QuantumLeap, a Framework for Engineering Gestural User Interfaces based on the Leap Motion Controller

ARTHUR SLUŸTERS, Université catholique de Louvain, Belgium
MEHDI OUSMER, Université catholique de Louvain, Belgium
PAOLO ROSELLI, Università degli Studi di Roma "Tor Vergata", Italy
JEAN VANDERDONCKT, Université catholique de Louvain, Belgium

Despite the tremendous progress made for recognizing gestures acquired by various devices, such as the Leap Motion Controller, developing a gestural user interface based on such devices still induces a significant programming and software engineering effort before obtaining a running interactive application. To facilitate this development, we present QuantumLeap, a framework for engineering gestural user interfaces based on the Leap Motion Controller. Its pipeline software architecture can be parameterized to define a workflow among modules for acquiring gestures from the Leap Motion Controller, for segmenting them, recognizing them, and managing their mapping to functions of the application. To demonstrate its practical usage, we implement two gesture-based applications: an image viewer that allows healthcare workers to browse DICOM medical images of their patients without any hygiene issues commonly associated with touch user interfaces and a large-scale application for managing multimedia contents on wall screens. To evaluate the usability of QuantumLeap, seven participants took part in an experiment in which they used QuantumLeap to add a gestural interface to an existing application.

CCS Concepts: • **Human-centered computing** → **Gestural input**; **Graphical user interfaces**; *Interactive systems and tools*; • **Software and its engineering** → **Graphical user interface languages**; Runtime environments; • **Computing methodologies** → *Model development and analysis.*

Additional Key Words and Phrases: Engineering gestural user interfaces, Framework for interactive systems, Gesture-based interfaces, Gesture recognition, Leap Motion Controller, Mid-air gestural interaction, New datasets for engineering interactive systems, Processes for engineering interactive systems, Stroke gestures.

ACM Reference Format:
Arthur Sluÿters, Mehdi Ousmer, Paolo Roselli, and Jean Vanderdonckt. 2022. QuantumLeap, a Framework for Engineering Gestural User Interfaces based on the Leap Motion Controller. *Proc. ACM Hum.-Comput. Interact.* 6, EICS, Article 161 (June 2022), 47 pages. https://doi.org/10.1145/3532211

1 INTRODUCTION

Continuous improvement of motion-sensing technologies by sensor input and computer vision [23, 64] promotes a shift towards more natural and intuitive gestural User Interfaces (UIs) [48].

Authors' addresses: Arthur Sluÿters, Université catholique de Louvain, Louvain Research Institute in Management and Organizations, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium, arthur.sluyters@uclouvain.be; Mehdi Ousmer, Université catholique de Louvain, Louvain Research Institute in Management and Organizations, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium, mehdi.ousmer@uclouvain.be; Paolo Roselli, Università degli Studi di Roma "Tor Vergata", Vaile della Ricerca Scientifica, Roma, 00177, Italy, roselli@mat.uniroma2.it; Jean Vanderdonckt, Université catholique de Louvain, Louvain Research Institute in Management and Organizations, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium, jean.vanderdonckt@uclouvain.be.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.
2573-0142/2022/6-ART161 $15.00
https://doi.org/10.1145/3532211

After mastering 2D stroke gesture recognition [126] on touch-enabled, surface computing with multiple algorithms [103, 105, 114, 116], researchers and practitioners turn their attention to 3D motion gestures [5] performed in mid-air interaction spaces [7, 81].

Released in 2012, the Leap Motion Controller (LMC)[1] is an off-the-shelf device that precisely senses and tracks the positions of two hands and their fingers [109] to navigate through gestural user interfaces. Due to its accuracy and robustness [119], its affordability, and its wide availability, the LMC triggered a lot of projects in mid-air gesture recognition and interactive applications [57], such as for virtual reality [26, 80], mixed reality [106], music performance [32], games [71], human-robot interaction [42, 94], sign language recognition [37, 96, 99], and medicine [70, 129]. This LMC is shipped with an Application Programming Interface (API) for acquiring raw gesture data to be transformed into meaningful gestures by the developer. Hopefully, the LMC comes with a handful of *system-defined gestures*, such Key/screen tap, Swipe right/left/up/down, and Circle. Code[2] exists for recognizing these predefined gestures, for connecting them to a listener, and for incorporating their behavior into an interactive application in some particular language such as Unity.

However, when these system-defined gestures should be modified (*e.g.*, in their physical shape, in their templates) or when new gestures should be incorporated (*e.g.*, designer/user-defined gestures [44]), the corresponding code becomes no longer appropriate and should be updated significantly. This programming effort is substantive, tedious, and error-prone as gestures should be differentiated to be properly recognized. Although advanced programming techniques are applicable [100], this problem cannot be efficiently solved according to such a programmatic approach, but should invoke efficient gesture recognizers, which requires mastering object-oriented programming and machine learning [8].

Fortunately, many papers publish algorithms to effectively and efficiently recognize 3D gestures in general [123] and for the LMC in particular [38], primarily inspired by Artificial Intelligence (AI), Machine Learning (ML) [38], and computer vision [10, 91] techniques. While such techniques do exist, many of them do not make publicly available the programming code of their gesture recognizers and their associated gesture sets. When they release code and gesture sets, it is mainly for the challenge of recognizing certain gesture classes and not for developing gesture-based applications, thus making it challenging to reproduce given their intrinsic complexity and sophistication for mainstream development. Consequently, a recognizer working for one type of gesture may not be working for another type, unless preprocessing is repeated, including remodeling, re-parameterization, re-calibration, and retraining [3, 120]. As a result, developers of mainstream interactive applications are left with a significant amount of work for integrating everything (*e.g.*, dealing with multithreaded programming challenges to create responsive interfaces [100]).

In sum, there is no software engineering environment today that facilitates the development of gesture-based UIs by offering some device independence with respect to an LMC: almost everything should be programmed manually and little or no code can be reused from existing works since tied to a particular application domain. For instance, Fonk *et al.* developed a framework that acquires motions from an LMC, generates motion files, and exports them to software for kinematic analysis. Filho *et al.* [38, 101] initiated some work towards LMC independence with a controller invoking the LMC Unity API to transfer gestures in real-time to SciKit-Learn to rely on three ML algorithms compared: k-NN, SVM, and decision trees.

All these barriers should be leveraged to fully deploy gesture-based interactive applications based on the LMC. For this purpose, we present QuantumLeap, a framework for engineering

[1]See https://www.ultraleap.com/product/leap-motion-controller/

[2]See https://developer.leapmotion.com/. A binding exists for C#, older bindings are no longer supported for C++, C#, Java, JavaScript, Python, and Objective-C.

gestural UIs based on the LMC. This framework adopts a pipeline software architecture that can be parameterized to define a workflow of modules for acquiring, segmenting, recognizing, and sending gestures from the LMC to a gestural UI and its encompassing application.

This paper makes the following contributions to the field of Engineering Interactive Computing Systems (EICS). First, we report on our main observations and introduce the rationale behind the development of QuantumLeap after undertaking a Systematic Literature Review (SLR) of the LMC-based gesture recognition techniques and discussing the main challenges of gesture recognition in Section 2. Section 3 then describes QuantumLeap's software architecture and its modules based on frozen and hot spots. Ten gesture recognizers are integrated into the framework, some existing ones and some new recognizers with novel algorithms. Section 4 demonstrates the practical usage of the QuantumLeap framework by instantiating it into two different applications: a DICOM-compatible medical image viewer and a large-scale application for browsing multimedia contents, such as pictures, videos, maps, on wall screens. Section 5 evaluates QuantumLeap in an experiment with seven participants and proposes future changes that should further improve its usability. Section 6 discusses the limitations of the framework and its evaluation procedure. Finally, Section 7 concludes this paper and discusses future avenues of QuantumLeap.

Open Science. The QuantumLeap open source code is publicly available on GitHub at https://github.com/sluyters/QuantumLeap. Our website https://sites.uclouvain.be/ingenious/projects/quantumleap/ provides the reader with more resources, including the list of references from the SLR (Section 1), the configuration files for both applications (Section 4), the spreadsheets with data from the experiment (Section 5), the script for raw data acquisition of LMC-based gestures, as well as the pseudo-code of the implemented recognizers (see Appendix B), their documentation, and their explanation. Demonstration videos of the two applications are accessible on YouTube at https://youtu.be/tBhn7ae_Yio?list=PLj1KcEqIa0fHyDyeW_NpiLHCP6A8j7mS3.

2 RELATED WORK

This section motivates the existence of QuantumLeap in four parts: Section 2.1 describes the capabilities of the LMC, Section 2.2 then undertakes a Systematic Literature Review (SLR) [54] to identify and discuss the techniques for LMC-based gesture recognition, Section 2.3 discusses the numerous challenges of continuous gesture recognition, and Section 2.4 summarizes our findings.

2.1 The Leap Motion Controller

The LMC is an innovative, 3D motion capturing device designed specifically to track two hands and its fingers [109] with high precision and robustness [119]. Its API transforms the raw data from its two near-infrared cameras into a hand skeleton to be used in any application. Typical domains of application cover virtual reality, augmented reality, and mixed reality [46]. The LMC reaches sub-millimeter (0.2 mm) accuracy for static gestures and falls to 1.2 mm for dynamic gestures [119]. It natively recognizes a handful of system-defined gestures but custom gesture recognition algorithms are often preferred as they enable the support of a wider range of gestures [8].

2.2 Gesture Recognition Techniques for the LMC

We undertook a Systematic Literature Review (SLR) [54] to determine the algorithms used for LMC-based gesture recognition. We ran the search query "*[Full Text: "leap motion"] AND [Full Text: "gesture recognition"]*" on The ACM Guide to Computing Literature, the option covering both the ACM Digital Library of references and its associated venues[3]. The query resulted in 390 publications. After removing six duplicates and excluding 341 irrelevant references (266 based on their title,

[3]See https://dl.acm.org

Type	Algorithm	References
Opportunistic	Hard-Coded Thresholds	[12, 16, 39, 40, 94, 108, 110, 121, 122, 124, 127]
	Leap Gestures	[16, 21, 31, 71, 80, 94, 108, 110, 122, 128]
	GameWAVE Software	[71]
Nearest Neighbor	K-Nearest Neighbors	[17, 24, 26, 31, 38, 42, 69, 101, 105, 107]
Support Vector Machines	Support Vector Machine	[37, 38, 52, 53, 72, 79, 83, 99, 101]
Neural Networks	Multilayer Perceptron	[77, 109]
	Deep Feedforward Neural Network	[96]
	Feedforward Neural Network	[32]
	Gated Recurring Units	[107]
	Neural Network	[52]
	Radial Basis Function Network	[125]
Hidden Markov Models	Hidden Markov Model	[60, 61, 118, 129]
	Coupled Hidden Markov Model	[59]
Ensemble Learning	Random Forest	[79, 96]
	Bagging Trees	[52]
	Gradient Tree Boosting	[53]
Other AI/ML Techniques	Decision Tree	[38, 101, 127]
	Decision Table	[70]
	Linear Discriminant Analysis	[52]
	Fuzzy Integral	[70]
	Multinomial Logistic Regression	[53]
	Naive Bayes	[90]

Table 1. Summary of the gesture recognition algorithms identified in the SLR.

abstract, and introduction, and 75 based on their full-text), we identified 43 references with an LMC gesture recognizer (Table 1).

The vast majority of the analyzed publications ($\frac{16}{43}$=37%) implement gestural support in an opportunistic manner, *i.e.*, by relying on the few system-defined gestures natively supported by the LMC or by recognizing gestures based on hard-coded thresholds (*e.g.*, [12, 40, 127]). While this opportunistic approach may be appropriate in some cases (*e.g.*, for quick prototyping[4] or very simple UIs), it often forces developers to make significant compromises that could be avoided with more advanced recognition techniques. For instance, Liang *et al.* [71] designed a storytelling system in which children can interact with hand gestures, but these gestures are hard-coded and cannot be modified easily to better fit the motor abilities and preferences of each child. Zocco *et al.* [128] showed that an LMC-based system could be more efficient than the traditional trackpad-keyboard combination to interact with a Command and Control system. However, as they were limited to the few gestures natively supported by the LMC, they did not study whether more suitable types of gestures could result in even greater efficiency and better usability, such as those obtained with user-defined gestures [44].

Found in nine references ($\frac{9}{43}$=21%), K-Nearest Neighbors (KNN) classifiers [34] are also a popular choice for gesture recognition. These algorithms are a simple yet powerful alternative to the opportunistic approach as they are easy to implement and train while being reasonably fast and accurate. Some are presented as all-purpose recognizers [17, 105] while others target more specific applications such as handwritten numerals recognition [24] and Sign Language recognition, which remains a key topic for any language: American [37, 59, 77, 96], Greek [99], and Thai [107].

The remainder of the papers use more advanced ML techniques such as Neural Networks (NNs) from Deep Learning, Support Vector Machines (SVMs), Hidden Markov Models (HMMs), or Ensemble Learning. These techniques are used in all kinds of applications. For instance, Simos and Nikolaos [99] use an SVM to classify the 24 letters of the Greek Sign Language alphabet with high accuracy, and Kumar *et al.* [61] train an HMM to recognize words from single-stroke Latin sentences.

Finally, some papers combine algorithms and/or sensors. Daniels *et al.* [31] combine the LMC's native gestures with Wobbrock *et al.*'s $1 recognizer [120] to recognize a larger set of gestures for the manipulation of protein structures. Jiang *et al.* [52] combine an LMC with myography data, Marin *et al.* augment the LMC with an MS Kinect [78] or a depth sensor [79] to increase accuracy.

We observe that none of these papers propose an approach that is reusable across gesture sets, gestural UIs, and applications towards their mainstream development. Reusability, interoperability, and maintainability, three important (sub-)factors for software quality defined in the ISO/IEC 25010 [51], are not explicitly addressed.

2.3 Challenges of Continuous Gesture Recognition

This section highlights the challenges stemming from continuous gesture recognition, *i.e.*, the recognition of gestures in real-time.

2.3.1 Sensors limitations. Depending on their technology, sensors may be subject to a series of limitations. For example, vision-based sensors such as the LMC are sensitive to lighting conditions and may raise privacy concerns. On the other hand, radar-based systems preserve privacy and can function in darkness, but are sensitive to clutter, noise, and interference. Wearable sensors (*e.g.*, data gloves) may provide more accurate data, but this higher accuracy comes at the cost of practicality. Besides the limitations intrinsic to their technology, most sensors result in a compromise between a series of factors, including cost, field of view, framerate, and resolution.

Overall, practitioners should take all of these limitations into account in the design of their gesture-based user interfaces: gesture sets should be designed such that the sensor(s) can differentiate between each gesture and the selected sensor(s) should suit the context of use of the UI. If no single sensor suits the requirements of an application at a reasonable cost, a combination of multiple sensors (*e.g.*, the LMC and the MS Kinect [78]) could be considered.

2.3.2 Jitter. Jitter can be caused by tracking errors, sensor noise, or even a slight tremor of the user's hands. Too much jitter can significantly hinder interactions with gesture-based applications [87], as it may induce recognition errors, as well as a lack of precision when pointing and selecting, especially on larger screens. It is thus important to perform at least some filtering in a gesture recognition pipeline to increase the precision of pointing and provide cleaner data to the gesture recognizers.

Depending on the sensors and recognition algorithms used, some filtering may already occur at several points in the pipeline. For instance, the LMC applies some filtering on the data that it provides [27], and the sampling performed by some recognizers such as 3 cent [17], Jackknife [105], or $P+ [114] can act as a low pass filter. However, additional filtering may be applied to further smoothen the data.

Filters should be carefully selected and configured to provide reasonable smoothing while keeping latency low [87] and minimizing the risk of introducing inaccuracies. In [20], Casiez *et al.* propose the 1€ filter, a simple and fast filter based on double exponential smoothing that promises good smoothing and low latency for both fast and slow motion. They compare it to other filtering methods, namely, moving average, single and double exponential [63], and Kalman, and show that it manages *state-of-the-art* performance while being faster to execute.

2.3.3 Gesture production variability. Some variability in gesture production is usually observed both in gestures performed by a single user (intra-user variability) and by several users (inter-user variability). This variability can originate from differences in execution (*e.g.*, speed, trajectory) or in anatomy (*e.g.*, hand size, posture). If ignored, it can reduce gesture recognition accuracy and thus negatively impact the usability of gesture-based applications.

Several techniques exist to deal with gesture production variability. For instance, increasing the size of the training set by including more gesture templates produced by a greater range of users often results in higher recognition accuracy [113]. However, large training sets may increase execution time and are time-consuming to create. Synthetic data generation has been shown to increase accuracy while requiring little to no effort from developers [66, 102]. A series of tools, such as MAGIC [5, 55, 56] and Gesture à Go Go [66], have been developed to quickly augment gesture datasets with synthetic gestures. In [74], Maghoumi and LaViola used synthetic gestures generation in their DeepGRU recognizer to avoid overfitting.

To increase accuracy for single users, recognizers could be trained only on gestures performed by this specific user. Indeed, template-matching algorithms have been shown to perform better in user-dependent settings [113], *i.e.*, where the recognizers are trained and tested with templates produced by the same user. Coupled with synthetic data generation, one could achieve high accuracy with little input from the target user.

Finally, gesture recognition algorithms can be designed to accommodate some gesture production variability: for example, techniques such as Dynamic Time Warping (DTW) and spatial sampling can help deal with differences in gesture articulation speed [105, 113].

2.3.4 Static vs. dynamic gestures. Two main gesture types emerge from popular taxonomies: static and dynamic gestures [1, 25, 88, 117]. Inspired by [117], we propose the following definitions: a *static* gesture is a particular configuration of the body or of body parts at one instant of time, whereas a *dynamic* gesture corresponds to a particular motion of the body or of body parts through time. Different techniques may be leveraged to recognize each type of gesture. For instance, static gestures can be recognized at each frame to provide real-time recognition, while dynamic gestures are usually extracted from the raw stream of data (*i.e.*, gesture segmentation) before attempting to recognize them. This distinction between static and dynamic gestures is confirmed by our SLR on gesture recognition techniques for the LMC: of the 27 publications relying exclusively on non-opportunistic algorithms for gesture recognition, only four ($\frac{4}{27}$=14.8%) targeted both static and dynamic gestures, while the rest focused exclusively on one type of gesture.

2.3.5 Gesture segmentation. One of the greatest challenges in continuous gesture recognition is gesture segmentation, *i.e.* extracting user gestures from a continuous stream of data. To maximize user satisfaction, a good gesture recognition pipeline should recognize all intentional gestures with minimal input from the user, while ignoring parasitic gestures (*e.g.*, transitions between gestures or unintentional motions) [5].

There are many different approaches to segmentation. A sliding window segmenter feeds the input data into a fixed-length buffer. The content of the buffer is continuously sent to the recognizer for evaluation. While this technique does not require input from the user, it makes assumptions about the length of gestures and relies on the recognizer to distinguish gestures from noise. Sliding windows have been used in combination with the 3 cent and Jackknife recognizers [15, 105]. They have also been ML-augmented to distinguish intentional gestures from parasitic movements [58]. Segmentation may also be triggered by user input, *e.g.*, pressing a button performing a simple motion or hand pose to delimit gesture boundaries, or performing the gesture in a specific "active" zone in space. While accurate, these techniques require active input from the user and are thus likely to feel less natural to the user. Other publications have focused on using some properties of input data to trigger segmentation, such as speed and acceleration. For instance, in [22], Chen *et al.* used the energy and root mean square (RMS) of raw EMG signals to properly segment gestures (energy-based segmentation). Finally, techniques such as Machete [104] rely on simple recognizers that continuously evaluate input data to determine the possible boundaries of gestures.

As gesture segmentation techniques are still far from perfect, some measures should be taken to limit the number of false positives. Examples include training recognition algorithms to recognize parasitic gestures (*e.g.*, by generating synthetic "noise" gestures) [105] and rejecting recognized gestures if their score falls below a threshold. This threshold may be arbitrarily defined or automatically computed based on the training set [15].

2.3.6 Overlapping gestures. Two gestures are overlapping when part of one gesture is identical to part of the other gesture. While well-designed overlapping gestures may lead to increased memorability [92], they are challenging to recognize as they rely on near-perfect gesture segmentation. Overlapping may occur between two gestures of one gesture set or between a gesture from the gesture set and a parasitic gesture (*e.g.*, typing on a keyboard or reaching for an object). In the first case, one can design the gesture set in such a way that each gesture is easily distinguishable from all other gestures. In the second case, the designer should take into account as many potential parasitic gestures as possible to reduce the risk of false positives. Tools such as Ashbrook *et al.*'s MAGIC [5] assist designers in the creation of gesture sets with as little overlap as possible. GEsture Clustering toolKit (GECKo) [2] also helps investigate how end users articulate stroke gestures in terms of stroke number, order, and direction, thus assisting the designer in keeping or discarding a particular gesture to avoid overlapping.

2.4 Summary

The landscape of contact-less UIs is constantly evolving. Sensors such as the LMC are paving the way towards cheap and accurate gesture recognition, while researchers are continuously working on improving gesture recognition algorithms. However, the lack of a common framework for gesture recognition is making it increasingly harder to keep track of this evolution outside of the research community, resulting in only a handful of real-life applications. QuantumLeap aims at bridging the gap between researchers and developers by providing a common framework that helps researchers share the product of their research to the world and allows developers to write applications without spending time to solve the numerous challenges of gesture recognition.

3 QUANTUMLEAP, A FRAMEWORK FOR LMC-BASED APPLICATIONS

QuantumLeap is designed as an intermediate layer between gesture-based applications and the LMC. It contains all of the logic necessary for gesture recognition and only exposes a simple JavaScript API that allows developers to map gestures to actions without focusing on acquiring, segmenting, and recognizing gestures. Implemented in JavaScript (ECMAScript 2019) [49], the QuantumLeap framework is divided into two parts according to Pree's notation [89]: *frozen spots*, which are "fixed" parts which remain constant for all of its instantiations, and *hot spots*, which are generic parts to be modified depending on the context of use [14]. QuantumLeap is a *gray box* software framework [84], as it provides developers with two options: pick modules from a list of predefined components (black box) or write new components that meet their needs better (white box). The rest of this section describes the overall software architecture of QuantumLeap.

3.1 Pipeline Architecture

The pipeline architecture of QuantumLeap (Fig. 1) has been designed to be versatile and simple to configure (see Section 3.2). It consists of eight modules (*i.e.*, hot spots), each stemming from the challenges of gesture recognition presented in Section 2.3: a sensor module, filtering module, loaders for static and dynamic datasets, static and dynamic gesture recognizers, gesture segmentation module, and an analyzer module. Various implementations are provided for each module to suit the needs of a specific application (see Appendix A). When these implementations are not suited

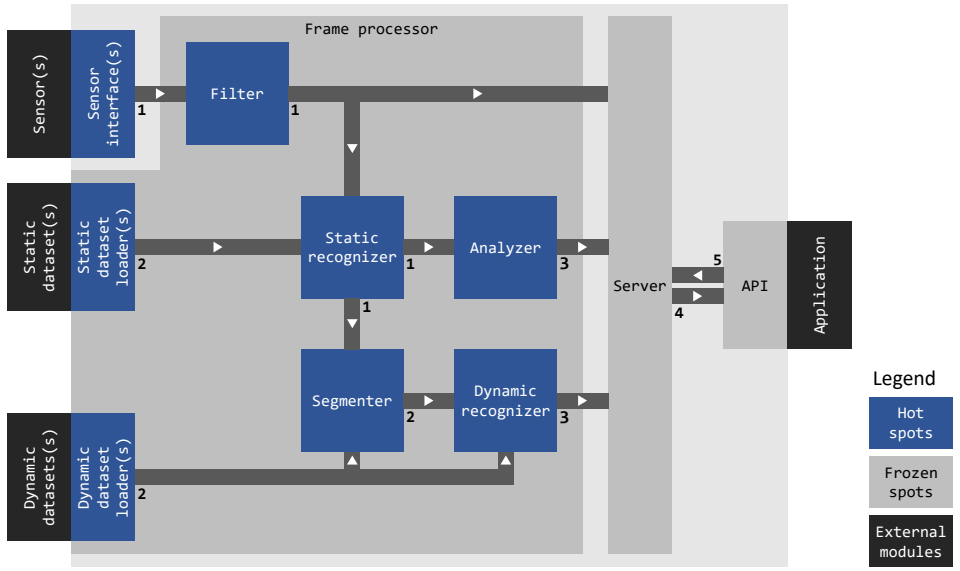


Fig. 1. QuantumLeap Framework software architecture.

for a particular application, the framework is extensible and developers are thus encouraged to propose new implementations. The modules communicate using the following data structure:

(1) A standardized frame.
(2) A Sample object.
(3) A JavaScript object with three properties: the gesture type (*i.e.*, static or dynamic), the gesture name, and the data from the analyzer if the gesture is static.
(4) A WebSocket message of type Data containing frames, static, and dynamic gestures.
(5) A WebSocket message of type Operation.

The rest of this section details the role of each module in the pipeline.

3.1.1 Sensor. This module serves as an interface between QuantumLeap and any physical sensor, such as an LMC, a radar sensor, or a Microsoft Kinect. It transforms raw, sensor-specific data into a standardized format usable throughout the framework. This module is called a set number of times per second by QuantumLeap to retrieve tracking data at one instant in time. Depending on the context of use, multiple sensor modules may be combined, *e.g.*, to increase recognition accuracy. A sensor module should implement three methods:

- getPoints(timestamp): get a list of points from the sensor at this instant in time.
- connect(): connect to the sensor.
- disconnect(): disconnect from the sensor.

One sensor module is provided with QuantumLeap. Its description can be found in Appendix A.1.

3.1.2 Filter. As discussed in Section 2.3.2, noise can negatively impact the user experience by reducing pointing and gesture recognition accuracy. Before it can be used by the other modules, each frame is first processed by the filter module, which computes a new frame with reduced noise. Filtering should be carefully tuned to fit the sensor and the target application, as too much noise reduction runs the risk of introducing some lag. A filter module should implement one method:

- filter(frame): compute a new frame with less noise.

Four different modules are provided. Their description can be found in Appendix A.2.

3.1.3 Static and Dynamic Datasets. These modules transform a gesture dataset into a GestureSet object, *i.e.* a standardized format compatible with the framework. Datasets are then used to train the static and dynamic recognizers. Fig. 2 depicts the UML V2.5 [11] Class diagram of the modular

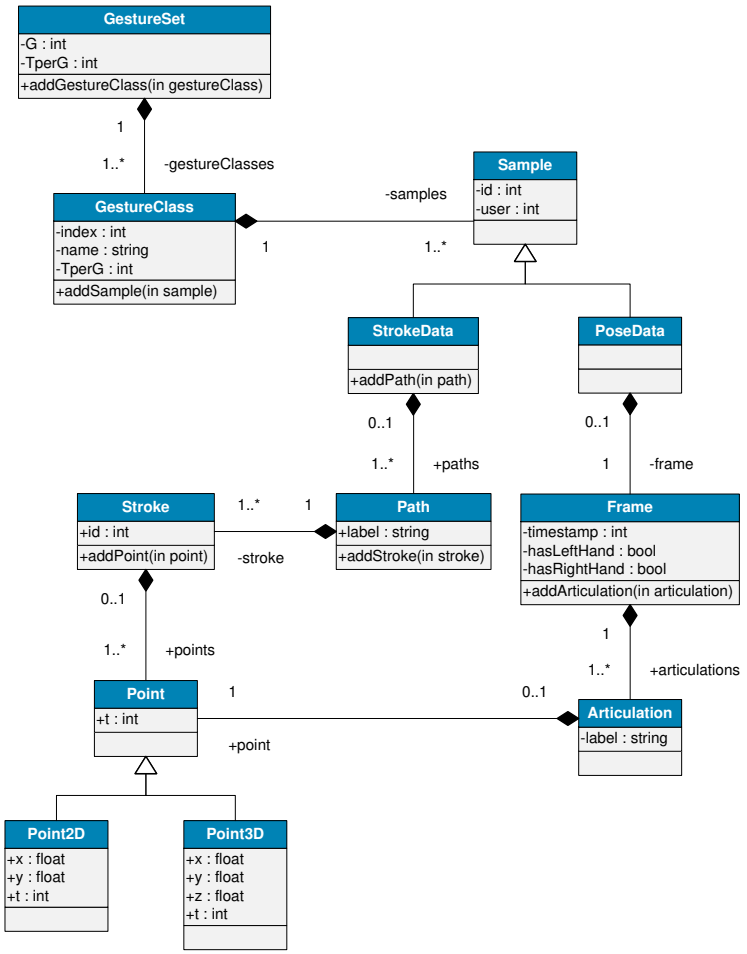


Fig. 2. UML V2.5 class diagram of the gesture sets.

data structure used in the framework. The GestureSet object can contain samples (also known as *templates* [112]) of two types: (1) PoseData, for static gestures consisting of a single frame that represents a hand pose; and (2) StrokeData, for dynamic gestures, which comprise multiple paths, where a path corresponds to the trajectory [18] of a specific articulation (*e.g.*, the tip of the right index). Paths are in turn decomposed into a series of strokes. Only one method is required:

- loadDataset(name, path, identifier): generate a GestureSet object from a gesture dataset.

Developers have two possibilities for selecting a gesture set. They can create a custom gesture set with any specialized software, such as Magic2 [55] or GestMan [76], or via an LMC gesture recorder[4]. Another option is to pick one of the many publicly available datasets, such as SHREC2019 [15], and convert it into the standardized format to be considered in QuantumLeap.

[4]We release the code of the LMC Gesture Recorder on GitHub at https://github.com/sluyters/LeapGesturePlayback.

3.1.4 Static Recognizer. The static recognizer module attempts to recognize static gestures at each frame. For each recognized gesture, the recognizer should compute a confidence score ranging from 0 to 1, where 1 indicates that it is absolutely confident that it has correctly identified the gesture. If the score is lower than a manually set threshold, the gesture is rejected. In the future, this threshold could be automatically computed by QuantumLeap for each gesture class based on the training dataset(s). Having a dedicated module for static gesture recognition allows using more efficient recognizers that are specialized for this task. This should thus result in increased accuracy and lower CPU usage. A static recognizer module should implement three methods:

- addGesture(name, frame): add a static gesture to the training set of this recognizer.
- removeGesture(name): remove a static gesture from the training set of this recognizer.
- recognize(frame): determine the name of the static gesture performed in the frame given as argument.

To avoid accidentally triggering dynamic gestures while the user is performing a static gesture, the recognition of dynamic gestures is stopped when a static gesture is recognized. Two static recognizer modules are provided. Their description can be found in Appendix A.4.

3.1.5 Analyzer. These modules extract additional data from the sensor frames, which can then be used inside applications. They can serve to offload heavy computation from the front-end to the QuantumLeap pipeline, to implement more advanced feature extraction techniques that would otherwise clutter the front-end application, or to remove sensor-dependent code from the application. Any analyzer module should implement two methods:

- analyze(frame): extract and return relevant information from a frame.
- reset(): reset the internal state of the analyzer.

As analyzer modules are highly specific to the application, we expect developers to create their own modules. However, one simple module is provided and described in Appendix A.5.

3.1.6 Segmenter. The role of this module is to identify intentional dynamic gestures from a continuous stream of data. It analyses frames one by one until it identifies a gesture. It then returns one or more lists of frames (*e.g.*, if it has multiple sliding windows), which are subsequently transformed into StrokeData objects that are sent to the dynamic gesture recognizer for recognition. If multiple lists of frames are returned at the same time, only the gesture with the highest confidence score is kept. The most appropriate method for gesture segmentation varies depending on the application, environment, and sensor(s). For instance, a simple sliding window segmenter could be appropriate for a multimedia application, while a more accurate technique may be required in environments where mistakes are costly, such as operating rooms. It is thus important to leave the choice of segmenter to the designer. A segmenter module should implement four methods:

- addGesture(name, sample): add a dynamic gesture to the training set of the segmenter.
- removeGesture(name): remove a dynamic gesture from the training set of the segmenter.
- computeSegments(frame): process a frame and return one or more lists of frames (*i.e.*, segments) if the frame marks the ending of a gesture.
- notifyRecognition(): notify the segmenter that a gesture has been recognized. The segmenter may use this information to take the coarticulation between two subsequent gestures into account (*e.g.*, by temporarily pausing gesture segmentation after a gesture is recognized).

Three different segmenters are provided. Their description can be found in Appendix A.6.

3.1.7 Dynamic Recognizer. Distinct from the static recognizers, dynamic recognizers analyze sequences of frames instead of individual frames. They are thus called only when the segmenter detects an intended gesture, which limits CPU usage and reduces the number of false positives. As

for the static recognizers, they should return a confidence score for each recognized gesture and implement three methods:

- addGesture(name, sample): add a gesture to the training set of this recognizer.
- removeGesture(name): remove a gesture from the training set of this recognizer.
- recognize(sample): determine the name of the gesture that corresponds to the candidate sample given as argument.

Ten dynamic recognizers are provided that cover multiple properties of gesture sets, such as uni- *vs.* multi-stroke and properties of invariance (see Appendix A.7).

3.2 Configuration

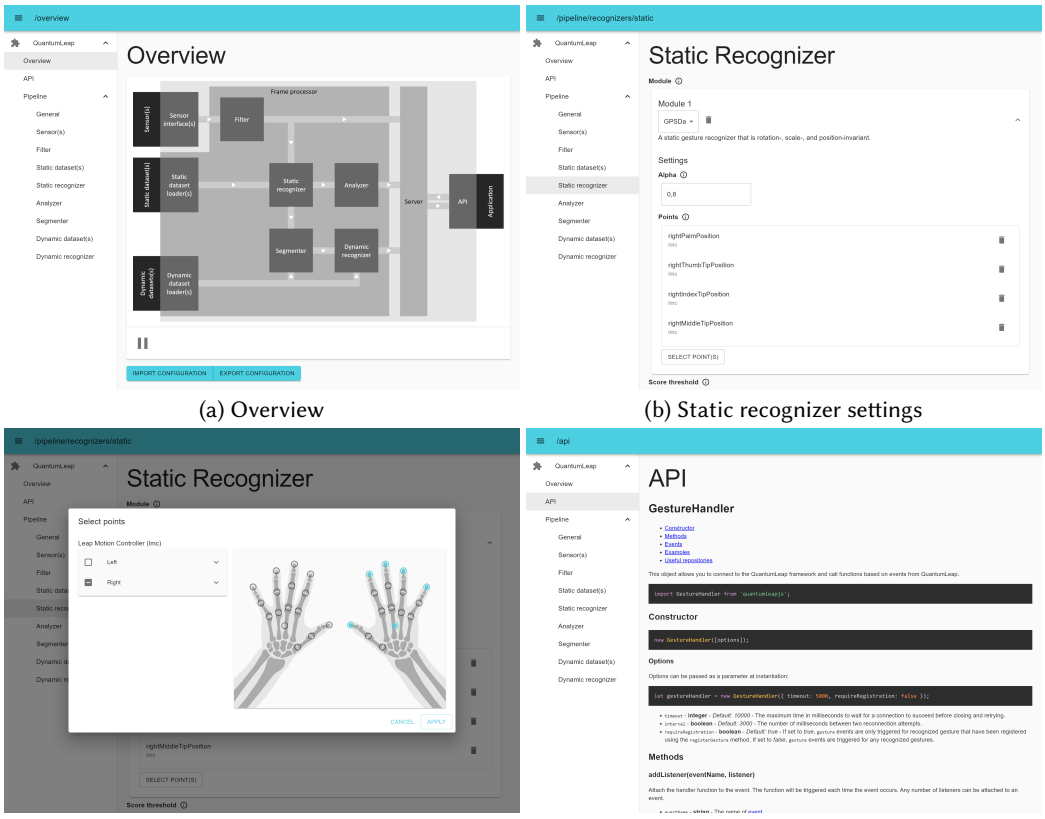
QuantumLeap comes with a configuration UI that allows developers to quickly set it up for their needs. It is divided into a series of pages: an "overview" page which shows the complete pipeline architecture and allows to navigate to the page of each module (Fig. 3a), an "API" page that contains the complete API documentation (Fig. 3d), and dedicated "module" pages for each type of module (Fig. 3b and 3c). The "module" pages let developers select a specific implementation of the module and configure it to their needs (*e.g.*, select the points used by a recognizer). The settings of each module can be defined in their corresponding "config-template.json" file. Framework configurations can be imported and exported as JSON files from the UI.

The configuration UI provides limited guidance in the form of short descriptions for each implemented module to help developers select the best set of modules for their application. Further guidance, such as recommending the most appropriate modules (*e.g.* gesture recognizers) for a particular application, is currently outside of the scope of this paper. However, resources like the comparative studies of Magrofuoco *et al.* [75] or Erazo and Pérez-Medina [35] may prove extremely valuable when choosing, *e.g.* a gesture recognizer.

3.3 Application Programming Interface

The QuantumLeap JavaScript API exposes the GestureHandler class, which is a member of the EventEmitter class of the Node.js core API. A GestureHandler object attempts to connect to a running instance of the QuantumLeap framework. It then emits events based on the data received from QuantumLeap. The reliance on a standard event-driven architecture should flatten the learning curve for front-end developers and facilitate the transition to gestural UIs, as events are already an integral part of modern UIs. The methods provided by the GestureHandler class include:

- connect([address]): connect to a running instance of QuantumLeap.
- disconnect(): disconnect from QuantumLeap.
- registerGestures(type, names): register gestures to QuantumLeap, to enable their recognition.
- unregisterGestures(type, names): un-register gestures from QuantumLeap.
- addListener(eventName, listener): attach a listener to a particular event. Possible events include:
  - frame: a frame has been received from QuantumLeap.
  - gesture: a gesture (static or dynamic) has been recognized.
  - connect: the application has connected to QuantumLeap.
  - disconnect: the application has disconnected from QuantumLeap.
- removeListener(eventName, listener): remove a listener attached to an event.



(a) Overview (b) Static recognizer settings

(c) Point selection (d) API documentation

Fig. 3. Screenshots of the QuantumLeap UI.

4 APPLICATIONS OF QUANTUMLEAP

This section illustrates how the QuantumLeap framework could be used to configure its modules for obtaining LMC-based gesture UIs for two different applications: the DICOM Viewer, a small application for manipulating medical images (Section 4.1), and the Large User Interface (LUI) project, a larger-scale application for browsing multimedia content such as photos and videos (Section 4.2).

4.1 DICOM Medical Image Viewer

As a baseline, we consider Pauchot *et al.*'s [86] application for browsing medical images stored in the DICOM format[5]. An LMC-based gestural UI is obtained on top of Carestream Vue PACS v11.3.4, an existing WIMP application for browsing medical images: LMC's hand tracking data emulate a mouse cursor that maps a gesture onto actions performed with a mouse in the application. Although an executable software is accessible along with a tutorial[6], the source code is not available and no details are provided on how gestures are designed, recognized, and mapped.

Instead of emulating mouse actions by replacing them with gestures, we implement a DICOM Medical Image Viewer based on QuantumLeap in order to produce a running application that is explicitly structured according to a configuration file. In its current state, the application shows a list of DICOM "studies" associated with a patient, each study corresponding to a particular examination

[5]A format widely used in medicine and life sciences, see https://www.dicomstandard.org/current
[6]See https://www.youtube.com/watch?v=yVPTgxg-SIk.

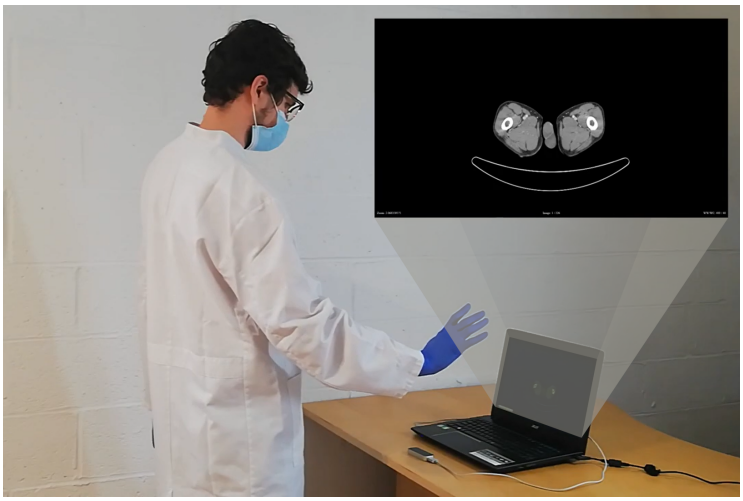


Fig. 4. The DICOM Medical Image Viewer application.

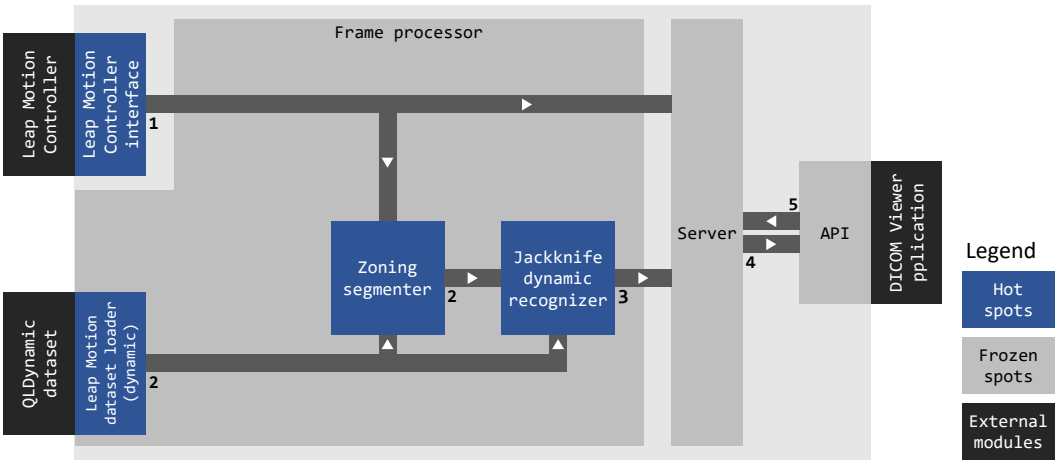


Fig. 5. Framework configuration for the DICOM application.

of the patient over time, with one or many images, such as X-rays, EMG. Healthcare workers, such as medical staff, surgeons, navigate from one study to the next using Swipe right/left/up/down gestures. Within a study, multiple actions are possible, such as manipulating the image, zooming in/out, rotating, and highlighting (Fig. 4).

Figure 5 shows the configuration of the QuantumLeap framework for our DICOM Medical Image Viewer, equipped with 14 actions and their corresponding gestures selected from a gesture elicitation study [120] conducted by Madapana *et al.* [73] in medical imaging. The Static recognizer and Analyzer modules are not needed as all the fourteen gestures are dynamic. It is configured with the Jackknife Dynamic recognizer since it benefits from a very high recognition rate, even on a large number of gesture classes [105]. Since this gesture set will not change anymore, it was decided to train the recognizer on this gesture set only.

This application has been developed in three days by an independent developer with 5 years of experience who is not a co-author of this paper in 375 Lines of Code (LOCs) for configuring QuantumLeap and defining the mapping between a gesture and an action. Without the framework,

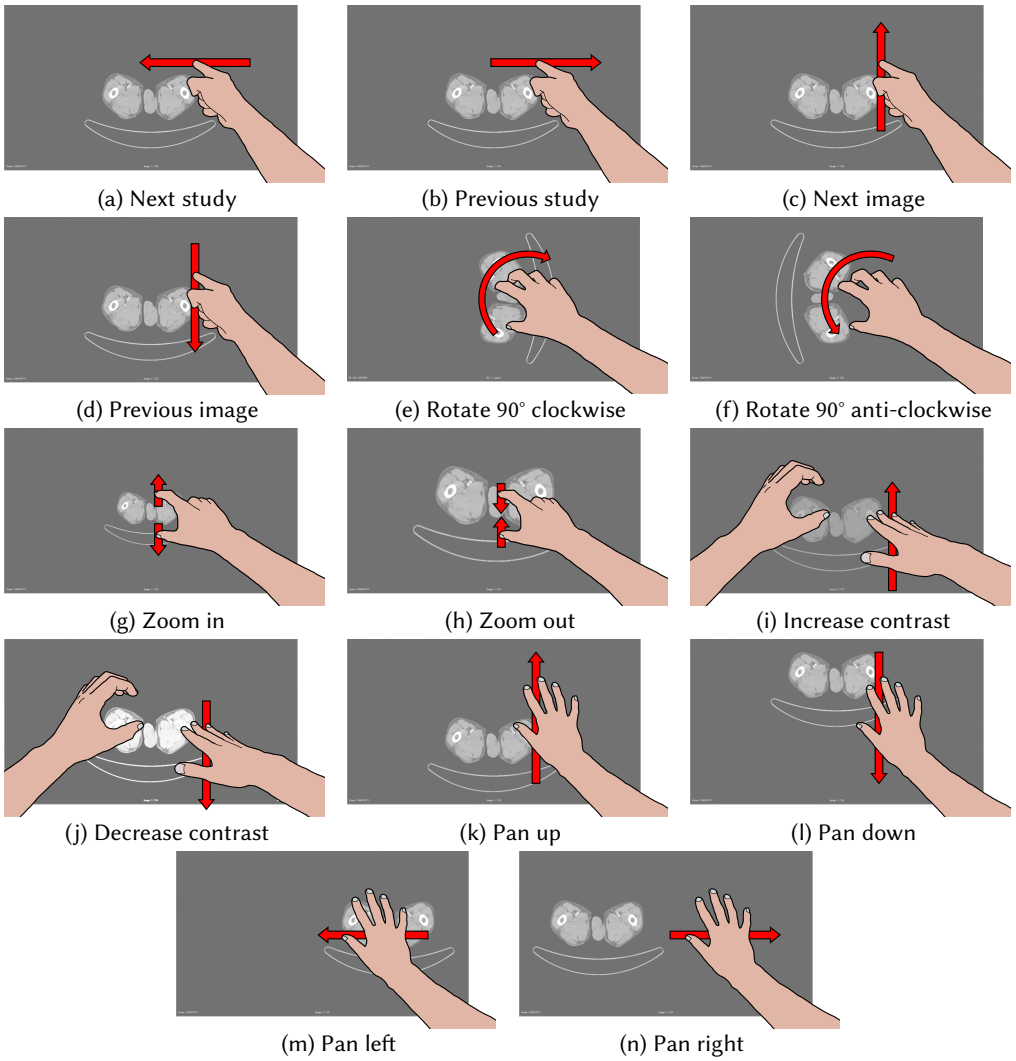


Fig. 6. Illustrations of each action and its assigned gesture in the DICOM Viewer application.

we estimate that the number of LOCs could increase fourfold, as the modules involved in this instantiation total 1012 LOCs distributed as follows:

- 101 LOCs to connect to the LMC and extract data in the correct format.
- 110 LOCs to load the dynamic gesture dataset.
- 76 LOCs for the zoning segmenter.
- 725 LOCs for the Jackknife dynamic gesture recognizer.

However, while counting LOCs can help quantify the saving enabled by QuantumLeap, there are different ways to implement gesture recognition into an application which may not take the same amount of LOCs. For instance, developers could recognize gestures based on hard-coded thresholds, develop their own gesture recognition algorithm from scratch, or rely on existing algorithms such as Jackknife (see Table 1). As such, LOCs saving using QuantumLeap could be as small as 200 LOCs when compared to a situation in which the developer relies on existing gesture recognition and segmentation algorithms. While more difficult to quantify, QuantumLeap also

allows substantial time-saving, as it cuts the time to search for, understand, and combine existing algorithms into an efficient pipeline for gesture recognition. In addition, the separation between the gesture recognition logic and the core application code enabled by QuantumLeap greatly simplifies the development process of the application.

4.2 Large User Interface

This section presents a second, large-size application of the QuantumLeap framework: integrating 3D gesture recognition into the semantic core of the Large User Interface (LUI)[7] project [85]. Our application is aimed at allowing people to naturally interact by gesture to manipulate multimedia content on large displays, such as a vertical wall display or a horizontal tabletop. As multimedia contents, such as pictures, videos, and maps, are probably the most common elements that are widely manipulated by the largest amount of users [33], an intuitive and accessible gesture-based UI should be adopted by a significant part of the population.

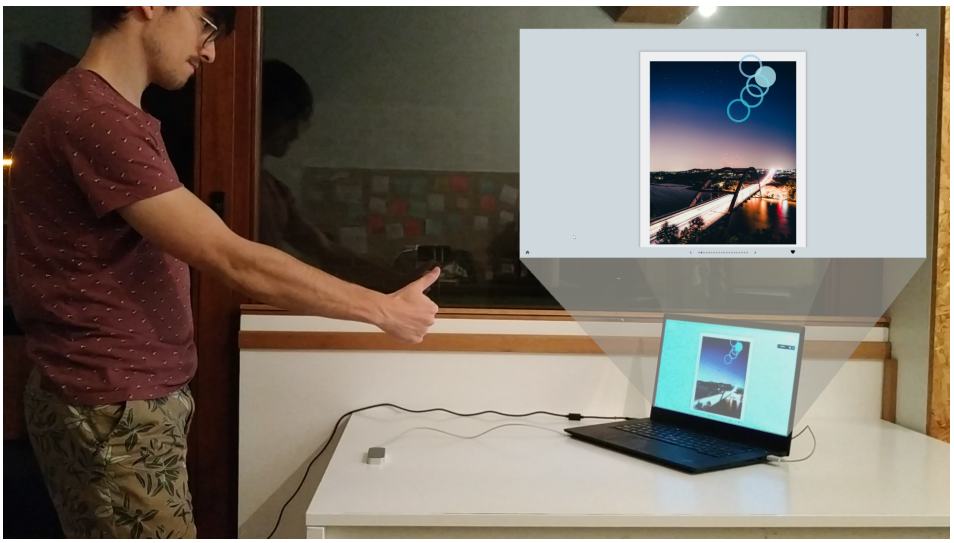


Fig. 7. The LUI application.

Built with the popular ReactJS framework, the version of LUI that we worked on was composed of four major parts: a welcome screen, an application menu, as well as two modules for browsing pictures and videos, respectively (Fig. 7). We contributed by implementing gestural support using the QuantumLeap framework by proceeding in three steps:

(1) Design of a gesture set (Fig. 9) by performing a gesture elicitation study [120] with 23 participants based on the protocol described by Gheran *et al.* [41].
(2) Selection of the most appropriate gesture recognizers by testing 2 static and 6 dynamic recognizers on the newly defined gesture set.
(3) Implementation of the gestures into the LUI project by using the API to call the appropriate Handler functions each time a gesture is performed.

Fig. 8 shows the resulting QuantumLeap configuration: combined with the Analyzer module, our new GPSDa static recognizer recognizes users' hand poses, which makes it possible to finely manipulate images and videos with direct and continuous visual feedback (Fig. 9e, 9f, 9g, 9h, 9i, 9j,

[7] See https://www.media.mit.edu/projects/large-user-interface-with-gesture-and-voice-feedback/overview/.

9l, 9m, 9n, 9o). Jackknife is combined with the window segmenter to recognize the set of discrete gestures (Fig. 9a, 9b, 9c, 9d, 9k).

For this application that is larger in size than the DICOM Viewer, we observe that the source code of LUI fits into 3830 LOCs, of which a conservative estimation totals 1280 LOCs distributed as follows:

- 101 LOCs to connect to the LMC and extract data in the correct format.
- 110 LOCs to load the dynamic gesture dataset.
- 63 LOCs to load the static gesture dataset.
- 143 LOCs for the GPSDa static recognizer.
- 60 LOCs for the analyzer.
- 78 LOCs for the window segmenter.
- 725 LOCs for the JackKnife dynamic gesture recognizer.

Compared to a worst-case scenario, in which developers would write their own gesture recognition and segmentation algorithms, using QuantumLeap would thus enable a $\frac{1280}{3830}$=33% reduction of LOCs. However, if we factor out the time required to conceive, to design, and to develop an adequate pipeline for gesture recognition from scratch, and to integrate it into a working environment, the potential benefits of QuantumLeap in terms of time-saving become more obvious than in terms of LOCs saving.

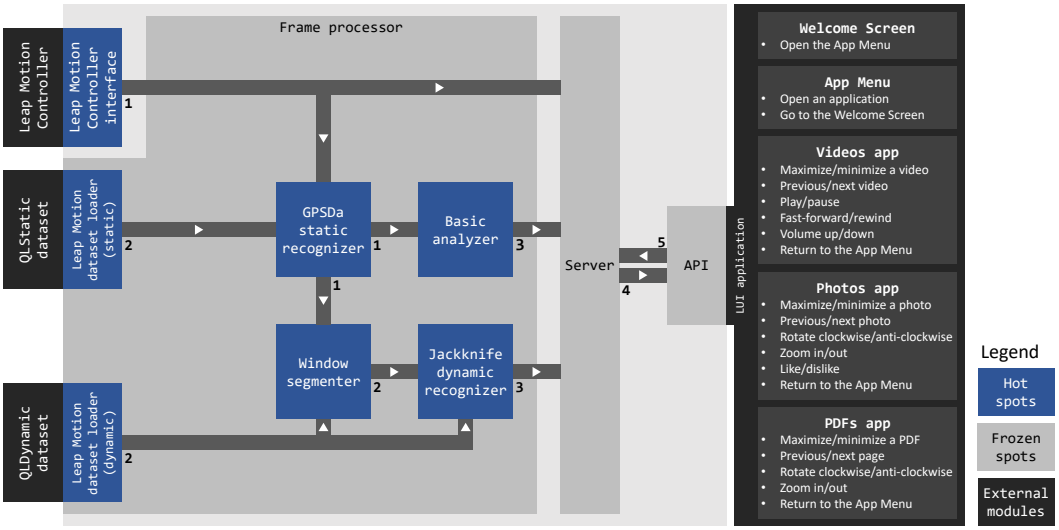


Fig. 8. Framework configuration for the LUI application.

5 EVALUATION

The ISO 9241-Part 11 [50] defines the "Usability" quality factor appearing in the ISO 25010 [51] as "*the effectiveness, the efficiency, and the satisfaction with which specified users achieve specified goals in particular environments*". Therefore, to evaluate the overall usability of QuantumLeap, an experiment was conducted to quantitatively evaluate the task complete rate (to cover effectiveness), the task completion time and the cognitive workload based on NASA-TLX [47] (to cover efficiency), and the System Usability Scale (SUS) [9] to assess its overall usability (to also cover satisfaction). This section describes the evaluation procedure and discusses its results.



(a) Previous (b) Next (c) Enable fullscreen

(d) Disable fullscreen (e) Zoom in (f) Zoom out

(g) Volume up (h) Volume down (i) Rotate clockwise

(j) Rotate anti-clockwise (k) Play/pause (l) Like

(m) Dislike (n) Fast-forward (o) Rewind

Fig. 9. Illustrations of each action and its assigned gesture in the LUI application.

5.1 Experiment

5.1.1 Environment. Participants were sitting in front of a computer in a quiet room. An LMC was already set up and placed in front of them. Each participant was provided with two elements:

- A small React web application consists of four buttons linked to four gestures (“left swipe”, “right swipe”, “thumb”, and “point index”). Clicking on a button displays the type, name, and image representation of the gesture on the screen.
- The QuantumLeap framework, including the documentation of its API. The configuration of the framework was reset before each experiment.

5.1.2 Consent. Before the experiment, participants were introduced to the procedure and informed that they could leave the experiment at any time. They were then invited to sign a GDPR-compliant consent form and fill in a questionnaire with their age, highest education level, current occupation,

estimated number of years of experience in the field of computer science, and estimated expertise in development based on the nine expertise levels defined by Costabile *et al.* [29] (Fig. 10).

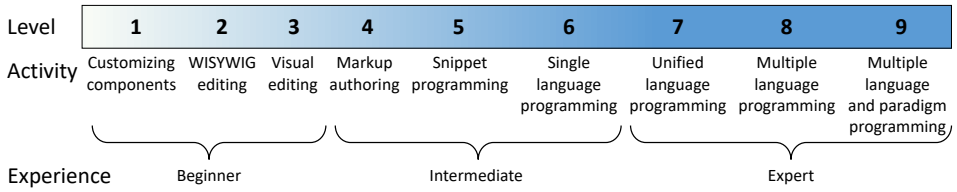


Fig. 10. The nine development expertise levels [29].

5.1.3 Participants. A purposive and snowball sampling was used to recruit 7 participants (all males), aged between 21 and 27 years old (M=23.3, SD=1.9), who volunteered for the experiment. They reported between 1 and 7 years of experience in the field of computer science (M=4.9, SD=2.2), and expertise levels ranging from 5 to 9 (M=7.6, SD=1.8). Education levels included bachelor's degree (4 participants) and master's degree (3 participants). Our criteria for inclusion in this experiment required participants to have at least one year of professional development experience.

5.1.4 Procedure and Tasks. Participants were first introduced to the application and the QuantumLeap framework. They were given a few minutes to try them out and look at the source code of the application before continuing the experiment. The rest was divided into five tasks:

(1) *Configuration*: the participants configure the QuantumLeap framework for the application with its configuration UI. This task comprises 7 steps, detailed in Appendix C.1.
(2) *Development*: the participants modify the source code of the application to make it react to static and dynamic hand gestures. This task comprises 13 steps, detailed in Appendix C.2.
(3) *SUS Questionnaire*: once the participants had finished the above two tasks and their related steps, they were asked to fill out the System Usability Scale (SUS) questionnaire [9], which comprises ten questions evaluating the usability of a system (in this case, QuantumLeap). This questionnaire has been selected for its proven reliability [6] and its efficient administration.
(4) *NASA-TLX Questionnaire*: participants were then asked to fill out the NASA-Task Load Index (NASA-TLX) questionnaire [47], which assesses the estimated workload of the *configuration* and *development* tasks. It consists of six subscales: mental demand, physical demand, temporal demand, effort, performance, and frustration level. Each subscale is then assigned a weight that is determined by asking the participant to select the subscale that contributes the most to the workload in 15 pairwise comparisons. In this way, each participant was able to express which subscale is perceived as the most important with respect to others, as opposed to maintaining the same weight for all subscales. Pandian and Tuleri's NASA-TLX software [82] was used to simplify the process of data collection and analysis (Fig. 11).
(5) *Open-Ended Questions and Comments*: the participants answer the following four open-ended questions to provide insights on their overall feeling:
   - If you did not complete all the tasks, what was the reason?
   - What did you think of QuantumLeap?
   - What did you like?
   - What would you change?

Each evaluation took approximately 80 minutes per participant.

5.1.5 Quantitative Measurements. Participants were timed during the experiment. The "task completion time" is measured as the time taken by the participant for each of the *Configuration* and *Development* tasks. The "step completion time" is measured as the time taken to perform each step

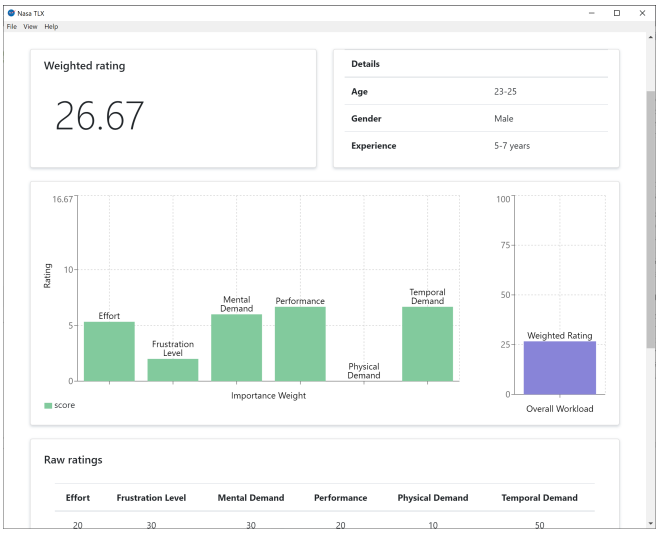


Fig. 11. NASA-TLX software [82] screenshot showing results of a participant: the overall weighted rating, the rating per subscale, and the individual weights for each subscale.

of each of the two aforementioned tasks. We also measure the "task completion rate" as the ratio between the number of steps completed successfully by a participant and the total number of steps required to complete the procedure.

5.2 Results and Discussion

5.2.1 Task Completion Rate. Despite their differences in terms of years of experience and level of expertise, six of the seven participants ($\frac{6}{7}$=86%) completed all the steps of each task without any issue on the first run. Only one participant initially performed the fifth step of the *Development* task incorrectly. But this participant immediately realized his mistake and subsequently corrected it after completing the next step, ultimately resulting in a task completion rate of 100%. This suggests that the effectiveness is well covered by QuantumLeap.

5.2.2 Task and Step Completion Times. As a potential optimal performance, the first author of this paper (23 years, 6 years of experience, level 8) completed the *Configuration* and *Development* tasks of the experiment in 2.1 minutes and 10.2 minutes, respectively. On average, participants took 6.6 minutes to complete the first task (SD=1.2), with the slowest participant completing it in 9.1 minutes and the fastest in 5.2 minutes. For the second task, the completion time was between 22.2 and 65.1 minutes (M=37.4, SD=18.2). Fig. 13 shows that participants spent a more significant amount of time on the fifth step of this task (*i.e.*, adding an event listener that reacts to the "swipe left" gesture), which is normal since this is the most complex step.

While steps five and eight are similar, participants took seven times longer to perform the former than the latter. A plausible explanation could be that the first time they encountered the problem of reacting to a gesture, participants took some time to check the API before modifying the source code of the application. When they mastered the API, participants could then quickly complete the remaining steps.

To investigate the relationship between participants' expertise and their task completion time, we compute a Spearman's rank correlation between participants' reported level of expertise in development and their completion time for the *Configuration* and *Development* tasks. While no significant correlation was observed between the completion time of the first task and the expertise

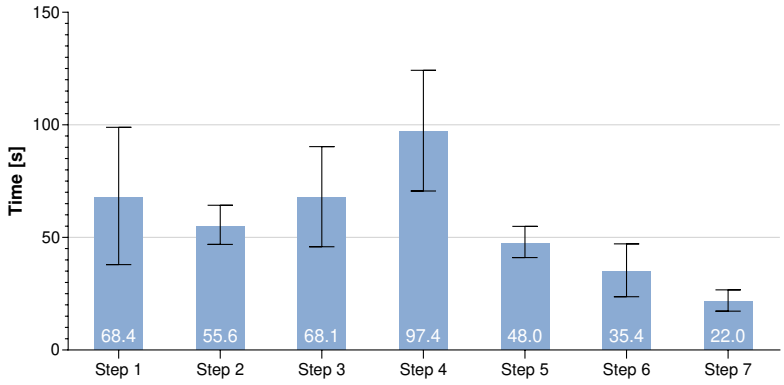


Fig. 12. Average time for each step of the *Configuration* task. Error bars show a confidence interval with α=0.5.

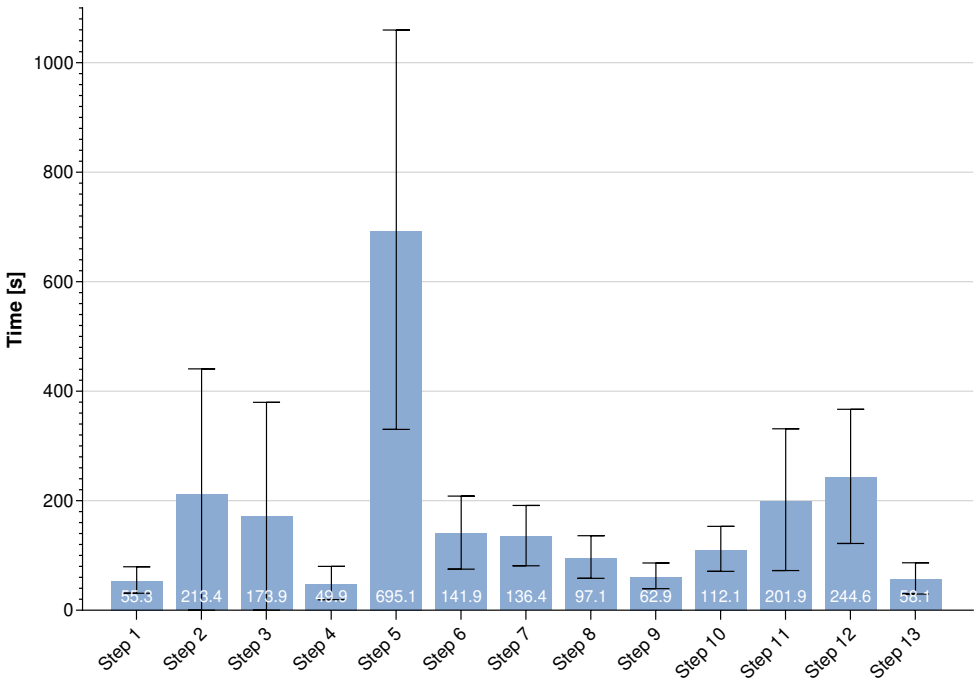


Fig. 13. Average time for each step of the *Development* task. Error bars show a confidence interval with α=0.5. Note that one participant performed tasks 8, 9, and 10 in parallel. The participant's time for each of these tasks was thus inferred from the total time to perform the three tasks and the ratio between the average time for these tasks.

level, thus suggesting that the configuration UI is perceived with a similar easiness regardless of the skill level, the completion time of the second task significantly correlates with expertise (r= − .93, p=.002**). This result was expected, as the second task of the experiment requires participants to leverage their programming skills to modify the source code of the application.

5.2.3 SUS Questionnaire. Since the answers to the SUS questionnaire were not normally distributed based on a Shapiro-Wilk test (*e.g.*, for question 3, W=0.45, p<.001***), we investigated the overall appreciation of the participants by conducting a one-sample Wilcoxon test comparing their answers to the midpoint of the scale (*i.e.*, 3.0). For odd-numbered, resp. for even-numbered questions, an answer is considered positive if it is significantly higher, resp. lower than the midpoint value. All odd-numbered, resp. even-numbered, questions received a mean higher, resp. lower than the midpoint of the scale, thus representing an already positive assessment. Furthermore, six of the ten questions are statistically significantly different from the midpoint value, suggesting an even more positive answer for these questions: question 1 (W=21.0, p=.031*), question 3 (W=28.0, p=.016*), question 5 (W=21.0, p=.031*), question 6 (W=−21.0, p=.031*), question 7 (W=28.0, p=.016*), and question 10 (W=−21.0, p=.031*). As recommended by Brooke [9] and Bangor *et al.* [6], we also analyzed the SUS questionnaire as a whole. SUS scores range between 62.5 and 85.0 out of 100 (M=77.9, SD=7.8), resulting in a rating of B+ ("good") on Sauro's rating scale [95], which denotes a reasonable level of usability for such an environment. According to Lewis and Sauro [68], the overall SUS score can be further divided into two subcomponents, namely learnability and usability. Learnability is computed by summing the scores of questions 4 and 10 of the questionnaire and multiplying the result by 12.5. Usability is computed by summing the scores of the other questions and multiplying the result by 3.125. QuantumLeap obtains a mean learnability score of 78.6 and a usability score of 77.7.

To investigate the internal consistency reliability among participants, we also computed Guttman's λ_4=.91 coefficient [45] with n=2,000 iterations. This value is above $\lambda_4 \geq .90$ which represents the highest rank of consistency, usually required for high-stakes decisions like school admissions [13], thus suggesting that participants consistently answered the SUS questionnaire and that their answers were reliable enough. Guttman's λ_4 coefficient represents a good measure of reliability that produces a value higher than the usual Cronbach's α, but with a positive bias likely to be small if the estimated reliability is larger than 0.85 (which is the case here) when the sample is small (n=7). After computing the Spearman's rank correlation between participants' SUS scores and their reported level of expertise, we observe a positive correlation (r=.82, p=.033*), which suggests that the system feels more usable to experienced users than to intermediary ones.

5.2.4 NASA-TLX Questionnaire. Workload scores range from 8.7 to 65.3 out of 100 (M=34.0, SD=19.5). According to Grier [43], who analyzed the NASA-TLX scores of 37 computer activities, the measured workload for QuantumLeap falls into the second quartile (score lower than 54.0). Each participant managed to complete all steps of the experiment and some even reported being impressed by the results, considering that they would not have been able to do the same without the help of the framework. This could explain the low impact of the "performance subfactor". Contrarily to the SUS score, no significant correlation was observed between participants' workload scores and their reported level of expertise after computing the Spearman's rank correlation.

Subscale	**Score**	
	M	SD
Mental Demand	10.0	6.4
Physical Demand	0.4	1.0
Temporal Demand	3.9	2.4
Effort	11.0	8.7
Performance	4.5	2.9
Frustration level	4.3	8.5
Overall	34.0	19.5

Table 2. Average weighted score and standard deviation for each subscale of the NASA-TLX questionnaire.

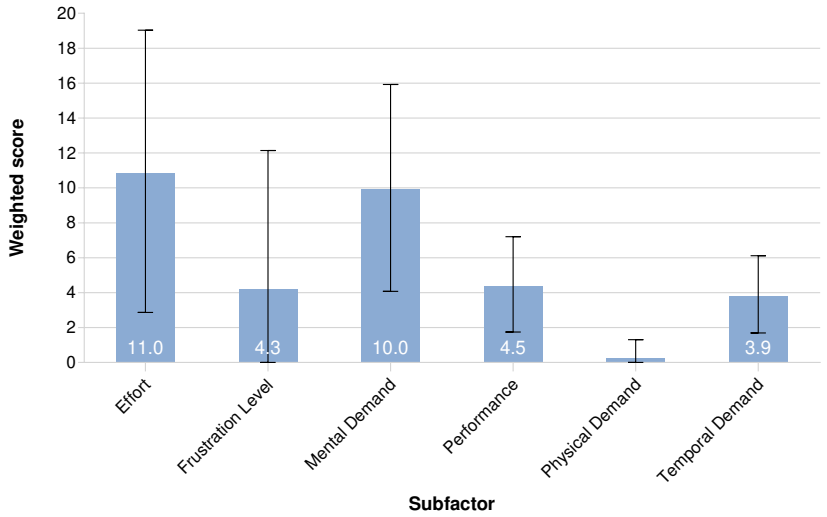


Fig. 14. Average weighted score for each subscale of the NASA-TLX questionnaire. Error bars show a confidence interval with α=0.5.

5.2.5 Open-Ended Questions. Overall, participants found that adding gesture support to an application was surprisingly quick and easy. However, some participants expressed some concern about not being able to configure the pipeline correctly without external help. Their remarks and suggestions are summarized as follows:

(1) *Configuration.* Comments targeting the configuration process, the UI, and the usability.
- There should be some feedback to indicate to the user that the framework is starting/stopping ($\frac{6}{7}$=86%).
- There should be some feedback to notify the user that the settings were successfully saved ($\frac{5}{7}$=71%).
- After selecting a module, its settings should be displayed by default, instead of having to click to display them ($\frac{5}{7}$=71%).
- A lot of steps are required to configure the pipeline ($\frac{2}{7}$=29%).
- There should be a button to add a module, instead of just selecting the module in a dropdown menu ($\frac{2}{7}$=29%).
- There should be a tutorial on how to configure the pipeline ($\frac{1}{7}$=14%).
- There are some inconsistencies in the UI ($\frac{1}{7}$=14%).
- There should be a simple way to switch from right-handed gestures to left-handed ($\frac{1}{7}$=14%).

(2) *Development.* Comments targeting the development of a gesture-based application using the QuantumLeap API.
- There should be a method to directly attach a listener to a specific gesture, instead of having to listen to the generic "gesture" event and then check the gesture name ($\frac{2}{7}$=29%).
- The API documentation should provide the data types of event properties ($\frac{1}{7}$=14%).
- The API documentation should clearly show the link between listeners and events ($\frac{1}{7}$=14%).
- The distinction between static and dynamic gestures may be confusing ($\frac{1}{7}$=14%).

(3) *Experiment* Comments targeting the experiment itself (*e.g.*, unclear steps).
- The order of some steps was confusing: the call to registerGesture should have been introduced before the call to addListener ($\frac{2}{7}$=29%).

- In step three of the second task, the address and port of the running instance of QuantumLeap should have been provided ($\frac{2}{7}$=29%).

5.3 Changes

Following this experiment, we are planning on improving QuantumLeap based on the aforementioned participants' comments.

5.3.1 Short-term Changes. The most requested changes have already been implemented in the latest version of QuantumLeap. They include:

- Displaying a loading icon in the UI when QuantumLeap is starting/stopping.
- Displaying a message in the UI when the settings have been saved.
- Displaying the settings of modules by default in the UI, instead of requiring the user to click on a button to display them.

5.3.2 Long-term Changes. We believe that these long-term changes would prove especially useful for less experienced developers:

- Add the possibility to attach listeners to specific gestures in one step, instead of requiring developers to listen to the generic "gesture" event.
- Propose a set of default configurations to allow developers to quickly prototype gesture-enabled applications without having to build a complete pipeline configuration.
- Create an interactive tutorial that guides users through the configuration of the QuantumLeap pipeline.
- Automatically configure the pipeline based on the selected gesture set and target usage. For instance, QuantumLeap could suggest a segmenter that is well-suited to the environment and determine the most accurate recognizers for the gesture set.

6 LIMITATIONS

6.1 QuantumLeap Framework

As mentioned in Section 3.2, this paper does not provide guidance for configuring the gesture recognition pipeline. As such, while QuantumLeap reduces the effort required for developing a gesture-based application, some understanding of the modules (*e.g.*, gesture recognizers) is still required to build an efficient pipeline.

The pipeline architecture of QuantumLeap combines modularity with some rigidity. For instance, adding new module types (*e.g.*, a third type of gesture recognizer) or changing their order in the pipeline cannot be done easily without modifying the source code of the framework. This rigidity makes QuantumLeap more accessible to less experienced developers but may prevent some developers from experimenting with more advanced gesture recognition techniques.

Finally, this work focuses exclusively on the LMC sensor, which can be restrictive. Extending the framework to other devices is possible and would allow developers to make their applications accessible to more users by supporting various sensors, as well as to experiment with combinations of devices (*e.g.* LMC and Kinect).

6.2 Evaluation

In this section, we discuss the limitations of our evaluation based on three of the four types of evaluation strategies defined by Ledo *et al.* [65]: (1) demonstration, (2) usage, and (3) technical evaluation. The fourth one, *i.e.* heuristic evaluation, evaluates to what extend a toolkit meets standard guidelines, which is not relevant in our case.

6.2.1 Demonstration. This evaluation strategy is useful to show what can be built using a toolkit or, in this case, a framework. We developed two applications that demonstrate the uses of QuantumLeap in two different contexts and showed that complex gestural interfaces could be built in relatively few steps. However, these two examples only cover a limited subset of possible gesture-based applications and are rather small compared to typical industry projects.

6.2.2 Usage. This strategy is used to highlight who can use the toolkit and how they can use it. In this paper, we conducted a usability study with seven participants (Section 5) to identify and correct issues with QuantumLeap. However, the high guidance in the "configuration" phase of the study poses a threat to its external validity [36], as our findings may not generalize to a real-life scenario in which no such guidance is available. In addition, the study could have featured an A/B comparison between the development of a gesture-based application with and without QuantumLeap to better highlight the advantages of using QuantumLeap.

6.2.3 Technical Evaluation. A technical evaluation serves to validate the performance of a toolkit, usually against a threshold or the state-of-the-art. In the case of our work, such evaluation is difficult to conduct as there is no real point of comparison. While some points could be benchmarked in isolation (*e.g.* latency, framerate, recognition rate), they are highly dependent on the selected configuration (*e.g.*, gesture set, recognizers, and segmenters).

7 CONCLUSION

In this paper, we introduced QuantumLeap, a framework for engineering gestural user interfaces based on the Leap Motion Controller by leveraging several barriers encountered in the development life cycle of gesture-based interactive applications. In order to facilitate the development of such LMC gesture-based UIs, manual development is replaced by configuring existing modules implemented in QuantumLeap in a pipeline way that can be saved in a configuration file for future usage. We performed this process for two different interactive applications to illustrate its versatility: a small application for browsing medical images and a large-scale application for browsing multimedia contents on large interaction surfaces. We also demonstrated its usability in an experiment with seven participants.

Future work can be envisioned at increasing levels of granularity. At the coarsest level (*i.e.*, the framework itself), we are planning on providing an easy installer for the framework, automatically proposing suitable configurations based on the selected gesture set, and allowing run-time modifications of the configuration through an API (*e.g.*, to replace a default gesture by a custom one directly from an application). We could also investigate how well software quality properties such as compatibility (*e.g.*, the ability to support external gesture datasets), maintainability (*e.g.*, the ability to expand QuantumLeap with new, more powerful modules), and portability (*e.g.*, the ability of to run on a wide variety of systems) – three key software quality factors defined by ISO [51]– are satisfied by QuantumLeap. For example, in order to assess maintainability, the cost for introducing a new module (*e.g.*, a new recognizer using deep learning and a new segmenter) can be estimated and compared to a manual integration of such a module in a handcrafted application.

At a finer level (*i.e.*, the framework modules), future work will focus on gesture segmentation, gesture recognition, and motion tracking sensors. We will explore novel methods for segmentation, such as Taranta *et al.*'s work [105] which trains a recognizer with automatically generated examples of parasitic gestures to distinguish them from those that are actually intended by the user. Equally interesting is Leiva *et al.*'s approach [67] that automatically generates synthetic gestures based on a template to train a recognizer, thus avoiding to ask several participants to acquire gestures. Combining this with a formal model of gestures is also possible, like DG3 [19] is exploiting for

real-time stroke gesture recognition. In addition, we will study the impact of combining multiple gesture recognizers on recognition accuracy.

Finally, we would like QuantumLeap to cover a much wider range of sensors. As such, we are currently working on adding support for the Microchip 3D Touchpad, Kinemic Band, and Google MediaPipe Hands. In addition, we will consider the inclusion of more sensors types (*e.g.*, Microsoft Kinect Azure, Intel RealSense cameras) and investigate combinations of multiple sensors to improve tracking accuracy. We hope that this work will positively impact research in gesture recognition and encourage the mainstream development of LMC-based gesture applications.

ACKNOWLEDGMENTS

The authors of this paper are very grateful to the anonymous EICS (meta-)reviewers for their constructive comments on earlier versions of this manuscript. They also thank the participants of the gesture study reported in the paper for their involvement. Arthur Sluÿters is funded by the "Fonds de la Recherche Scientifique - FNRS" under Grant n°40001931.

REFERENCES

[1] Roland Aigner, Daniel Wigdor, Hrvoje Benko, Michael Haller, David Lindbauer, Alexandra Ion, Shengdong Zhao, and Jeffrey Tzu Kwan Valino Koh. 2012. *Understanding Mid-Air Hand Gestures: A Study of Human Preferences in Usage of Gesture Types for HCI.* Technical Report MSR-TR-2012-111. https://www.microsoft.com/en-us/research/publication/understanding-mid-air-hand-gestures-a-study-of-human-preferences-in-usage-of-gesture-types-for-hci/

[2] Lisa Anthony, Radu-Daniel Vatavu, and Jacob O. Wobbrock. 2013. Understanding the Consistency of Users' Pen and Finger Stroke Gesture Articulation. In *Proceedings of Graphics Interface 2013* (Regina, Sascatchewan, Canada) *(GI '13)*. Canadian Information Processing Society, CAN, 87–94.

[3] Lisa Anthony and Jacob O. Wobbrock. 2010. A Lightweight Multistroke Recognizer for User Interface Prototypes. In *Proceedings of Graphics Interface 2010* (Ottawa, Ontario, Canada) *(GI '10)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 245–252. http://dl.acm.org/citation.cfm?id=1839214.1839258

[4] Lisa Anthony and Jacob O. Wobbrock. 2012. $N-ProTractor: A Fast and Accurate Multistroke Recognizer. In *Proceedings of Graphics Interface 2012* (Toronto, Ontario, Canada) *(GI '12)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 117–120. http://dl.acm.org/citation.cfm?id=2305276.2305296

[5] Daniel Ashbrook and Thad Starner. 2010. MAGIC: A Motion Gesture Design Tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 2159–2168. https://doi.org/10.1145/1753326.1753653

[6] Aaron Bangor, Philip T. Kortum, and James T. Miller. 2008. An Empirical Evaluation of the System Usability Scale. *International Journal of Human–Computer Interaction* 24, 6 (2008), 574–594. https://doi.org/10.1080/10447310802205776

[7] Mohamed-Ikbel Boulabiar, Gilles Coppin, and Franck Poirier. 2014. The Study of the Full Cycle of Gesture Interaction, The Continuum between 2D and 3D. In *Human-Computer Interaction. Advanced Interaction Modalities and Techniques*, Masaaki Kurosu (Ed.). Springer International Publishing, Cham, 24–35.

[8] Sanders Brndon. 2014. *Mastering Leap Motion.* Packt Publishing, Birmingham.

[9] John Brooke. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* (1996), 189–194.

[10] R. Brunelli. 2009. *Template Matching Techniques in Computer Vision: Theory and Practice.* John Wiley & Sons, New York.

[11] Ashley Bush and Sandeep Purao. 2011. Mapping UML Techniques to Design Activities. In *Information Modeling in the New Millennium*, Matti Rossi and Keng Siau (Eds.). IDEA Publishing Group, Chapter 12, 199–217.

[12] Minghao Cai and Jiro Tanaka. 2019. Go Together: Providing Nonverbal Awareness Cues to Enhance Co-Located Sensation in Remote Communication. *Hum.-Centric Comput. Inf. Sci.* 9, 1, Article 180 (Dec. 2019), 25 pages. https://doi.org/10.1186/s13673-019-0180-y

[13] John C. Callender and H. G. Osburn. 1979. An Empirical Comparison of Coefficient Alpha, Guttman's Lambda - 2, and MSPLIT Maximized Split-Half Reliability Estimates. *Journal of Educational Measurement* 16, 2 (1979), 89–99. http://www.jstor.org/stable/1434452

[14] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for multi-target user interfaces. *Interact. Comput.* 15, 3 (2003), 289–308. https://doi.org/10.1016/S0953-5438(03)00010-9

[15] F. M. Caputo, S. Burato, G. Pavan, T. Voillemin, H. Wannous, J. P. Vandeborre, M. Maghoumi, E. M. Taranta II, A. Razmjoo, J. J. LaViola Jr., F. Manganaro, S. Pini, G. Borghi, R. Vezzani, R. Cucchiara, H. Nguyen, M. T. Tran, and A.

Giachetti. 2019. Online Gesture Recognition. In *Eurographics Workshop on 3D Object Retrieval*, Silvia Biasotti, Guillaume Lavoué, and Remco Veltkamp (Eds.). The Eurographics Association, 93–102. https://doi.org/10.2312/3dor.20191067

[16] Fabio Marco Caputo and Andrea Giachetti. 2015. Evaluation of Basic Object Manipulation Modes for Low-Cost Immersive Virtual Reality. In *Proceedings of the 11th Biannual Conference on Italian SIGCHI Chapter* (Rome, Italy) *(CHItaly 2015)*. Association for Computing Machinery, New York, NY, USA, 74–77. https://doi.org/10.1145/2808435.2808439

[17] F. M. Caputo, P. Prebianca, A. Carcangiu, L. D. Spano, and A. Giachetti. 2017. A 3 Cent Recognizer: Simple and Effective Retrieval and Classification of Mid-Air Gestures from Single 3D Traces. In *Proceedings of the Conference on Smart Tools and Applications in Computer Graphics* (Catania, Italy) *(STAG '17)*. Eurographics Association, Goslar, DEU, 9–15. https://doi.org/10.2312/stag.20171221

[18] Fabio M. Caputo, Pietro Prebianca, Alessandro Carcangiu, Lucio D. Spano, and Andrea Giachetti. 2018. Comparing 3D trajectories for simple mid-air gesture recognition. *Computers & Graphics* 73 (2018), 17 – 25. https://doi.org/10.1016/j.cag.2018.02.009

[19] Alessandro Carcangiu, Lucio Davide Spano, Giorgio Fumera, and Fabio Roli. 2019. DEICTIC: A compositional and declarative gesture description based on hidden markov models. *International Journal of Human-Computer Studies* 122 (2019), 113 – 132. https://doi.org/10.1016/j.ijhcs.2018.09.001

[20] Géry Casiez, Nicolas Roussel, and Daniel Vogel. 2012. 1€ filter: a simple speed-based low-pass filter for noisy input in interactive systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2527–2530. https://doi.org/10.1145/2207676.2208639

[21] Ishan Chatterjee, Robert Xiao, and Chris Harrison. 2015. Gaze+Gesture: Expressive, Precise and Targeted Free-Space Interactions. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction* (Seattle, Washington, USA) *(ICMI '15)*. Association for Computing Machinery, New York, NY, USA, 131–138. https://doi.org/10.1145/2818346.2820752

[22] Yineng Chen, Xiaojun Su, Feng Tian, Jin Huang, Xiaolong (Luke) Zhang, Guozhong Dai, and Hongan Wang. 2016. Pactolus: A Method for Mid-Air Gesture Segmentation within EMG. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI EA '16)*. Association for Computing Machinery, New York, NY, USA, 1760–1765. https://doi.org/10.1145/2851581.2892492

[23] Hong Cheng, Lu Yang, and Zicheng Liu. 2016. Survey on 3D Hand Gesture Recognition. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 9 (2016), 1659–1673. https://doi.org/10.1109/TCSVT.2015.2469551

[24] Cheng-Chin Chiang, Ren-Hong Wang, and Bo-Ruei Chen. 2017. Recognizing Arbitrarily Connected and Superimposed Handwritten Numerals in Intangible Writing Interfaces. *Pattern Recogn.* 61, C (Jan. 2017), 15–28. https://doi.org/10.1016/j.patcog.2016.07.018

[25] Eunjung Choi, Heejin Kim, and Min K. Chung. 2014. A taxonomy and notation method for three-dimensional hand gestures. *International Journal of Industrial Ergonomics* 44, 1 (Jan. 2014), 171–188. https://doi.org/10.1016/j.ergon.2013.10.011

[26] Andrew Clark and Deshendran Moodley. 2016. A System for a Hand Gesture-Manipulated Virtual Reality Environment. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists* (Johannesburg, South Africa) *(SAICSIT '16)*. Association for Computing Machinery, New York, NY, USA, Article 10, 10 pages. https://doi.org/10.1145/2987491.2987511

[27] Alex Colgan. 2017. How Does the Leap Motion Controller Work? https://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

[28] Harisson Cook, Quang Vinh Nguyen, Simeone Simoff, and Mao Lin Huang. 2016. Enabling Gesture Interaction with 3D Point Cloud. In *Proceedings of the 24th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, Vol. 2602. Computer Science Research Notes CSRN, 59–68. https://opus.lib.uts.edu.au/handle/10453/45407

[29] Maria Francesca Costabile, Piero Mussio, Loredana Parasiliti Provenza, and Antonio Piccinno. 2008. End Users as Unwitting Software Developers. In *Proceedings of the 4th International Workshop on End-User Software Engineering* (Leipzig, Germany) *(WEUSE '08)*. Association for Computing Machinery, New York, NY, USA, 6–10. https://doi.org/10.1145/1370847.1370849

[30] Elena-Gina Craciun, Ionela Rusu, and Radu-Daniel Vatavu. 2016. Free-Hand Gesture Recognizer Pseudocode. http://www.eed.usv.ro/mintviz/projects/GIVISIMP/data/Pseudocode2.pdf. [Online; accessed 09-August-2020].

[31] Zachary A. Daniels, Steven R. Stinson, Shenchi Tian, Evan Mulbry, and Brian Y. Chen. 2014. A Gesture-Based Interface for the Exploration and Classification of Protein Binding Cavities. In *Proceedings of the 2014 Workshop on Mobile Augmented Reality and Robotic Technology-Based Systems* (Bretton Woods, New Hampshire, USA) *(MARS '14)*. Association for Computing Machinery, New York, NY, USA, 47–50. https://doi.org/10.1145/2609829.2609838

[32] Roberto De Prisco, Delfina Malandrino, Gianluca Zaccagnino, and Rocco Zaccagnino. 2016. Natural User Interfaces to Support and Enhance Real-Time Music Performance. In *Proceedings of the International Working Conference on*

Advanced Visual Interfaces (Bari, Italy) *(AVI '16)*. Association for Computing Machinery, New York, NY, USA, 204–211. https://doi.org/10.1145/2909132.2909249

[33] Giannis Drossis, Dimitris Grammenos, Chryssi Birliraki, and Constantine Stephanidis. 2013. MAGIC: Developing a Multimedia Gallery Supporting mid-Air Gesture-Based Interaction and Control. In *HCI International 2013 - Posters' Extended Abstracts*, Constantine Stephanidis (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 303–307.

[34] Richard O. Duda, Peter E. Hart, and David G. Stork. 2000. *Pattern Classification*. Wiley & Sons, New York.

[35] Ana Belén Erazo and Jorge Luis Pérez Medina. 2020. Algorithmic Efficiency of Stroke Gesture Recognizers: a Comparative Analysis. *International Journal on Advanced Science, Engineering and Information Technology* 10, 2 (2020), 438–446. https://doi.org/10.18517/ijaseit.10.2.10807

[36] Robert Feldt and Ana Magazinius. 2010. Validity Threats in Empirical Software Engineering Research - An Initial Survey. In *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010), Redwood City, San Francisco Bay, CA, USA, July 1 - July 3, 2010*. Knowledge Systems Institute Graduate School, 374–379.

[37] Pedro M. Ferreira, Jaime S. Cardoso, and Ana Rebelo. 2019. On the Role of Multimodal Learning in the Recognition of Sign Language. *Multimedia Tools Appl.* 78, 8 (April 2019), 10035–10056. https://doi.org/10.1007/s11042-018-6565-5

[38] Ivo Aluízio Stinghen Filho, Estevam Nicolas Chen, Jucimar Maia da Silva Junior, and Ricardo da Silva Barboza. 2018. Gesture Recognition Using Leap Motion: A Comparison between Machine Learning Algorithms. In *ACM SIGGRAPH 2018 Posters* (Vancouver, British Columbia, Canada) *(SIGGRAPH '18)*. Association for Computing Machinery, New York, NY, USA, Article 65, 2 pages. https://doi.org/10.1145/3230744.3230750

[39] Zeqing Fu, Yiyi Deng, Xin Jia, Bin Gao, Xiaoming Zhu, and Yanlin Luo. 2016. Automated Brain Extraction and Associated 3D Inspection Layers for the Rhesus Macaque MRI Datasets. In *Proceedings of the 15th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry - Volume 1* (Zhuhai, China) *(VRCAI '16)*. Association for Computing Machinery, New York, NY, USA, 261–269. https://doi.org/10.1145/3013971.3013984

[40] Claire Galea and Chris Porter. 2018. Accessible Choral Ensembles for Visually Impaired Singers. In *Proceedings of the 32nd International BCS Human Computer Interaction Conference* (Belfast, United Kingdom) *(HCI '18)*. BCS Learning & Development Ltd., Swindon, GBR, Article 37, 10 pages. https://doi.org/10.14236/ewic/HCI2018.37

[41] Bogdan-Florin Gheran, Jean Vanderdonckt, and Radu-Daniel Vatavu. 2018. Gestures for Smart Rings: Empirical Results, Insights, and Design Implications. In *Proceedings of the 2018 Designing Interactive Systems Conference* (Hong Kong, China) *(DIS '18)*. Association for Computing Machinery, New York, NY, USA, 623–635. https://doi.org/10.1145/3196709.3196741

[42] Francisco Gomez-Donoso, Sergio Orts-Escolano, Alberto Garcia-Garcia, Jose Garcia-Rodriguez, John Alejandro Castro-Vargas, Sergiu Ovidiu-Oprea, and Miguel Cazorla. 2017. A Robotic Platform for Customized and Interactive Rehabilitation of Persons with Disabilities. *Pattern Recogn. Lett.* 99, C (Nov. 2017), 105–113. https://doi.org/10.1016/j.patrec.2017.05.027

[43] Rebecca A. Grier. 2015. How High is High? A Meta-Analysis of NASA-TLX Global Workload Scores. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 59, 1 (Sept. 2015), 1727–1731. https://doi.org/10.1177/1541931215591373 Publisher: SAGE Publications Inc.

[44] Daniela Grijincu, Miguel A. Nacenta, and Per Ola Kristensson. 2014. User-Defined Interface Gestures: Dataset and Analysis. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces* (Dresden, Germany) *(ITS '14)*. Association for Computing Machinery, New York, NY, USA, 25–34. https://doi.org/10.1145/2669485.2669511

[45] Louis Guttman. 1945. A basis for analyzing test-retest reliability. *Psychometrika* 10 (December 1945), 255–282. https://doi.org/10.1007/BF02288892

[46] Tibor Guzsvinecz, Veronika Szucs, and Cecilia Sik-Lanyi. 2019. Suitability of the Kinect Sensor and Leap Motion Controller—A Literature Review. *Sensors* 19, 5 (2019), 25. https://doi.org/10.3390/s19051072

[47] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. North-Holland, 139 – 183. https://doi.org/10.1016/S0166-4115(08)62386-9

[48] Jinmiao Huang, Prakhar Jaiswal, and Rahul Rai. 2019. Gesture-based system for next generation natural and intuitive interfaces. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 33, 1 (2019), 54–68. https://doi.org/10.1017/S0890060418000045

[49] ECMA International. 2017. Standard ECMA-404 – The JSON Data Interchange Syntax. https://www.ecma-international.org/publications/standards/Ecma-404.htm

[50] ISO. 2018. *ISO/IEC 9241, Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. standard. International Standard Organization, Geneva. https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:en

[51] ISO. 2019. *ISO/IEC 25010 - Software Quality Product Standard*. standard. International Standard Organization, Geneva. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&limitstart=0

[52] Xianta Jiang, Zhen Gang Xiao, and Carlo Menon. 2018. Virtual Grasps Recognition Using Fusion of Leap Motion and Force Myography. *Virtual Real.* 22, 4 (Nov. 2018), 297–308. https://doi.org/10.1007/s10055-018-0339-2

[53] Vasilij Kiselev, Maxim Khlamov, and Kirill Chuvilin. 2019. Hand Gesture Recognition with Multiple Leap Motion Devices. In *Proceedings of the 24th Conference of Open Innovations Association FRUCT* (Moscow, Russia) *(FRUCT'24)*. FRUCT Oy, Helsinki, Uusimaa, FIN, Article 23, 7 pages.

[54] Barbara Kitchenham, Rialette Pretorius, David Budgen, O. Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman. 2010. Systematic literature reviews in software engineering – A tertiary study. *Information and Software Technology* 52, 8 (2010), 792–805. https://doi.org/10.1016/j.infsof.2010.03.006

[55] D. Kohlsdorf, T. Starner, and D. Ashbrook. 2011. MAGIC 2.0: A web tool for false positive prediction and prevention for gesture recognition systems. In *Face and Gesture 2011*. 1–6.

[56] Daniel Kyu Hwa Kohlsdorf and Thad E. Starner. 2013. MAGIC summoning: towards automatic suggesting and testing of gestures with low probability of false positives during use. *The Journal of Machine Learning Research* 14, 1 (Jan. 2013), 209–242.

[57] Panayiotis Koutsabasis and Panagiotis Vogiatzidakis. 2019. Empirical Research in Mid-Air Interaction: A Systematic Review. *International Journal of Human–Computer Interaction* 35, 18 (2019), 1747–1768. https://doi.org/10.1080/10447318.2019.1572352

[58] Sven Kratz and Jason Wiese. 2016. GestureSeg: Developing a Gesture Segmentation System Using Gesture Execution Phase Labeling by Crowd Workers. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Brussels, Belgium) *(EICS '16)*. Association for Computing Machinery, New York, NY, USA, 61–72. https://doi.org/10.1145/2933242.2933261

[59] Pradeep Kumar, Himaanshu Gauba, Partha Pratim Roy, and Debi Prosad Dogra. 2017. Coupled HMM-Based Multi-Sensor Data Fusion for Sign Language Recognition. *Pattern Recogn. Lett.* 86, C (Jan. 2017), 1–8.

[60] Pradeep Kumar, Partha Pratim Roy, and Debi Prosad Dogra. 2018. Independent Bayesian Classifier Combination Based Sign Language Recognition Using Facial Expression. *Inf. Sci.* 428, C (Feb. 2018), 30–48. https://doi.org/10.1016/j.ins.2017.10.046

[61] Pradeep Kumar, Rajkumar Saini, Partha Pratim Roy, and Debi Prosad Dogra. 2017. 3D Text Segmentation and Recognition Using Leap Motion. *Multimedia Tools Appl.* 76, 15 (Aug. 2017), 16491–16510. https://doi.org/10.1007/s11042-016-3923-z

[62] Gordon Kurtenbach, George Fitzmaurice, Thomas Baudel, and Bill Buxton. 1997. The Design of a GUI Paradigm Based on Tablets, Two-hands, and Transparency. In *Proceedings of the ACM International Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '97)*. ACM, New York, NY, USA, 35–42. https://doi.org/10.1145/258549.258574

[63] Joseph J. LaViola. 2003. Double exponential smoothing: an alternative to Kalman filter-based predictive tracking. In *Proceedings of the workshop on Virtual environments 2003 (EGVE '03)*. Association for Computing Machinery, New York, NY, USA, 199–206. https://doi.org/10.1145/769953.769976

[64] Joseph J. LaViola. 2013. 3D Gestural Interaction: The State of the Field. *International Scholarly Research Notices* 2013, Article 514641 (2013), 18 pages. https://doi.org/10.1155/2013/514641

[65] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the ACM Int. Conf. on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3173574.3173610

[66] Luis A. Leiva, Daniel Martín-Albo, and Réjean Plamondon. 2015. Gestures à Go Go: Authoring Synthetic Human-Like Stroke Gestures Using the Kinematic Theory of Rapid Movements. *ACM Transactions on Intelligent Systems Technology* 7, 2, Article 15 (nov 2015), 29 pages. https://doi.org/10.1145/2799648

[67] Luis A. Leiva, Daniel Martín-Albo, and Radu-Daniel Vatavu. 2018. GATO: Predicting Human Performance with Multistroke and Multitouch Gesture Input. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services* (Barcelona, Spain) *(MobileHCI '18)*. Association for Computing Machinery, New York, NY, USA, Article 32, 11 pages. https://doi.org/10.1145/3229434.3229478

[68] James R. Lewis and Jeff Sauro. 2009. The Factor Structure of the System Usability Scale. In *Human Centered Design*, Masaaki Kurosu (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 94–103.

[69] Feifei Li, Yujun Li, Baozhen Du, Hongji Xu, Hailiang Xiong, and Min Chen. 2019. A Gesture Interaction System Based on Improved Finger Feature and WE-KNN. In *Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence* (Chegndu, China) *(ICMAI 2019)*. Association for Computing Machinery, New York, NY, USA, 39–43. https://doi.org/10.1145/3325730.3325759

[70] Qiming Li, Chen Huang, Shengqing Lv, Zeyu Li, Yimin Chen, and Lizhuang Ma. 2017. An Human-Computer Interactive Augmented Reality System for Coronary Artery Diagnosis Planning and Training. *J. Med. Syst.* 41, 10 (Oct. 2017), 1–11. https://doi.org/10.1007/s10916-017-0805-5

[71] Hui Liang, Jian Chang, Ismail K. Kazmi, Jian J. Zhang, and Peifeng Jiao. 2017. Hand Gesture-Based Interactive Puppetry System to Assist Storytelling for Children. *Vis. Comput.* 33, 4 (April 2017), 517–531. https://doi.org/10.1007/s00371-016-1272-6

[72] Gustavo López, Luis Quesada, and Luis A. Guerrero. 2015. A Gesture-Based Interaction Approach for Manipulating Augmented Objects Using Leap Motion. In *Ambient Assisted Living. ICT-based Solutions in Real Life Situations*, Ian Cleland, Luis Guerrero, and José Bravo (Eds.). Springer International Publishing, Cham, 231–243.

[73] Naveen Madapana, Glebys Gonzalez, Rahul Taneja, Richard Rodgers, Lingsong Zhang, and Juan Wachs. 2019. Preference Elicitation: Obtaining Gestural Guidelines for PACS in Neurosurgery. *International Journal of Medical Informatics* 130 (07 2019). https://doi.org/10.1016/j.ijmedinf.2019.07.013

[74] Mehran Maghoumi and Joseph J. LaViola. 2019. DeepGRU: Deep Gesture Recognition Utility. In *Advances in Visual Computing*, George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Daniela Ushizima, Sek Chai, Shinjiro Sueda, Xin Lin, Aidong Lu, Daniel Thalmann, Chaoli Wang, and Panpan Xu (Eds.). Springer International Publishing, Cham, 16–31.

[75] Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. 2021. Two-Dimensional Stroke Gesture Recognition: A Survey. *ACM Comput. Surv.* 54, 7, Article 155 (jul 2021), 36 pages. https://doi.org/10.1145/3465400

[76] Nathan Magrofuoco, Paolo Roselli, Jean Vanderdonckt, Jorge Luis Pérez-Medina, and Radu-Daniel Vatavu. 2019. GestMan: A Cloud-Based Tool for Stroke-Gesture Datasets. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Valencia, Spain) *(EICS '19)*. Association for Computing Machinery, New York, NY, USA, Article 7, 6 pages. https://doi.org/10.1145/3319499.3328227

[77] Rajesh B. Mapari and Govind Kharat. 2016. American Static Signs Recognition Using Leap Motion Sensor. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies* (Udaipur, India) *(ICTCS '16)*. Association for Computing Machinery, New York, NY, USA, Article 67, 5 pages. https://doi.org/10.1145/2905055.2905125

[78] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. 2014. Hand gesture recognition with Leap Motion and Kinect devices. *2014 IEEE International Conference on Image Processing, ICIP 2014* (2014), 1565–1569. https://doi.org/10.1109/ICIP.2014.7025313

[79] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. 2016. Hand Gesture Recognition with Jointly Calibrated Leap Motion and Depth Sensor. *Multimedia Tools Appl.* 75, 22 (Nov. 2016), 14991–15015. https://doi.org/10.1007/s11042-015-2451-6

[80] Roi Méndez, Julián Flores, Enrique Castelló, and José Ramón Viqueira. 2018. New Distributed Virtual TV Set Architecture for a Synergistic Operation of Sensors and Improved Interaction between Real and Virtual Worlds. *Multimedia Tools Appl.* 77, 15 (Aug. 2018), 18999–19025. https://doi.org/10.1007/s11042-017-5353-y

[81] K. Ngan, D.N. Straub, and P. Bartello. 2004. Three-dimensionalization of freely-decaying two-dimensional turbulence. *Physics of Fluids* 16, 8 (2004), 2918–2932. https://doi.org/10.1063/1.1763191 arXiv:https://doi.org/10.1063/1.1763191

[82] Vinoth Pandian Sermuga Pandian and Sarah Suleri. 2020. NASA-TLX Web App: An Online Tool to Analyse Subjective Workload. arXiv:2001.09963 [cs.HC]

[83] Chanho Park, Hyunwoo Cho, Sangheon Park, Young-Suk Yoon, and Sung-Uk Jung. 2019. HandPoseMenu: Hand Posture-Based Virtual Menus for Changing Interaction Mode in 3D Space. In *Proceedings of the 2019 ACM International Conference on Interactive Surfaces and Spaces* (Daejeon, Republic of Korea) *(ISS '19)*. Association for Computing Machinery, New York, NY, USA, 361–366. https://doi.org/10.1145/3343055.3360752

[84] D. Parsons, Awais Rashid, Andreas Speck, and Alexandru C. Telea. 1999. A "framework" for object oriented frameworks design. In *Proceedings Technology of Object-Oriented Languages and Systems. TOOLS 29 (Cat. No.PR00275)*. 141–151. https://doi.org/10.1109/TOOLS.1999.779007

[85] Vik Parthiban and Ashley Jieun Lee. 2019. LUI: A Multimodal, Intelligent Interface for Large Displays. In *The 17th International Conference on Virtual-Reality Continuum and Its Applications in Industry* (Brisbane, QLD, Australia) *(VRCAI '19)*. Association for Computing Machinery, New York, NY, USA, Article 48, 2 pages. https://doi.org/10.1145/3359997.3365743

[86] Julien Pauchot, Laetitia Di Tommaso, Ahmed Lounis, Mourad Benassarou, Pierre Mathieu, Dominique Bernot, and Sébastien Aubry. 2015. Leap Motion Gesture Control With Carestream Software in the Operating Room to Control Imaging: Installation Guide and Discussion. *Surgical Innovation* 22, 6 (2015), 615–620. https://doi.org/10.1177/1553350615587992 PMID: 26002115.

[87] Andriy Pavlovych and Wolfgang Stuerzlinger. 2009. The tradeoff between spatial jitter and latency in pointing tasks. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems (EICS '09)*. Association for Computing Machinery, New York, NY, USA, 187–196. https://doi.org/10.1145/1570433.1570469

[88] Thammathip Piumsomboon, Adrian Clark, Mark Billinghurst, and Andy Cockburn. 2013. User-Defined Gestures for Augmented Reality. In *Human-Computer Interaction – INTERACT 2013 (Lecture Notes in Computer Science)*, Paula Kotzé, Gary Marsden, Gitte Lindgaard, Janet Wesson, and Marco Winckler (Eds.). Springer, Berlin, Heidelberg, 282–299.

https://doi.org/10.1007/978-3-642-40480-1_18

[89] Wolfgang Pree. 1994. Meta Patterns - A Means For Capturing the Essentials of Reusable Object-Oriented Design. In *Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP '94)*. Springer-Verlag, Berlin, Heidelberg, 150–162.

[90] Alexandros Preventis, Kostas Stravoskoufos, Stelios Sotiriadis, and Euripides G. M. Petrakis. 2014. Interact: Gesture Recognition in the Cloud. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC '14)*. IEEE Computer Society, USA, 501–502. https://doi.org/10.1109/UCC.2014.71

[91] Siddharth S. Rautaray and Anupam Agrawal. 2015. Vision based hand gesture recognition for human computer interaction: a survey. *Artif. Intell. Rev.* 43, 1 (2015), 1–54. https://doi.org/10.1007/s10462-012-9356-9

[92] Quentin Roy, Sylvain Malacria, Yves Guiard, Eric Lecolinet, and James Eagan. 2013. *Augmented Letters: Mnemonic Gesture-Based Shortcuts.* Association for Computing Machinery, New York, NY, USA, 2325–2328. https://doi.org/10.1145/2470654.2481321

[93] Dean Rubine. 1991. Specifying Gestures by Example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. ACM, New York, NY, USA, 329–337. https://doi.org/10.1145/122718.122753

[94] Ahmed R. Sadik, Bodo Urban, and Omar Adel. 2017. Using Hand Gestures to Interact with an Industrial Robot in a Cooperative Flexible Manufacturing Scenario. In *Proceedings of the 3rd International Conference on Mechatronics and Robotics Engineering* (Paris, France) *(ICMRE 2017)*. Association for Computing Machinery, New York, NY, USA, 11–16. https://doi.org/10.1145/3068796.3068801

[95] Jeff Sauro. 2010. *A practical guide to measuring usability: 72 answers to the most common questions about quantifying the usability of websites and software.* Measuring Usability LCC.

[96] Jacob Schioppo, Zachary Meyer, Diego Fabiano, and Shaun Canavan. 2019. Sign Language Recognition: Learning American Sign Language in a Virtual Environment. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI EA '19)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3290607.3313025

[97] Jia Sheng. 2004. *A Study of AdaBoost in 3D Gesture Recognition.* technical report CSC2515. Department of Computer Science, University of Toronto. http://www.dgp.toronto.edu/~jsheng/doc/CSC2515/Report.pdf

[98] Beat Signer, U. Kurmann, and Moira C. Norrie. 2007. iGesture: A General Gesture Recognition Framework. In *9th International Conference on Document Analysis and Recognition (ICDAR 2007), 23-26 September, Curitiba, Paraná, Brazil*. IEEE Computer Society, 954–958. https://doi.org/10.1109/ICDAR.2007.4377056

[99] Merkourios Simos and Nikolaos Nikolaidis. 2016. Greek Sign Language Alphabet Recognition Using the Leap Motion Device. In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence* (Thessaloniki, Greece) *(SETN '16)*. Association for Computing Machinery, New York, NY, USA, Article 34, 4 pages. https://doi.org/10.1145/2903220.2903249

[100] Mischa Spiegelmock. 2013. *Leap Motion Development Essentials* (1 ed.). Packt Publishing. https://www.packtpub.com/eu/hardware-and-creative/leap-motion-development-essentials

[101] Ivo Stinghen and Bernardo Gatto. 2018. Gesture Recognition Using Leap Motion: A Machine Learning-based Controller Interface. In *17th Brazilian Symposium on Computer Games and Digital Entertainment* (Foz do Iguaçu, Brazil) *(SBGames '18)*. IEEE Computer Society.

[102] Eugene M. Taranta, Mehran Maghoumi, Corey R. Pittman, and Joseph J. LaViola. 2016. A Rapid Prototyping Approach to Synthetic Data Generation for Improved 2D Gesture Recognition. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. Association for Computing Machinery, New York, NY, USA, 873–885. https://doi.org/10.1145/2984511.2984525

[103] Eugene M. Taranta, II and Joseph J. LaViola, Jr. 2015. Penny Pincher: A Blazing Fast, Highly Accurate $-family Recognizer. In *Proceedings of the 41st Graphics Interface Conference* (Halifax, Nova Scotia, Canada) *(GI '15)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 195–202. http://dl.acm.org/citation.cfm?id=2788890.2788925

[104] Eugene M. Taranta II, Corey R. Pittman, Mehran Maghoumi, Mykola Maslych, Yasmine M. Moolenaar, and Joseph J. Laviola Jr. 2021. Machete: Easy, Efficient, and Precise Continuous Custom Gesture Segmentation. *ACM Trans. Comput.-Hum. Interact.* 28, 1, Article 5 (Jan. 2021), 46 pages. https://doi.org/10.1145/3428068

[105] Eugene M. Taranta II, Amirreza Samiei, Mehran Maghoumi, Pooya Khaloo, Corey R. Pittman, and Joseph J. LaViola Jr. 2017. Jackknife: A Reliable Recognizer with Few Samples and Many Modalities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 5850–5861. https://doi.org/10.1145/3025453.3026002

[106] Marlon Tayag, Bituin Ronie, and Reyes Marvin. 2021. Leap Motion Controller Enabled Simulations of Personal Computer Assembly Programs. In *Proceedings of the 4th International Conference on Software Engineering and Information Management* (Yokohama, Japan) *(ICSIM '21)*. Association for Computing Machinery, New York, NY, USA, 17–21. https://doi.org/10.1145/3451471.3451474

[107] Pakpoom Thaweesitthichat, Thuchchai Jiamsorn, Proadpran Punyabukkana, Ekapol Chuangsuwanich, and Atiwong Suchato. 2018. Thai Sign Language Recognition with Leap Motion. In *Proceedings of the 12th International Convention on Rehabilitation Engineering and Assistive Technology* (Shanghai, China) *(i-CREATe 2018)*. Singapore Therapeutic, Assistive & Rehabilitative Technologies (START) Centre, Midview City, SGP, 47–50.

[108] Michail Theofanidis, Saif Iftekar Sayed, Alexandros Lioulemes, and Fillia Makedon. 2017. VARM: Using Virtual Reality to Program Robotic Manipulators. In *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments* (Island of Rhodes, Greece) *(PETRA '17)*. Association for Computing Machinery, New York, NY, USA, 215–221. https://doi.org/10.1145/3056540.3056541

[109] Enkhtogtokh Togootogtokh, Timothy K. Shih, W. G. Kumara, Shih-Jung Wu, Shih-Wei Sun, and Hon-Hang Chang. 2018. 3D Finger Tracking and Recognition Image Processing for Real-Time Music Playing with Depth Sensors. *Multimedia Tools Appl.* 77, 8 (April 2018), 9233–9248. https://doi.org/10.1007/s11042-017-4784-9

[110] Van Thanh Tran, Jaewoon Lee, Dongho Kim, and Young-Sik Jeong. 2016. Easy-to-Use Virtual Brick Manipulation Techniques Using Hand Gestures. *J. Supercomput.* 72, 7 (July 2016), 2752–2766. https://doi.org/10.1007/s11227-015-1588-4

[111] Jean Vanderdonckt, Paolo Roselli, and Jorge Luis Pérez-Medina. 2018. !!FTL, an Articulation-Invariant Stroke Gesture Recognizer with Controllable Position, Scale, and Rotation Invariances. In *Proceedings of the 20th ACM International Conference on Multimodal Interaction* (Boulder, CO, USA) *(ICMI '18)*. Association for Computing Machinery, New York, NY, USA, 125–134. https://doi.org/10.1145/3242969.3243032

[112] Radu-Daniel Vatavu. 2011. The Effect of Sampling Rate on the Performance of Template-Based Gesture Recognizers. In *Proceedings of the 13th International Conference on Multimodal Interfaces* (Alicante, Spain) *(ICMI '11)*. Association for Computing Machinery, New York, NY, USA, 271–278. https://doi.org/10.1145/2070481.2070531

[113] Radu-Daniel Vatavu. 2013. The impact of motion dimensionality and bit cardinality on the design of 3D gesture recognizers. *International Journal of Human-Computer Studies* 71, 4 (2013), 387 – 409. https://doi.org/10.1016/j.ijhcs.2012.11.005

[114] Radu-Daniel Vatavu. 2017. Improving Gesture Recognition Accuracy on Touch Screens for Users with Low Vision. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 4667–4679. https://doi.org/10.1145/3025453.3025941

[115] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2012. Gestures As Point Clouds: A $P Recognizer for User Interface Prototypes. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction* (Santa Monica, California, USA) *(ICMI '12)*. ACM, New York, NY, USA, 273–280. https://doi.org/10.1145/2388676.2388732

[116] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2018. $Q: A Super-quick, Articulation-invariant Stroke-gesture Recognizer for Low-resource Devices. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services* (Barcelona, Spain) *(MobileHCI '18)*. ACM, New York, NY, USA, Article 23, 12 pages. https://doi.org/10.1145/3229434.3229465

[117] Radu-Daniel Vatavu and Stefan Pentiuc. 2008. Multi-Level Representation of Gesture as Command for Human Computer Interaction. *Computing and Informatics* 27 (01 2008), 837–851.

[118] Christina Volioti, Sotiris Manitsaris, Edgar Hemery, Stelios Hadjidimitriou, Vasileios Charisis, Leontios Hadjileontiadis, Eleni Katsouli, Fabien Moutarde, and Athanasios Manitsaris. 2018. A Natural User Interface for Gestural Expression and Emotional Elicitation to Access the Musical Intangible Cultural Heritage. *J. Comput. Cult. Herit.* 11, 2, Article 10 (April 2018), 20 pages. https://doi.org/10.1145/3127324

[119] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. 2013. Analysis of the Accuracy and Robustness of the Leap Motion Controller. *Sensors* 13 (05 2013), 6380–6393. https://doi.org/10.3390/s130506380

[120] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. 2007. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology* (Newport, Rhode Island, USA) *(UIST '07)*. ACM, New York, NY, USA, 159–168. https://doi.org/10.1145/1294211.1294238

[121] Lin Yang, Long Ye, Wei Zhong, Yingying Zhang, and Qin Zhang. 2017. A Real-Time Interactive System Based on Hand Gesture Recognition in Virtual Fitting. In *LNCS on Transactions on Edutainment XIII - Volume 10092*. Springer-Verlag, Berlin, Heidelberg, 86–96. https://doi.org/10.1007/978-3-662-54395-5_8

[122] Yuan Yao, Po-Tsung Chiu, and Wai-Tat Fu. 2017. A Gestural Interface for Practicing Children's Spatial Skills. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces Companion* (Limassol, Cyprus) *(IUI '17 Companion)*. Association for Computing Machinery, New York, NY, USA, 43–47. https://doi.org/10.1145/3030024.3038265

[123] Mais Yasen and Shaidah Jusoh. 2019. A systematic review on hand gesture recognition techniques, challenges and applications. *PeerJ Computer Science* 5 (Sept. 2019), e218. https://doi.org/10.7717/peerj-cs.218

[124] Ge Yu, Ji Liang, and Lili Guo. 2018. Multi-Modal Interaction for Space Telescience of Fluid Experiments. In *Proceedings of the 2018 International Conference on Artificial Intelligence and Virtual Reality* (Nagoya, Japan) *(AIVR 2018)*. Association

for Computing Machinery, New York, NY, USA, 35–41. https://doi.org/10.1145/3293663.3293672

[125] Wei Zeng, Cong Wang, and Qinghui Wang. 2018. Hand Gesture Recognition Using Leap Motion via Deterministic Learning. *Multimedia Tools Appl.* 77, 21 (Nov. 2018), 28185–28206.

[126] Shumin Zhai, Per Ola Kristensson, Caroline Appert, Tue Haste Anderson, and Xiang Cao. 2012. Foundational Issues in Touch-Surface Stroke Gesture Design — An Integrative Review. *Foundations and Trends in Human–Computer Interaction* 5, 2 (2012), 97–205. https://doi.org/10.1561/1100000012

[127] Andrew Jie Zhou and Grace Hui Yang. 2018. Minority Report by Lemur: Supporting Search Engine with Virtual Reality. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (Ann Arbor, MI, USA) *(SIGIR '18)*. Association for Computing Machinery, New York, NY, USA, 1329–1332. https://doi.org/10.1145/3209978.3210179

[128] Alessandro Zocco, Matteo D. Zocco, Antonella Greco, Salvatore Livatino, and Lucio Tommaso Paolis. 2015. Touchless Interaction for Command and Control in Military Operations. In *Proceedings of the Second International Conference on Augmented and Virtual Reality - Volume 9254*. Springer-Verlag, Berlin, Heidelberg, 432–445. https://doi.org/10.1007/978-3-319-22888-4_32

[129] Yi-Bo Zou, Yi-Min Chen, Ming-Ke Gao, Quan Liu, Si-Yu Jiang, Jia-Hui Lu, Chen Huang, Ze-Yu Li, and Dian-Hua Zhang. 2017. Coronary Heart Disease Preoperative Gesture Interactive Diagnostic System Based on Augmented Reality. *J. Med. Syst.* 41, 8 (Aug. 2017), 1–18. https://doi.org/10.1007/s10916-017-0768-6

A DESCRIPTION OF THE PROVIDED MODULES

This appendix provides a short description of the modules provided with the QuantumLeap framework. If necessary, developers may write new modules to better fit their needs.

A.1 Sensors

A.1.1 Leap Motion Controller. A module that connects to the LMC and fetches frames using its JavaScript API. It returns the position of a set of articulations of the hand at one instant in time.

A.2 Filters

A.2.1 1€. A simple and efficient low-pass filter inspired by the $-family of recognizers [20]. Designed for interactive systems, it relies on an adaptive cutoff frequency that allows it to greatly reduce jitter at low speed while limiting lag at high speed.

A.3 Static and Dynamic Datasets

A.3.1 QLDynamic. A dataset composed of 17 dynamic gestures performed by four participants. Each participant performed each gesture four times above the LMC. We used this dataset in the two applications described in Sections 4.1 and 4.2.

A.3.2 QLStatic. A dataset composed of 5 static poses performed by only one participant above the LMC. It was initially used in the application described in Section 4.2.

A.4 Static Recognizers

QuantumLeap comes with two modules for static gesture recognition. Table 3 summarizes their invariance properties.

Recognizer	Type of invariance			
	Position	Scale	Rotation	Direction
$P^3+	●	●	○	●
GPSDa	●	●	●	●

Table 3. Invariance properties of the static recognizers.

A.4.1 $P^3+.* A 3D version of *$P+ algorithm [114]. It is position-, direction-, and scale-invariant [62], but not rotation-invariant. It is described into more details in Appendix A.7.

A.4.2 GPSDa. A novel position-, scale-, direction-, and rotation-invariant recognizer for hand poses regardless of the position of the hand. Its rotation-invariance makes it well-suited to situations where hand poses should be recognized regardless of their direction (*e.g.*, rotate a picture by performing a "grab" static gesture and gradually rotating the hand).

A.5 Analyzers

A.5.1 Basic Analyzer. This module extracts four values from each frame: (1) rotation, the rotation of the hand compared to the previous frame; (2) pinch, the ratio of the distances between the index and the thumb of the current and the previous frames; (3) translation, the displacement of the palm compared to the previous frame; and (4) thumbVector, a 3D vector that gives the orientation of the thumb.

A.6 Segmenters

Three simple segmentation methods have currently been implemented to demonstrate the possibilities of QuantumLeap. We are planning on implementing more advanced segmentation techniques such as Taranta *et al.*'s Machete [104] and Chen *et al.*'s Pactolus [22]. However, support for Pactolus will only be relevant once EMG sensors are supported by QuantumLeap.

A.6.1 Zoning Segmenter. A segmenter that triggers gesture recognition when one or more selected points are within a bounding box above the sensor. After all the selected points have left, the recorded frames are sent to the recognizer.

A.6.2 Threshold Segmenter. A sensor that triggers gesture recognition when one or more parameters meet a pre-defined condition. The conditions (*e.g.*, the value of a parameter exceeds/falls below some threshold) can be manually configured from the UI. Once the conditions are no longer met, the recorded frames are sent to the recognizer.

A.6.3 Window Segmenter. A segmenter that triggers gesture recognition at regular intervals by relying on fixed-size buffers called *windows*. To adapt to varying gesture duration, it can be configured with any number of windows. Each time a new frame is received, it is pushed in the window(s), while the oldest frame is removed. Every fixed number of frames, the buffered gesture data are sent to the recognizer.

A.7 Dynamic Recognizers

QuantumLeap comes with ten modules for dynamic gesture recognition. Table 4 compares their invariance properties.

Recognizer	Type of invariance			
	Position	Scale	Rotation	Direction
$P^3	●	●	○	●
$Q^3	●	●	○	●
$P^3+	●	●	○	●
$P^3+X	●	●	○	◐
3 cent	●	●	○	○
Jackknife	●	●	○	○
$F	●	●	○	●
FreeHandUni	●	●	○	●
Rubine3D	●	●	○	○
Rubine-Sheng	●	●	○	○

Table 4. Invariance properties of the dynamic recognizers.

A.7.1 $P^3. A generalization of $P [115] towards supporting 3D multi-stroke gestures which is similar to the $P3D recognizer implemented by[28]. However, unlike $P3D the recognizer $P^3 does not support 3D static poses and 2D dynamic gestures recognition. To keep memory usage and execution time low, gestures are represented as unordered sets of points, called *point clouds*. Gesture recognition happens in two phases: normalization and cloud-matching. The normalization process is similar to other $-family algorithms and is divided into three steps: (1) resample the gesture to a fixed number of equidistant points; (2) scale the gesture uniformly to keep its shape; (3) translate the centroid of the point cloud to the origin, *i.e.*, $(0, 0, 0)$. The cloud-matching phase matches a gesture's point cloud to the point cloud of each template by associating each point from

the template to exactly one point from the gesture. It then computes the resulting distance, as the sum of Euclidean distances between all pairs of matching points, and returns the template that minimizes it. This recognizer is position-, direction-, and scale-invariant, but not rotation-invariant.

A.7.2 $Q³. A 3D variant of $Q [116], which achieved a 46X speedup on average over *$P* with no loss of accuracy. It brings two key changes to $P³: (1) early abandonment of templates, as soon as the distance exceeds the current shortest distance; (2) each point cloud has a 16×16×16 3D look-up table (LUT), where each location (x, y, z) refers to the closest point. These LUTs allow $Q³ to compute a lower bound of the distance between a gesture and a template in $O(n)$. If it exceeds the current shortest distance, the template can be rejected without computing the exact distance. It has the same invariance properties as $P³.

A.7.3 $P³+. A more accurate version of $P³, adapted from Vatavu's $P+ [114]. It brings three key improvements to $P³: (1) each point from the template can now be matched to more than one point of the gesture; (2) the distance between a gesture and a template takes into account the connections between consecutive points (in the form of their turning angles); (3) early abandonment of templates. This recognizer has the same invariance properties as $P³.

A.7.4 $P³+X. A variant of $P³+, which supports partial direction-invariance by keeping track of conflicting templates (*i.e.*, templates that represent the same gesture but drawn in different directions). If a gesture matches a conflicting template, its direction is compared with the direction of each conflicting template and the closest one is returned.

A.7.5 3 cent. An optimized 3D port of Wobbrock *et al.*'s $1 [120] by Caputo *et al.* [17], which recognizes uni-stroke gestures in two phases: the gesture is first normalized (similarly to *$P*) and its distance to each template is then computed as the sum of the squared distances of corresponding points. The template with the shortest distance is returned. 3¢ is position- and scale-invariant, but neither rotation- nor direction-invariant.

A.7.6 Jackknife. A general-purpose 3D gesture recognizer [105] that supports any number of articulations. Unlike most $-family recognizers, it represents gestures as time-series. It uses the nearest neighbor approach, where it compares a gesture to each template and returns the closest one. It uses *Dynamic Time Warping* as a distance measure but applies a series of correction factors to take into account differences in gesture scale and span. As a result, this recognizer is both position- and scale-invariant, but neither direction- nor rotation-invariant.

A.7.7 $F. A new recognizer that adds $P+'s flexible cloud matching [114] to $P³. As for most $-recognizers, the points of the candidate and the templates are resampled to equidistantly-spaced points, scaled within a unit box (isometric [111]), and translated so their centroid is at the origin $(0, 0, 0)$. The template with the lowest dissimilarity score is considered as the best matching template for the candidate. As opposed to $P³, $F's cloud matching is flexible, as points can be matched to more than one point. It consists of matching the points from the first cloud with their closest point from the second cloud, then matching the points from the second cloud that have not been matched yet with their closest point in the first cloud.

A.7.8 FreeHandUni. A recognizer derived from Vatavu *et al.*'s Free-Hand recognizer [30], which extends $P [115] to support 3D hand gestures. It replaces the hand pose structure with a 3D point structure (x, y, z). With this modification, FreehandUni improves $P³ using a flexible cloud matching based on a one-to-many alignment between points [114]. The pre-processing stays the same but the matching process is more flexible: each point of the template cloud is matched with the closest point from the candidate cloud, then the remaining points from the candidate cloud

are matched with the closest point from the template cloud. It returns the gesture class with the lowest dissimilarity score. FreeHandUni is different from $F in that the early abandoning is not implemented, to align the computational complexity to $P^3.

A.7.9 Rubine3D. Inspired by the iGesture framework [98], Rubine3D is a feature-based recognizer that combines a set of three individual 2D Rubine recognizers [93], one for each plane XY, YZ, and ZX. Before it can recognize 3D trajectories, Rubine3D pre-processes the training templates to compute weights for each feature of each gesture class. Then, it can recognize gestures in four steps: (1) it pre-processes the gesture by scaling and filtering its points; (2) it projects it onto each plane and extracts a feature vector $(f_1, .., f_{13})$ from each of the three projections; (3) for each projection, it selects the gesture class that maximizes the similarity score (computed by combining the weights of the gesture class with the feature vector of the gesture); (4) it determines the resulting class by merging the results from the three projections.

A.7.10 Rubine-Sheng. Rubine-Sheng (RS) is a 3D recognizer inspired by Rubine. It supports 3D gestures by adding three new features proposed in the AdaBoost recognizer [97] to Rubine's existing 13 features. Aside from this difference, it is extremely similar to its 2D counterpart.

Note. The implemented multipath recognizers include an optimization which avoids comparing the candidate gesture performed with one hand against the gestures performed with both hands in the training set.

B PSEUDOCODE OF IMPLEMENTED GESTURE RECOGNIZERS

We provide the pseudocode of the gesture recognizers implemented in QuantumLeap.

B.1 $P³ Recognizer

$P³ is based on Vatavu *et al.*'s $P[115]. The modifications to the original algorithm are highlighted in blue. Point is a structure with the `x`, `y`, `z`, and `strokeId` properties. Points is a list of Point and Templates is a list of Points with their associated gesture class data (*e.g.*, the name of the gesture). Please note that the templates should be pre-processed to improve performance, but this is not shown in the pseudocode in order to keep it concise.

$P³Recognizer(Points `pts`, Templates `templates`)

```
1:  n ← 32                                              ▷ Number of points
2:  Normalize(pts, n)
3:  score ← ∞
4:  for each template in templates do
5:      Normalize(template, n)                          ▷ Should be pre-processed
6:      d ← GreedyCloudMatch(pts, template, n)
7:      if d < score then
8:          score ← d
9:          result ← template
10: return template
```

GreedyCloudMatch(Points `ptsA`, Points `ptsB`, int `n`)

```
1: e ← .50
2: step ← ⌊n^(1−e)⌋
3: min ← ∞
4: for i = 0 to n − 1 step step do
5:     d_1 ← CloudDistance(ptsA, ptsB, n, i)
6:     d_2 ← CloudDistance(ptsB, ptsA, n, i)
7:     min ← Min(min, d_1, d_2)
8: return min
```

CloudDistance(Points ptsA, Points ptsB, int n, int start)

```
matched ← new bool [n]
sum ← 0
i ← start
repeat
    min ← ∞
    for each j such that not matched[j] do
        d ← EuclideanDistance(ptsA[i], ptsB[j])
        if d < min then
            min ← d
            index ← j
    matched[index] ← true
    weight ← 1 − ((i − start + n)%n)/n
    sum ← sum + weight ∗ min
    i ← (i + 1)%n
until i == start
return sum
```

Normalize(Points pts, int n)

```
pts ← Resample(pts, n)
Scale(pts)
TranslateToOrigin(pts, n)
```

Resample(Points pts, int n)

```
I ← PathLength(pts)/(n − 1)
D ← 0
resampledPts ← {pts[0]}
for each p_i in pts such that i ≥ 1 do
    if p_i.strokeId == p_{i−1}.strokeId then
        d ← EuclideanDistance(p_i, p_{i−1})
        if (D + d) ≥ I then
            q.x ← p_{i−1}.x + ((I − D)/d) ∗ (p_i.x − p_{i−1}.x)
            q.y ← p_{i−1}.y + ((I − D)/d) ∗ (p_i.y − p_{i−1}.y)
            q.z ← p_{i−1}.z + ((I − D)/d) ∗ (p_i.z − p_{i−1}.z)
            Append(resampledPts, q)
            Insert(pts, i, q)                    ▷ q will be the next p_i
            D ← 0
        else
            D ← D + d
while resampledPts.length ≤ n do
    Append(resampledPts, pts[pts.length - 1])
return resampledPts
```

PathLength(Points pts)

```
d ← 0
for each p_i in pts such that i ≥ 1 do
    if p_i.strokeId == p_{i−1}.strokeId then
        d ← d + EuclideanDistance(p_i, p_{i−1})
return d
```

Scale(Points pts)

```
x_min ← ∞, x_max ← 0, y_min ← ∞, y_max ← 0, z_min ← ∞, z_max ← 0
for each p in pts do
    x_min ← Min(x_min, p.x)
    y_min ← Min(y_min, p.y)
    z_min ← Min(z_min, p.z)
    x_max ← Max(x_max, p.x)
    y_max ← Max(y_max, p.y)
    z_max ← Max(z_max, p.z)
scale ← Max(x_max − x_min, y_max − y_min, z_max − z_min)
for each p in pts do
    p ← ((p.x − x_min)/scale, (p.y − y_min)/scale, (p.z − z_min)/scale, p.strokeId)
```

TranslateToOrigin(Points pts, int n)

```
c ← (0, 0, 0)
for each p in pts do                          ▷ Compute the centroid
    c ← (c.x + p.x, c.y + p.y, c.z + p.z)
c ← (c.x/n, c.y/n, c.z/n)
for each p in pts do                          ▷ Translate each point
    p ← (p.x − c.x, p.y − c.y, p.z − c.z, p.strokeId)
```

B.2 $P^3+ Recognizer

$P^3+ is an improved version of $P^3 based on Vatavu *et al.*'s $P+[114]. We only show the parts that were updated from $P^3. The modifications to the original algorithm ($P+) are highlighted in blue.

$P^3+Recognizer(Points pts, Templates templates)

```
n ← 32                                        ▷ Number of points
Normalize(pts, n)
score ← ∞
for each template in templates do
    Normalize(template, n)                    ▷ Should be pre-processed
    d ← Min(CloudDistance(pts, template, n, score),
            CloudDistance(template, pts, n, score))
    if d < score then
        score ← d
        result ← template
return template
```

CloudDistance(Points ptsA, Points ptsB, int n, float minSoFar)

```
matched ← new bool [n]
sum ← 0
for i = 0 to n − 1 do    ▷ Match points from cloud ptsA with points from ptsB; one-to-many matchings allowed
    min ← ∞
    for j = 0 to n − 1 do
        d ← PointDistance(ptsA[i], ptsB[j])
        if d < min then
            min ← d
            index ← j
    matched[index] ← true
    sum ← sum + min
    if sum ≥ minSoFar then
        return sum
    for each j such that not matched[j] do ▷ Match remaining points from cloud ptsB with points from ptsA; one-to-many matchings allowed
        min ← ∞
        for i = 0 to n − 1 do
            d ← PointDistance(ptsA[i], ptsB[j])
            if d < min then
                min ← d
        sum ← sum + min
        if sum ≥ minSoFar then
            return sum
        return sum
```

PointDistance(Point a, Point b)

```
return √((a.x − b.x)² + (a.y − b.y)² + (a.z − b.z)² + (a.θ − b.θ)²)
```

Normalize(Points pts, int n)

```
pts ← Resample(pts, n)
Scale(pts)
TranslateToOrigin(pts, n)
ComputeNormalizedTurningAngles(pts, n)
```

ComputeNormalizedTurningAngles(Points pts, int n)

```
pts[0].θ ← 0
pts[n].θ ← 0
for i = 2 to n − 2 do
    dpX ← (pts[i + 1].x − pts[i].x) ∗ (pts[i].x − pts[i − 1].x)
    dpY ← (pts[i + 1].y − pts[i].y) ∗ (pts[i].y − pts[i − 1].y)
    dpZ ← (pts[i + 1].z − pts[i].z) ∗ (pts[i].z − pts[i − 1].z)
    pts[i].θ ← (1/π) arccos( (dpX+dpY+dpZ) / (‖pts[i+1]−pts[i]‖ ∗ ‖pts[i]−pts[i−1]‖) )
```

B.3 $F Recognizer

$F is an improved version of $P^3 with the flexible cloud matching of $P+ [114]. We only show the parts of $F that were updated. The modifications to the original algorithms ($P^3 and $P+) are highlighted in blue.

```
$FRecognizer(Points pts, Templates templates)
 1: n ← 32                                                    ▷ Number of points
 2: Normalize(pts, n)
 3: score ← ∞
 4: for each template in templates do
 5:     Normalize(template, n)                                ▷ Should be pre-processed
 6:     d ← Min(CloudDistance(pts, template, n, score),
 7:             CloudDistance(template, pts, n, score))
 8:     if d < score then
 9:         score ← d
10:         result ← template
11: return template
```

```
CloudDistance(Points ptsA, Points ptsB, int n, float minSoFar)
 1: matched ← new bool [n]
 2: sum ← 0
 3: for i = 0 to n − 1 do   ▷ Match points from cloud ptsA with points from ptsB; one-to-many matchings allowed
 4:     min ← ∞
 5:     for j = 0 to n − 1 do
 6:         d ← EuclideanDistance(ptsA[i], ptsB[j])
 7:         if d < min then
 8:             min ← d
 9:             index ← j
10:     matched[index] ← true
11:     sum ← sum + min
12:     if sum ≥ minSoFar then
13:         return sum
14: for each j such that not matched[j] do   ▷ Match remaining points from cloud ptsB with points from ptsA; one-to-many matchings allowed
15:     min ← ∞
16:     for i = 0 to n − 1 do
17:         d ← EuclideanDistance(ptsA[i], ptsB[j])
18:         if d < min then
19:             min ← d
20:     sum ← sum + min
21:     if sum ≥ minSoFar then
22:         return sum
23: return sum
```

B.4 $P^3+X Recognizer

$P^3+X is based on Vatavu *et al.*'s $P3+[114]. We only show the parts that were modified from $P^3.

P^3+XRecognizer(Points pts, Templates templates)

```
n ← 32                                                          ▷ Number of points
conflictThreshold ← 1.25          ▷ Maximum score to consider the template as conflicting
loadedTemplates ← {templates[0]}
for each template in templates do                               ▷ Should be pre-processed
    ⟨template2, score⟩ ← RecognizeHelper(template, loadedTemplates, n)
    if (template2.name ≠ template.name) and (score < conflictThreshold) then
        conflicts[template] ← template2
        conflicts[template2] ← template
    Append(loadedTemplates, template)
template ← RecognizeHelper(pts, templates, n)
if template ∈ conflicts then          ▷ If two templates are conflicting, compare their direction
    s1 ← DirectionalSimilarity(pts, template, n)
    s2 ← DirectionalSimilarity(pts, conflicts[template], n)
    if s2 > s1 then
        template ← conflicts[template]
return template
```

RecognizeHelper(Points pts, Templates templates, int n)

```
Normalize(pts, n)
score ← ∞
for each template in templates do
    d ← Min(CloudDistance(pts, template, n, score),
            CloudDistance(template, pts, n, score))
    if d < score then
        score ← d
        result ← template
return ⟨template, score⟩
```

DirectionalSimilarity(Points ptsA, Point ptsB, int n)

```
padding ← 2
dist ← 0
for i = padding to n − (2 + padding) do
    dir_{A,x} ← ptsA[i+1].x − ptsA[i].x
    dir_{A,y} ← ptsA[i+1].y − ptsA[i].y
    dir_{A,z} ← ptsA[i+1].z − ptsA[i].z
    dir_{B,x} ← ptsB[i+1].x − ptsB[i].x
    dir_{B,y} ← ptsB[i+1].y − ptsB[i].y
    dir_{B,z} ← ptsB[i+1].z − ptsB[i].z
    dist ← dist + dir_{A,x} * dir_{B,x} + dir_{A,y} * dir_{B,y} + dir_{A,z} * dir_{B,z}
return dist
```

B.5 Q^3

Q^3 is an improved version of P^3 based on Vatavu *et al.*'s $Q[116]. We only show the parts that were updated from P^3. The modifications to the original algorithm ($Q) are highlighted in blue.

Q^3Recognizer(Points pts, Templates templates)

```
n ← 32                                              ▷ Number of points
m ← 16                                              ▷ Size of the LUT
Normalize(pts, n, m)
score ← ∞
for each template in templates do
    Normalize(template, n, m)                       ▷ Should be pre-processed
    d ← CloudMatch(pts, template, n, score)
    if d < score then
        score ← d
        result ← template
return template
```

CloudMatch(Points pts, Points template, int n, int min)

```
e ← .50
step ← ⌊n^(1−e)⌋
LB1 ← ComputeLowerBound(pts, template, step, template.LUT)
LB2 ← ComputeLowerBound(template, pts, step, pts.LUT)
min ← ∞
for i = 0 to n − 1 step step do
    if LB1[i/step] < min then
        min ← Min(min, CloudDistance(pts, template, n, i, min))
    if LB2[i/step] < min then
        min ← Min(min, CloudDistance(template, ps, n, i, min))
return min
```

CloudDistance(Points pts, Points template, int n, int start, float minSoFar)

```
unmatched ← {0, 1, 2, ..., n − 1}
i ← start
weight ← n
sum ← 0
repeat
    min ← ∞
    for each j in unmatched do
        d ← SqrEuclideanDistance(pts[i], template[j])
        if d < min then
            min ← d
            index ← j
    Remove(unmatched, index)
    sum ← sum + weight ∗ min
    if sum ≥ minSoFar then
        return sum
    weight ← weight − 1
    i ← (i + 1)%n
until i == start
return sum
```

ComputeLowerBound(Points pts, Points template, int step, int[] LUT)

```
LB ← new float[n/step + 1]                ▷ Multiple lower bounds, one for each starting point
SAT ← new float[n]                        ▷ Summed area table for fast computations
LB[0] ← 0
for i = 0 to n − 1 do                     ▷ First, compute the lower bound for starting point index 0
    index ← LUT [pts[i].x, pts[i].y, pts[i].z]
    d ← SqrEuclideanDistance(pts[i], template[index])
    if i == 0 then
        SAT[i] ← d
    else
        SAT[i] ← SAT[i − 1] + d
    LB[0] ← LB[0] + (n − i) ∗ d
for i ← step to n − 1 step step do        ▷ Compute the lower bound for the other starting points
    LB[i/step] ← LB[0] + i ∗ SAT[n − 1] − n ∗ SAT[i − 1]
return LB
```

Normalize(Points pts, int n, int m)

```
pts ← Resample(pts, n)
Scale(pts)
TranslateToOrigin(pts, n)
LUT ← ComputeLUT(m, pts, n)
```

Scale(Points pts, int m)

```
x_min ← ∞, x_max ← −∞, y_min ← ∞, y_max ← −∞, z_min ← ∞, z_max ← −∞
for each p in pts do
    x_min ← Min(x_min, p.x)
    y_min ← Min(y_min, p.y)
    z_min ← Min(z_min, p.z)
    x_max ← Max(x_max, p.x)
    y_max ← Max(y_max, p.y)
    z_max ← Max(z_max, p.z)
scale ← Max(x_max − x_min, y_max − y_min, z_max − z_min)/(m − 1)
for each p in pts do
    p ← ((p.x − x_min)/scale, (p.y − y_min)/scale, (p.z − z_min)/scale, p.strokeId)
```

ComputeLUT(Points pts, int n, int m)

```
1:  LUT ← new int[m, m, m]
2:  for x = 0 to m − 1 do
3:      for y = 0 to m − 1 do
4:          for z = 0 to m − 1 do
5:              min ← ∞
6:              for i = 0 to n − 1 do
7:                  d ← SqrEuclideanDistance(pts[i], newPoint(x, y, z))
8:                  if d < min then
9:                      min ← d
10:                     index ← i
11:             LUT[x, y, z] ← index
12: return LUT
```

SqrEuclideanDistance(Point a, Point b)

```
1:  return (a.x − b.x)^2 + (a.y − b.y)^2 + (a.z − b.z)^2
```

C TASKS

C.1 Configure QuantumLeap

C.1.1 Initial State. The QuantumLeap framework is running in the background and its UI is opened on the "Overview" page.

C.1.2 Steps.

(1) Go to the "Sensor(s)" page and select the "Leap Motion Controller" module. Rename it as "lmc" and save the modifications.
(2) Go to the "Static dataset(s)" page and select the "Leap Motion Dataset Loader" module. Select the "Simple dataset (static)" dataset and give it the "lmc" identifier. Select 100 templates per class. Save the modifications.
(3) Go to the "Static recognizer" page and select the "GPSDa" module. Select all the points of the right hand. Save the modifications.
(4) Go to the "Segmenter" page and select the "Sliding window" module. Configure it with one (1) window with a length of 20 frames. Select the right hand palm. Save the modifications.
(5) Go to the "Dynamic dataset(s)" page and select the "Leap Motion Dataset Loader" module. Select the "Simple dataset (dynamic)" dataset and give it the "lmc" identifier. Select 8 templates per class. Save the modifications.
(6) Go to the "Dynamic recognizer" page and select the "Jackknife" module. Configure it with the right hand palm. Save the modifications.
(7) Restart QuantumLeap by toggling the play/pause button in the "Overview" page.

C.2 Add Gesture Recognition to a Small Application

C.2.1 Initial State. Visual Studio Code is opened on the App.js file. Only this file will be modified. The application is running in development mode in a Google Chrome tab. The application refreshes automatically when the modifications to App.js are saved. The QuantumLeap framework is running in the background.

C.2.2 Steps.

(1) To be able use the QuantumLeap API, first import GestureHandler from the quantumleapjs module.
(2) Instantiate a new GestureHandler object without any option in the constructor of the App class. Assign it to a new instance variable of App (*e.g.*, this.gestureHandler). In the next steps, you will use the methods of GestureHandler to implement gesture recognition.
(3) In componentDidMount, add a call to the connect method of GestureHandler in order to connect to the QuantumLeap framework after the page has loaded.
(4) Make sure to disconnect from the QuantumLeap framework when the page is closed by adding a call to the disconnect method of GestureHandler in componentWillUnmount.
(5) Let's add the first gesture. The application will display the "swipe left" image each time a "swipe left" gesture is recognized. In componentDidMount, add an event listener for the "gesture" event that calls this.onLeftSwipe each time "swipe left" is recognized. You should use the addEventListener method of GestureHandler.
(6) You may notice that nothing happens when performing a "swipe left" gesture. To fix this, register the gesture to QuantumLeap by adding a call to the registerGestures method of GestureHandler with the type ("dynamic") and name ("swipe left") of the gesture. This will notify QuantumLeap that you want it to recognize the "swipe left" gesture.

(7) Now, modify the listener that you added at step 5 to display the "swipe right" image (by calling this.onRightSwipe) each time the corresponding gesture is performed. Add the "swipe right" gesture to the call to registerGestures.
(8) Modify the listener again to display the "point index" image (by calling this.onPoint) every time the user points his index.
(9) The "point index" gesture is static. You should thus add a new call to the registerGestures method of GestureHandler with the type ("static") and name ("point index") of the gesture.
(10) Now, modify the listener to display the "thumb" image (by calling this.onThumb) each time the corresponding gesture is performed. Add the "thumb" gesture to the call to registerGestures with the "point index" gesture.
(11) Modify the listener one last time to display the name and type of each gesture when they are recognized. You can do it by adding a call to this.onGesture (with the type and name of gesture as arguments) in the listener.
(12) Let's now add a text at the bottom of the screen to show if the application is connected to QuantumLeap. In componentDidMount, use the addEventListener method of GestureHandler to call this.setConnected with "true" every time a "connect" event is emitted.
(13) To reset the text when the application is disconnected from QuantumLeap, use the addEventListener method of GestureHandler to call this.setConnected with "false" every time a "disconnect" event is emitted.

Received July 2021; revised October 2021; accepted November 2021