'application. Un service bien connu est *Amazon Web Services*[29] (AWS). Peu importe le fournisseur choisi, le principe est toujours le même, on crée une machine virtuelle (Linux dans la majorité des cas) sur le site web du fournisseur. Lorsque c'est fait, celui-ci donne accès l'adresse IP de la machine. Avec cette adresse, on peut commander le serveur à distance. Le choix du fournisseur pour ce projet s'est plutôt arrêté sur Digital Ocean[30]. Ce dernier fonctionne par-dessus les services d'AWS avec des performances similaires, mais offre une interface plus épurée et facile à utiliser.

La documentation de Digital Ocean comprend des guides détaillés pour tout le processus de déploiement. Cette section présente plutôt les éléments qui y sont impliquées et les étapes générales. Tout d'abord, la table 3.5 résume les fonctions de ces éléments.

TABLE 3.5 – Éléments principaux impliqués dans le déploiement de l'application web.

Élément	Définition	Fonction
Digital Ocean	Fournisseur de services *cloud*	Héberge un serveur virtuel Linux
Ubuntu	Distribution Linux	Système d'exploitation sur le serveur virtuel
NGINX	Serveur web, *reverse proxy*, *load balancer* et cache HTTP	Interface entre Linux et Internet
Gunicorn	*Web Server Gateway Interface* (WSGI) Python	Interface entre NGINX et Python/Django
Django	Framework de développement web	Facilite la programmation du comportement de l'application, spécifiquement le backend
PostgreSQL	Base de donnée et DBMS SQL	Contient les données nécessaires au fonctionnement de l'application
Let's Encrypt	Fournisseur de certificats SSL/TLS	Permet de crypter les échanges via HTTPS

Configuration initiale

Le serveur Linux n'offre pas d'interface graphique, mais plutôt une interface de ligne de commande. Pour y accéder, on doit passer par la console sécurisée SSH. On peut utiliser SSH à l'aide de logiciel comme PuTTY[31] sur Windows ou la commande *ssh* pour les systèmes Mac et Linux. Le principe est d'authentifier un ordinateur auprès du serveur distant à l'aide d'une clé ou d'une combinaison usager/mot de passe. Ainsi, on a accès au terminal du serveur distant sur cette machine locale.

La première connexion au serveur Ubuntu est faite en tant que l'utilisateur *root*. Travailler en tant que *root* n'est pas recommandé puisque les privilèges qui y sont reliés peuvent entraîner des erreurs irréparables. La première chose à faire était donc de créer un utilisateur, de lui

29. AWS, https://aws.amazon.com/
30. Digital Ocean, https://www.digitalocean.com/
31. PuTTY, https://www.putty.org/

procurer les bons privilèges administratifs et de se reconnecter en tant que ce nouvel utilisateur. Ensuite, un pare-feu devait être mis en place pour permettre seulement une connexion SSH. Les détails techniques de ces actions sont présentés par [26].

Configurer la base de données

Une fois la machine virtuelle bien configurée et sécurisée, la base de données PostgreSQL est prête à être configurée. Les étapes de configurations sont les suivantes :

1. Installer les paquetages Linux (libpq-dev, postgresql, postgresql-contrib).
2. Se connecter à PostgreSQL,
3. Créer une base de données,
4. Créer un utilisateur (et un mot de passe, le nom d'utilisateur PostgreSQL doit être le même que celui d'Ubuntu),
5. Donner les privilèges d'administrateur sur la nouvelle base de données au nouvel utilisateur.

Aucune autre action n'est nécessaire pour configurer la base de données. À partir de ce point, c'est Django qui crée les tables et qui exécute toutes les modifications subséquentes. Tous les détails et commandes à exécuter pour configurer la base de données, Django et le serveur sont présentés par [27]

Créer un environnement Python

L'objectif de créer un environnement Python est de faciliter l'installation de paquetages et de contrôler l'accès à ceux-ci pour garder un système organisé. C'est une couche supplémentaire d'encapsulation pour l'application. Python 3 est déjà installé sur Ubuntu, il a suffi d'installer *pip*, le gestionnaire de paquetages de Python. On utilise ensuite *pip* pour installer *virtualenv* qui permet de créer un environnement virtuel dans lequel installer les paquetages nécessaires à l'application Django. Un dossier de projet `sensorproject` à d'abord été créé. Dans ce dossier l'environnement virtuel *sensorenv* est créé et activé.

Importer les codes et configurer Django

Le principe pour transférer les programmes développés localement vers le serveur virtuel est d'utiliser le gestionnaire de version Git [32]. On pousse (*push*) le code depuis l'environnement de développement vers un répertoire distant Github [33], puis on ramène (*pull*) le code sur la machine virtuelle.

32. Git, `https://git-scm.com/`
33. Github, `https://github.com/`

Avant de pousser le code vers le répertoire distant, le code JavaScript du frontend doit être compilé (assemblé en un fichier, `main.js`) en mode production. Il suffit d'ouvrir un terminal dans le dossier `frontend` et d'exécuter,

```
$ npm run build
```

Une fois ce préalable exécuté le code est poussé sur Github.

Sur le serveur, l'environnement git est initialisé et le code y est ramené. À ce point, l'arborescence de répertoire *home* (∼) est celle de la figure 3.21. Il ne reste donc qu'à configurer Django.

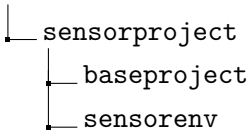


FIGURE 3.21 – Arborescence de répertoire *home* du serveur Ubuntu après avoir créé le projet.

Cette configuration s'est faite à l'aide du fichier `baseproject/manage.py` qui contient tous les scripts nécessaires au développement et déploiement. Les informations des Models pour configurer les tables de la base de données sont d'abord collectées,

```
$ python3 manage.py makemigrations
```

puis ces informations (migrations) sont envoyées à la base de données,

```
$ python3 manage.py migrate
```

Un utilisateur administrateur est créé,

```
$ python3 manage.py createsuperuser
```

et finalement les données statiques comme l'application React sont compilées dans le dossier défini par `settings.py`,

```
$ python3 manage.py collectstatic
```

Configurer le serveur web

Le serveur web permet de gérer efficacement les requêtes HTTP (*load balancing*, cache HTTP) et de servir le code effectif à un client. Pour ce projet, deux technologies sont utilisées à cet effet, NGINX [34] et Gunicorn [35]. Le premier étant le serveur web en tant que tel, il sert d'interface entre Internet et la machine Linux. Le second quant à lui est une interface (WSGI) entre

34. NGINX, https://www.nginx.com/
35. Gunicorn, https://gunicorn.org/

Python et NGINX. Il sert le programme de l'application pour ce dernier. Pour l'environnement de production, Gunicorn remplace en quelque sorte le serveur de développement. Théoriquement, il serait possible de lier le programme Django à un port réseau de la machine Linux avec la WSGI et d'y accéder avec la paire *addrese-ip :port*. Par contre, outre les problèmes de sécurité que cela engendrerait, le trafic de requête ne pourrait pas être géré correctement. C'est d'ailleurs la raison pour laquelle l'utilisation d'un serveur web comme NGINX est requise.

On doit plutôt configurer Gunicorn comme un service qui est utilisé par NGINX. Les configurations de ces deux éléments se font en modifiant certains fichiers de configuration. De plus, on doit permettre l'accès à NGINX à travers le pare-feu. Le processus est détaillé dans [27].

Domaine et connexion sécurisée

Le domaine est l'adresse web. Il remplace l'adresse IP et évite au serveur d'exposer celle-ci directement. Le domaine pour cette application est lié à l'adresse IP avec l'interface de Digital Ocean. Ce domaine doit être indiqué dans les fichiers de configuration de NGINX et le fichier `setting.py` de Django.

Le port standard pour toutes requêtes HTTP faites au serveur est 80. L'accès à ce port est permis lorsque les permissions pour NGINX sont ajoutées au pare-feu. Comme c'est le port standard, il permet d'accéder au serveur avec l'adresse IP (ou le domaine) seulement (sans :80 suivant l'adresse). Cependant, le protocole HTTP via le port 80 ne permet pas une connexion sécurisée. Les données transmises sont en format texte, lisible pour quiconque intercepte la communication. Pour crypter les données, on doit utiliser le protocole HTTPS dont le port par défaut est 443. À cet effet, le programme automatisé Certbot est utilisé pour obtenir un certificat SSL de *Let's Encrypt*[36] et le renouveler aux 90 jours. De plus, il est encore une fois nécessaire de permettre la connexion à travers le pare-feu. Les indications détaillées sont expliquées dans [28] et les fichiers de configuration finaux pour NGINX et Gunicorn sont présents la l'annexe B.

3.10.3 Flux de travail du développement à la production

Le code déployé (dans l'environnement de production) et le code de développement doivent rester synchronisés. Pour ce faire, un mécanisme ou une méthode de travail est nécessaire. Le système de gestion de révision Git et le service d'hébergement Github sont les outils à privilégier pour une telle tâche. Ceux-ci gèrent le code de façon à avoir une copie centrale qui pourra être utilisée par différentes machines locales pour le développement et par le serveur pour le déploiement. L'idée générale du flux de travail est de

1. Modifier le code dans l'environnement de développement,
2. Pousser les modifications sur le répertoire distant (Github),

36. Let's Encrypt, `https://letsencrypt.org/`

3. Ramener (*pull*) les modifications sur le serveur depuis le répertoire distant,
4. Répéter.

Concrètement, voici les étapes à réaliser pour obtenir une méthode de travail fluide. Pour configurer un nouvel environnement de développement, il faut ramener le code depuis Github, puis configurer Python au besoin. Dans un dossier de travail vierge, on doit ouvrir un terminal et suivre les étapes suivantes.

1. Initialiser l'environnement git
   - Avoir git installé sur l'ordinateur, connaître l'adresse du répertoire Github.
   - Initialiser le répertoire de travail local

```
$ git init
$ git pull adresse-du-repertoire
$ git remote add adresse-du-repertoire
```

2. Initialiser l'environnement Python (Windows)
   - S'assurer d'avoir Python et pip installés sur l'ordinateur (varie selon les systèmes d'exploitation, avoir aussi Python ajouté au PATH comme variable d'environnement sur Windows).
   - Installer le paquetage *virtualenv* avec pip

```
$ py -m pip install --user virtualenv
```

   - Créer et activer un environnement virtuel python

```
$ py -m venv my_dev_env
$ my_dev_env\Scripts\activate
```

   - Intaller les paquetages Python à partir du fichier de dépendances `requirements.txt` présent dans le répertoire

```
$ pip install -r requirements.txt
```

Ces étapes permettent de créer l'environnement de développement. Voici maintenant l'algorithme à appliquer pour modifier localement et déployer les modifications.

1. Ouvrir un terminal dans le répertoire de travail sur l'ordinateur de développement.
2. Ramener le code depuis Github

```
$ git pull
```

3. Facultativement, travailler sur une autre branche Git à partir d'ici
4. Initialiser/actualiser l'environnement Django et la base de données

```
$ py manage.py makemigrations
$ py manage.py migrate
```

5. Rouler le serveur de développement (dans un autre terminal ouvert dans le même dossier que `manage.py`)

```
$ py manage.py runserver 0.0.0.0:8000
```

6. Installer les paquetages du programme frontend (dans un autre terminal ouvert dans le dossier de l'application `frontend`)

```
$ npm install
```

7. Rouler le script de développement frontend au besoin

```
$ npm run dev
```

8. Apporter des modifications au code
9. Arrêter le serveur de développement et le script pour le frontend
10. Tester les points d'accès du backend (il se peut que `baseproject/__init__.py` ait besoin d'être supprimé momentanément)

```
$ py manage.py test
```

11. Facultativement, fusionner avec la branche principale
12. Compiler le code du frontend s'il à été modifié

```
$ npm run build
```

13. Pousser le code sur Github.

```
$ git commit -a -m"message commit"
$ git push
```

14. Se connecter au serveur Linux (avec SSH).
15. Naviguer jusqu'à l'environnement de production.
16. Ramener le code sur le serveur.

```
$ git pull
```

17. Activer l'environnement virtuel python et compiler les éléments statiques

```
$ source ../sensorproject/bin/activate
$ python3 manage.py collectstatic
```

Si les modifications ne s'appliquent pas automatiquement, il se peut que Gunicorn, NGINX ou le serveur lui-même aient besoin d'être redémarrés.

```
$ sudo systemctl restart gunicorn
$ sudo systemctl restart nginx
$ sudo shutdown -r now
```

Le cas échéant, on peut exécuter une ou plusieurs des commandes ci-haut. Ce processus est fluide, mais pas entièrement automatisé. Pour les travaux futurs, il serait intéressant d'utiliser un service qui favorise l'intégration continue (CI) et le déploiement continu (CD) comme Jenkins[37]. L'application pourrait aussi être conteneurisée avec un service comme Docker[38] pour la rendre plus évolutive.

3.11 Conclusion

Ce chapitre a présenté les développements du second projet du mémoire portant sur la collecte de données et le suivi. Le but ici était de détailler les réalisations tout en justifiant le choix des méthodes et des technologies utilisées, mais aussi de guider les travaux futurs.

L'Internet des objets amène une nouvelle évolution dans le domaine des interfaces machine-machine. Ce paradigme offre une nouvelle approche pour de multiples applications en exploitant la connectivité et l'interconnectivité entre les appareils. Ce projet cherche à en tirer parti pour l'appliquer en réadaptation, que ce soit en milieu clinique, en laboratoire ou au domicile.

Ainsi, un écosystème comprenant un serveur web, des applications web mobiles et embarquées ont été développées et déployées. L'architecture du serveur exposant une API REST permet une méthode de communication uniforme et réutilisable avec chaque client, quelle que soit sa nature. Cet écosystème permet à l'utilisateur de récolter des données à partir des capteurs de son choix à l'aide d'un simple microcontrôleur connecté à Internet et de les visualiser tout aussi facilement. Cette approche peu coûteuse favorise la collecte rapide et minimise l'encombrement physique puisqu'elle repose sur des connexions sans fil.

La phase préliminaire de ce projet a consisté au développement de chaque élément du système. Ce développement s'est fait en partant de la base et c'est ce qui fait qu'au final, l'application est sur mesure aux besoins et a le potentiel de l'être encore plus. Cependant, étant donné que chaque pièce est indépendante, rien n'empêche de les échanger pour de futurs travaux. Par exemple, une plateforme cloud IoT commerciale pourrait être utilisée pour servir l'application. Si tel est le cas, les adresses utilisées par les clients pour faire des requêtes seraient différentes, mais le reste pourrait rester essentiellement inchangé. On pourrait aussi penser à utiliser un autre microcontrôleur que l'ESP32 auquel cas une simple mise à jour du logiciel pour assurer sa compatibilité avec la librairie Arduino serait requise. Il est aussi possible qu'un utilisateur

37. Jenkins, https://jenkins.io/
38. Docker, https://www.docker.com/

fasse sa propre librairie dans le langage de programmation de son choix. Celui-ci n'aura qu'à consulter la liste des points d'accès de l'API REST pour faire les requêtes dont il a besoin.

Finalement, bien que ce projet ait été développé pour des fins de réadaptation, cet outil de collecte de données s'applique tout aussi bien à d'autres domaines scientifiques. Il pourrait par exemple être utilisé en mécanique du bâtiment pour éliminer le besoin d'avoir une carte d'acquisition dispendieuse et un ordinateur non loin pour récolter les données de capteurs de température ou d'humidité.

3.12 Bibliographie

[1] F. Civerchia, S. Bocchino, C. Salvadori, E. Rossi, L. Maggiani, and M. Petracca, "Industrial internet of things monitoring solution for advanced predictive maintenance applications," *Journal of Industrial Information Integration*, vol. 7, pp. 4–12, 2017.

[2] J. Cheng, W. Chen, F. Tao, and C.-L. Lin, "Industrial iot in 5g environment towards smart manufacturing," *Journal of Industrial Information Integration*, vol. 10, pp. 10–19, 2018.

[3] D. Raposo, A. Rodrigues, S. Sinche, J. Sá Silva, and F. Boavida, "Industrial iot monitoring : Technologies and architecture proposal," *Sensors*, vol. 18, no. 10, p. 3568, 2018.

[4] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things : Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[5] M. S. Hossain and G. Muhammad, "Cloud-assisted industrial internet of things (iiot)–enabled framework for health monitoring," *Computer Networks*, vol. 101, pp. 192–202, 2016.

[6] Y. J. Fan, Y. H. Yin, L. Da Xu, Y. Zeng, and F. Wu, "Iot-based smart rehabilitation system," *IEEE transactions on industrial informatics*, vol. 10, no. 2, pp. 1568–1577, 2014.

[7] Frédéric Schweitzer, "TeamAT Sensor Web App." https://github.com/team-ingreadaptulaval/TeamAT_sensors-web-app.

[8] Frédéric Schweitzer, "TeamAT Sensor BLE React Native." https://github.com/team-ingreadaptulaval/TeamAT_Sensors-BLE-react-native.

[9] G. Cloud, "Google Cloud for AWS Professionals." https://cloud.google.com/docs/compare/aws, 2020. Online ; accessed 2021-03-15.

[10] S. Madakam, V. Lake, V. Lake, V. Lake, *et al.*, "Internet of things (iot) : A literature review," *Journal of Computer and Communications*, vol. 3, no. 05, p. 164, 2015.

[11] R. A. Light, "Mosquitto : server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.

[12] H. G. C. Ferreira, E. D. Canedo, and R. T. De Sousa, "Iot architecture to enable intercommunication through rest api and upnp using ip, zigbee and arduino," in *2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob)*, pp. 53–60, IEEE, 2013.

[13] L. Tan and N. Wang, "Future internet : The internet of things," in *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, vol. 5, pp. V5–376, IEEE, 2010.

[14] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet : the internet of things architecture, possible applications and key challenges," in *2012 10th international conference on frontiers of information technology*, pp. 257–260, IEEE, 2012.

[15] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, "Research on the architecture of internet of things," in *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, vol. 5, pp. V5–484, IEEE, 2010.

[16] R. T. Fielding, *Architectural styles and the design of network-based software architectures*, vol. 7. University of California, Irvine Irvine, 2000.

[17] T. Connolly and C. Begg, "Database systems : A practical approach to design, implementation," *And Management. Boston : Pearson Education*, 2010.

[18] Stack Overflow, "Developer Survey 2020." https://insights.stackoverflow.com/survey/2020#technology-databases, 2020. Online ; accessed 2021-03-15.

[19] Y. Li and S. Manoharan, "A performance comparison of sql and nosql databases," in *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pp. 15–19, IEEE, 2013.

[20] A. Makris, K. Tserpes, G. Spiliopoulos, and D. Anagnostopoulos, "Performance evaluation of mongodb and postgresql for spatio-temporal data.," in *EDBT/ICDT Workshops*, 2019.

[21] K. Mew, *Learning Material Design*. Packt Publishing Ltd, 2015.

[22] Espressif Systems, "ESP32 WROOM-32 Datasheet." https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf, 2021. Online ; accessed 2021-03-15.

[23] Frédéric Schweitzer, "TeamAT Sensor Arduino." https://github.com/fschweitzer00195/TeamAT_Sensor_ArduinoLib.

[24] randomnerdtutorials, "Getting Started with ESP32 Bluetooth Low Energy (BLE) on Arduino IDE." https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/, 2019. Online; accessed 2021-03-15.

[25] mozilla, "Progressive web apps (PWAs)." https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps, 2021. Online; accessed 2021-03-15.

[26] B. Boucheron, "Initial Server Setup with Ubuntu 20.04." https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-20-04, 2020. Online; accessed 2021-03-15.

[27] E. Glass, "How To Set Up Django with Postgres, Nginx, and Gunicorn on Ubuntu 20.04." https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-ubuntu-20-04, 2020. Online; accessed 2021-03-15.

[28] B. Boucheron, "How To Secure Nginx with Let's Encrypt on Ubuntu 20.04." https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-20-04, 2020. Online; accessed 2021-03-15.

Conclusion

Ce mémoire a présenté les développements de deux projets en lien avec l'ingénierie de la réadaptation.

Le premier projet avait pour but de développer un algorithme améliorant le contrôle de technologies d'assistance de haute dimensionnalité avec des interfaces basées sur des capteurs de faible dimensionnalité offrant des commandes physiques très limitées. Pour ce faire, un algorithme de correspondance de séquences a été élaboré. L'idée générale de cet algorithme était de, tout d'abord, reconnaître des actions simples et faciles à exécuter pour l'utilisateur telles qu'un souffle dans une paille ou un tapement sur une table. Par la suite, l'algorithme faisait correspondre une combinaison de ces actions, une séquence, avec un mode de contrôle sur une technologie d'assistance quelconque. Ainsi, en segmentant le problème de reconnaissance de signaux, un système robuste a été obtenu. Pour tester le système, deux applications ont été développées et mises à l'essai. D'abord, l'algorithme a été adapté pour une interface de contrôle par le souffle et un bras robotisé d'assistance, puis pour une interface avec des mouvements de la main pour contrôler la souris et le clavier d'un ordinateur. De façon générale, l'algorithme s'est avéré être efficace lors des tests auxquels il a été soumis et apprécié des participants. L'avantage majeur d'un tel algorithme est sa versatilité. En effet, il n'est pas spécifique à aucun capteur ni à aucune technologie d'assistance. Il fait simplement le lien entre des actions simples, sous forme de courts signaux temporels, et les modes d'une technologie d'assistance.

Le second projet devait apporter une solution pour faciliter la collecte de données en réadaptation et était quant à lui beaucoup plus axé sur les outils technologiques que sur l'Humain. Cette solution a finalement été d'utiliser des microcontrôleurs munis de connexions WiFi et Bluetooth pour lire n'importe quels capteurs selon les besoins de chaque situation. Ces appareils sans fil ont été intégrés dans un écosystème développé selon un modèle de type IoT. Pour ce faire, un serveur hébergeant une base de données et exposant une API REST a été mis en place et des applications web et mobile ont été développées pour visualiser, analyser et repérer les données collectées par les microcontrôleurs. La première version de cette suite d'applications connectées a pour but d'instaurer un processus de travail plus fluide et confortable dans toutes tâches impliquant une collecte de données. Bien que celle-ci ait été développée pour le milieu de la réadaptation, elle peut facilement être appliquée à d'autres domaines de

l'ingénierie, en mécanique du bâtiment par exemple.

Travaux futurs

Bien entendu, tous grands projets scientifiques ne s'arrêtent jamais vraiment. Cela est d'autant plus vrai pour le monde en constante évolution dans lequel on vit. Voici donc les dernières grandes pensées de ce mémoire et lignes directrices pour la continuation des projets qui y sont présentés.

1. Interface de contrôle avec algorithme de correspondance de séquences
   - Un mécanisme de suggestion à la manière de l'auto complétion du clavier sur un téléphone intelligent pourrait être développé. Pour faire le parallèle avec le code Morse, un algorithme pourrait compléter une lettre pour l'utilisateur ou même pallier une frappe ratée au milieu de la séquence en déduisant celle-ci selon les habitudes de l'utilisateur et les séquences probables.
   - Une montre intelligente munie d'accéléromètres et de gyroscopes pourrait servir de capteur pour l'exécution des actions simples. À l'opposé, un microcontrôleur bon marché avec un IMU commercial ferait aussi bien le travail que les XSENS utilisés dans le chapitre 2.
   - Les systèmes développés doivent maintenant être testés sur des utilisateurs réguliers de technologie d'assistance. Chaque personne est différente, utilise différents appareils, différentes technologies d'aide, mais la versatilité de l'algorithme devrait faire en sorte qu'il puisse s'adapter et potentiellement améliorer le système déjà utilisé par celle-ci.
2. Application connectée de collecte de données
   - L'application est prometteuse, mais devra bien sûr être testée rigoureusement en milieux réels.
   - En accord avec la méthodologie itérative du mémoire, il serait intéressant de récolter des commentaires et des suggestions de plusieurs utilisateurs pour orienter la prochaine phase du projet.
   - Comme l'application est développée pour le domaine de la santé, il serait pertinent de l'analyser plus en profondeur, du moins pour les développements futurs afin de s'assurer de la sécurité de celle-ci pour conserver la confidentialité des données récoltées.
   - Dans quelques itérations, cette application pourrait être intégrée à un mécanisme de gestion de données plus complet pour diviser celles-ci par expérience ou par sujet. Ainsi, l'application serait d'autant plus sur mesure pour le domaine.

Lien avec les travaux du laboratoire

Les travaux présentés dans ce mémoire sont en lien avec différents travaux effectués au cours des dernières années en ingénierie de la réadaptation au Centre interdisciplinaire de recherche en réadaptation et intégration sociale (CIRRIS) et au laboratoire de robotique de l'Université Laval. Afin de mettre les travaux de ce mémoire dans ce contexte et de donner des références au lecteur par rapport à ces travaux afin de continuer ses lectures, voici une courte mise en contexte.

L'objectif des travaux du groupe d'ingénierie de la réadaptation à l'Université Laval est de développer des technologies d'assistances pour les personnes en situation de handicap et pour prévenir les blessures en milieu de travail. Les travaux portent sur le développement de mécanismes d'assistance [1, 2, 3, 4, 5, 6, 7], d'algorithmes intelligents pour robots d'assistance [8, 9, 10, 11, 12, 13, 14, 15] d'interfaces de contrôle pour robots d'assistances [16, 17, 18, 19, 20, 21, 22, 23, 24, 25],l'évaluation des technologies d'assistance [26, 27, 28, 29], ainsi que le développement de technologies pour le suivi du mouvement et de la fatigue musculaire en milieu réel [30, 31, 32, 33, 34, 35, 36].

3.13 Bibliographie

[1] J. Clouâtre, C. Doyon, J. Bouffard, and A. Campeau-Lecours, "Preliminary development of siara (six-dof assistive robotic arm)," 2021.

[2] P. Turgeon, M. Dubé, T. Laliberté, P. S. Archambault, V. H. Flamand, F. Routhier, and A. Campeau-Lecours, "Mechanical design of a new device to assist eating in people with movement disorders," *Assistive Technology*, pp. 1–8, 2020.

[3] M. Dubé, T. Laliberté, V. Flamand, F. Routhier, and A. Campeau-Lecours, "Mechanical design improvement of a passive device to assist eating in people living with movement disorders," 2020.

[4] G. Lemire, T. Laliberte, K. Turcot, V. Flamand, and A. Campeau-Lecours, "Preliminary design of a device to assist handwriting in children with movement disorders," *Rehabilitation Engineering and Assistive Technology Society of North America (RESNA)*, 2019.

[5] G. Lemire, T. Laliberté, and A. Campeau-Lecours, "Mechanical orthosis mechanism to facilitate the extension of the leg," 2020.

[6] P. Turgeon, T. Laliberte, F. Routhier, and A. Campeau-Lecours, "Preliminary design of an active stabilization assistive eating device for people living with movement disorders," in *Rehabilitation Robotics (ICORR), 2019 International Conference on*, IEEE, 2019.

[7] P. Turgeon, M. Dube, T. Laliberte, P. Archambault, V. Flamand, F. Routhier, and A. Campeau-Lecours, "Mechanical design of a new assistive eating device for people living with movement disorders," *Assistive Technology Journal*, 2020.

[8] F. Schweitzer and A. Campeau-Lecours, "Intuitive sequence matching algorithm applied to a sip-and-puff control interface for robotic assistive devices," 2020.

[9] D.-S. Vu, U. C. Allard, C. Gosselin, F. Routhier, B. Gosselin, and A. Campeau-Lecours, "Intuitive adaptive orientation control of assistive robots for people living with upper limb disabilities," in *Rehabilitation Robotics (ICORR), 2017 International Conference on*, pp. 795–800, IEEE, 2017.

[10] A. Campeau-Lecours, U. Côté-Allard, D.-S. Vu, F. Routhier, B. Gosselin, and C. Gosselin, "Intuitive adaptive orientation control for enhanced human-robot interaction," *IEEE Transactions on robotics*, 2019.

[11] A. Campeau-Lecours, H. Lamontagne, S. Latour, P. Fauteux, V. Maheu, F. Boucher, C. Deguire, and L.-J. C. L'Ecuyer, "Kinova modular robot arms for service robotics applications," *International Journal of Robotics Applications and Technologies (IJRAT)*, vol. 5, no. 2, pp. 49–71, 2017.

[12] A. Campeau-Lecours, V. Maheu, S. Lepage, H. Lamontagne, S. Latour, L. Paquet, and N. Hardie, "Jaco assistive robotic device : Empowering people with disabilities through innovative algorithms," *Rehabilitation Engineering and Assistive Technology Society of North America (RESNA)*, 2016.

[13] A. Campeau-Lecours and C. Gosselin, "An anticipative kinematic limitation avoidance algorithm for collaborative robots : Two-dimensional case.," in *IROS*, pp. 4232–4237, 2016.

[14] P. LeBel, C. Gosselin, and A. Campeau-Lecours, "An anticipative kinematic limitation avoidance algorithm for collaborative robots : Three-dimensional case," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pp. 3075–3080, IEEE, 2017.

[15] P. LeBel, C. Gosselin, and A. Campeau-Lecours, "A constraint based kinematic limitation avoidance : The sliding algorithm for six-dimensional collaborative robots," *IEEE Transactions on robotics*, 2019.

[16] S. Poirier, U. Cote-Allard, F. Routhier, and A. Campeau-Lecours, "Voice control interface prototype for assistive robots for people living with upper limb disabilities," in *Rehabilitation Robotics (ICORR), 2019 International Conference on*, IEEE, 2019.

[17] C. Fall, A. Campeau-Lecours, C. Gosselin, and B. Gosselin, "Evaluation of a wearable and wireless human-computer interface combining head motion and semg for people with upper-body disabilities," in *IEEE International NEWCAS Conference*, IEEE, 2018.

[18] C. Fall, U. Côté-Allard, Q. Mascret, A. Campeau-Lecours, C. Gosselin, and B. Gosselin, "Toward a flexible and modular body-machine interface for individuals living with severe disabilities : a feasibility study," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 2019.

[19] U. Cote-Allard, C.-L. Fall, A. Campeau-Lecours, C. Gosselin, F. Laviolette, and B. Gosselin, "Deep learning for electromyographic hand gesture signal classification using transfer learning," *IEEE Transactions on Biomedical Engineering*, 2019.

[20] F. Nougarou, A. Campeau-Lecours, D. Massicotte, M. Boukadoum, C. Gosselin, and B. Gosselin, "Pattern recognition based on hd-semg spatial features extraction for an efficient proportional control of a robotic arm," *Biomedical Signal Processing and Control*, vol. 53, p. 101550, 2019.

[21] C. L. Fall, G. Gagnon-Turcotte, J.-F. Dubé, J. S. Gagné, Y. Delisle, A. Campeau-Lecours, C. Gosselin, and B. Gosselin, "Wireless semg-based body–machine interface for assistive technology devices," *IEEE journal of biomedical and health informatics*, vol. 21, no. 4, pp. 967–977, 2017.

[22] C. L. Fall, P. Turgeon, A. Campeau-Lecours, V. Maheu, M. Boukadoum, S. Roy, D. Massicotte, C. Gosselin, and B. Gosselin, "Intuitive wireless control of a robotic arm for people living with an upper body disability," in *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pp. 4399–4402, IEEE, 2015.

[23] R. Crepin, C. L. Fall, Q. Mascret, C. Gosselin, A. Campeau-Lecours, and B. Gosselin, "Real-time hand motion recognition using semg patterns classification," in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 2655–2658, IEEE, 2018.

[24] U. Côté-Allard, C. L. Fall, A. Drouin, A. Campeau-Lecours, C. Gosselin, K. Glette, F. Laviolette, and B. Gosselin, "Deep learning for electromyographic hand gesture signal classification by leveraging transfer learning," *arXiv preprint arXiv :1801.07756*, 2018.

[25] C. L. Fall, F. Quevillon, M. Blouin, S. Latour, A. Campeau-Lecours, C. Gosselin, and B. Gosselin, "A multimodal adaptive wireless control interface for people with upper-body disabilities," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 3, pp. 564–575, 2018.

[26] A. Lebrasseur, J. Lettre, F. Routhier, P. Archambault, and A. Campeau-Lecours, "Assistive robotic arm : Evaluation of the performance of intelligent algorithms," *Assistive Technology Journal*, 2019.

[27] A. Lebrasseur, J. Lettre, F. Routhier, P. Archambault, and A. Campeau-Lecours, "name," *Assistive Technology Journal*, 2019.

[28] J. Schmidtler, K. Bengler, F. Dimeas, and A. Campeau-Lecours, "A questionnaire for the evaluation of physical assistive devices (quead) : Testing usability and acceptance in physical human-robot interaction," in *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*, pp. 876–881, IEEE, 2017.

[29] A. Lebrasseur, J. Lettre, F. Routhier, P. Archambault, and A. Campeau-Lecours, "Assistive robotic device : evaluation of intelligent algorithms," *Rehabilitation Engineering and Assistive Technology Society of North America (RESNA)*, 2018.

[30] M. Boyer, L. Bouyer, J.-S. Roy, and A. Campeau-Lecours, "A real-time algorithm to estimate shoulder muscle fatigue based on surface emg signal for static and dynamic upper limb tasks," 2021.

[31] A. Campeau-Lecours, D.-S. Vu, F. Schweitzer, and J.-S. Roy, "Alternative representation of the shoulder orientation based on the tilt-and-torsion angles," *Journal of Biomechanical Engineering*, vol. 142, no. 7, 2020.

[32] M. Boyer, A. Frasie, L. Bouyer, J. Roy, I. Poitras, and A. Campeau-Lecours, "Development and validation of a data fusion algorithm with low-cost inertial measurement units to analyze shoulder movements in manual workers," 2020.

[33] J. Clouâtre, A. Campeau-Lecours, and V. Flamand, "Preliminary development of a wearable device to help children with unilateral cerebral palsy increase their consciousness of their upper extremity," 2020.

[34] A. Fortin-Côté, J.-S. Roy, L. Bouyer, P. Jackson, and A. Campeau-Lecours, "Allumo : Preprocessing and calibration software for wearable accelerometers used in posture tracking," *Sensors*, vol. 20, no. 1, p. 229, 2020.

[35] I. Poitras, M. Bielmann, A. Campeau-Lecours, C. Mercier, L. J. Bouyer, and J.-S. Roy, "Validity of wearable sensors at the shoulder joint : Combining wireless electromyography sensors and inertial measurement units to perform physical workplace assessments," *Sensors*, vol. 19, no. 8, p. 1885, 2019.

[36] I. Poitras, F. Dupuis, M. Bielmann, A. Campeau-Lecours, C. Mercier, L. J. Bouyer, and J.-S. Roy, "Validity and reliability of wearable sensors for joint angle estimation : A systematic review," *Sensors*, vol. 19, no. 7, p. 1555, 2019.

Annexe A

Points d'accès de l'API REST du serveur de l'application IoT

La table A.1 présente les principaux points d'accès de l'API REST. La librairie de tests unitaires peut servir à titre d'exemple pour l'utilisation de ces points d'accès.

Table A.1 – Liste des points d'accès de l'API.

Méthode	Description	Contenu attendu de la réponse
api/auth/register		
POST	Inscription de l'utilisateur.	Information sur l'utilisateur, Jeton.
api/auth/login		
POST	Connexion de l'utilisateur.	Information sur l'utilisateur, Jeton.
api/auth/user		
GET	Récupération des informations de l'usager.	Information sur l'utilisateur.
api/auth/logout		
POST	Déconnexion de l'utilisateur, destruction du jeton.	N/A
api/devices/		
GET	Obtention des informations et données de tous les appareils.	Tableau avec *ID*, informations et données.
POST	Création d'un nouvel appareil.	*ID* et nouvelles informations.
api/devices/ID/		
GET	Obtention des informations et des données de l'appareil *ID*.	Informations et données pour *ID*.
PUT	Modifications de informations de l'appareil *ID*.	Nouvelles informations.

Méthode	Description	Contenu attendu de la réponse
DELETE	Supprimer l'appareil.	N/A
api/dataset ?from=$datetime_1$&to=$datetime_2$&maxsize=$maxsize$		
GET	Obtention des données de tous les appareils. Arguments optionnels pour la zone de temps et le nombre de données maximal.	Liste des données par *ID*.
api/dataset/ID/ ?from=$datetime_1$&to=$datetime_2$&maxsize=$maxsize$		
GET	Obtention des données de l'appareil *ID*. Arguments optionnels pour la zone de temps et le nombre de données maximal.	Liste des données pour *ID*.
DELETE	Suppression des données de l'appareil *ID*.	N/A
api/datalog/		
POST	Assignation d'une donnée a un appareil.	Donnée assignée, *ID*.
api/datalogmany/		
POST	Assignation de plusieurs données a un ou plusieurs appareils.	Données assignées, *ID*s.
api/devicecatalog		
GET	Obtention des informations des appareils.	Liste de informations des appareils par *ID*.
api/datafile		
POST	Exportation des données d'un appareil.	Fichier *.csv*
api/devicelast		
GET	Obtention de la dernière donnée de tous les appareils.	Liste de données par *ID*.
api/devicelast/ID		
GET	Obtention de la dernière donnée de l'appareil *ID*.	Liste de données pour *ID*.
api/datapicture		
GET	Obtention du nombre de données enregistrées par jour.	Liste du total de données regroupée par jour pour chaque appareil.

Méthode	Description	Contenu attendu de la réponse
api/datapicture/ID		
GET	Obtention du nombre de données enregistrées par jour pour l'appareil *ID*.	Liste du total de données pour *ID* regroupée par jour.

Annexe B

Fichiers de configuration du serveur web

Cette annexe présente trois fichiers de configuration NGINX et Gunicorn pour le serveur web. Ils servent à configurer le domaine et les aspects sécurité de la machine Linux. Les emplacements de ces fichiers et leur contenu sont énumérés ici.

`/etc/nginx/sites-available/sensorproject`

```
server {
  server_name www.teamatsensors.com teamatsensors.com;

  location = /favicon.ico { access_log off; log_not_found off; }

  location /static/ {
    root /home/fred/sensorproject/baseproject;
  }

  location / {
    include proxy_params;
    proxy_pass http://unix:/run/gunicorn.sock;
  }

  # managed by Certbot
  listen 443 ssl;
  ssl_certificate /etc/letsencrypt/live/teamatsensors.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/teamatsensors.com/privkey.pem;
  include /etc/letsencrypt/options-ssl-nginx.conf;
  ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
}
```

```
server {
  if ($host = www.teamatsensors.com) {
    return 301 https://\$host\$request_uri;
  } # managed by Certbot

  listen 80;
  server_name www.teamatsensors.com teamatsensors.com;
  return 404; # managed by Certbot
}
```

/etc/systemd/system/gunicorn.service

```
[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target

[Service]
User=fred
Group=www-data
WorkingDirectory=/home/fred/sensorproject/baseproject
ExecStart=/home/fred/sensorproject/sensorenv/bin/gunicorn \
          --access-logfile - \
          --workers 3 \
          --bind unix:/run/gunicorn.sock \
          baseproject.wsgi:application

[Install]
WantedBy=multi-user.target
```

/etc/systemd/system/gunicorn.socket

```
[Unit]
Description=gunicorn socket

[Socket]
ListenStream=/run/gunicorn.sock

[Install]
WantedBy=sockets.target
```