University of Cagliari

PhD School in Electronic and Computer Engineering

XXXI cycle

Course in Computer Engineering

Combining declarative models and computer vision recognition algorithms for stroke gestures

---

Scientific Disciplinary Sector: ING-INF/05

Ph.D. Student:
Alessandro Carcangiu

Ph.D. Supervisors:
Prof. Fabio Roli
Prof. Lucio Davide Spano

Ph.D. Coordinator:
Prof. Fabio Roli

Final exam: Academic Year 2017 – 2018
Thesis defence: January – February 2019 Session

Abstract

Consumer-level devices that track user's gestures eased the design and the implementation of interactive applications relying on body movements as input. Gesture recognition based on computer vision and machine-learning focuses mainly on accuracy and robustness. The resulting classifiers label precisely gestures after their performance, but they do not provide intermediate information during their execution. Human-Computer Interaction research focused instead on providing an easy and effective guidance for performing and discovering interactive gestures. The compositional approaches developed for solving such problem provide information on both the whole gesture and on its sub-parts, but they exploit heuristic techniques that have a low recognition accuracy. In this thesis, we introduce two methods, DEICTIC and G-Gene, designed for establishing a compromise between accuracy and provided information. DEICTIC exploits a compositional and declarative description for stroke gestures. It uses basic Hidden Markov Models (HMMs) to recognise meaningful predefined primitives (gesture sub-parts) and it composes them to recognise complex gestures. It provides information for supporting gesture guidance and it reaches an accuracy comparable with state-of-the-art approaches on two datasets from the literature. Normalization of the gesture samples limits online recognition in the general case. G-Gene is a method for transforming compositional stroke gesture definitions into profile Hidden Markov Models (HMMs), able to provide both a good accuracy and information on gesture sub-parts. It supports online recognition without using any global feature, and it updates the information while receiving the input stream, with an accuracy useful for prototyping the interaction. We evaluated both approaches in a simplified development task with real developers, showing that they require less time and an effort comparable to compositional approaches, while the definition procedure and the perceived recognition accuracy is comparable to machine learning.

Contents

List of Figures

List of Tables

A Gianlu

Chapter 1

Introduction

Gestures represent an alternative and innovative modality with respect to traditional interactions e.g., through mouse and keyboard. During the '90s, consumer-level tracking technologies and interfaces supporting gestural input were a niche product, considering the hardware requirements and their cost. Such pioneering work produced the first gesture recognition algorithms, trackers, and the first applications exploiting them. In those years, the research community proposed several solutions that became the cornerstone of today's device operation. Their design focused mainly on hand input. For instance, Zimmerman et al. proposed DataGlove [259] a lightweight cotton glove allowing users to manipulate three or two-dimensional objects. It was composed of flex sensors, which measure finger bending, positioning, and orientation, including tactile feedback vibrators at the same time. In [117], Kramer and Leifer introduced Talking Glove, a useful communication aid for the deaf and the deaf-blind, supporting the verbal interaction with others. More specifically, this device is bi-directional: it analyses hand inputs to recognise and synthesize the spelt words and shows the incoming speech on the miniature LCD screen. Another example is the ALIVE [144] system, which supports full-body interaction between the user and a virtual environment.

It is worth pointing out that these approaches, in combination with the new technologies, have led the way to contemporary consumer-level tracking user's movements devices. In contrast to earlier systems, current devices are more precise, pervasive and tiny; they support the adoption of gestural interaction in different scenarios e.g., exhibitions, museums, and public spaces and, above all, in the entertainment field. Some examples are touchscreens, Nintendo Wiimote, Microsoft Kinect (version 1 and 2), Oculus Touch etc. On the one hand, such availability produced a set of de-facto standard gestures, for instance, the pinch and the swipe. On

Figure 1.1: The virtual world in Alive is composited of image in real-time of user with computer graphics [144]. The figure shows a virtual dog which mimics user's gesture.

the other hand, complex applications relying on full body gestures, touch interaction and pen strokes require a broader vocabulary, which must be "revealed" to the user during the interaction [174].

Substantially, the effectiveness of gestural interaction relies on how it supports the communication between the user and the application. In such systems, this property depends on two components. The first is an algorithm for recognizing user movements accurately. The second is the user's awareness of which movements are available for communicating with the system.

Figure 1.2: The use of gestures in augmented and virtual reality can make the experience more realistic.

Research in computer vision and machine learning focused mainly on the first problem, that is finding gesture tracking and recognition algorithms that are robust to noise in the input signal. The recognition of dynamic gestures (as opposed to static ones, that do not include a temporal dimension) has been addressed using machine learning techniques that explicitly consider the temporal dimension, like Hidden Markov Models (HMM), Dynamic Time Warping (DTW), Time-Delay Neural Networks (TDNN) and Finite-State Machines (FTM) [46, 156, 190], as well as traditional supervised classification algorithms like support vector machines (although they are more suited to static gestures [156]). All these approaches reached a very high accuracy in recognizing different gestures, and typically require the user to complete the entire gesture for recognizing it.

Such assumption does not match with the requirements set by the research on the second problem, that is making the user aware of which gestures are available for communicating with the application. In summary, the interface should provide two pieces of information to the user during the gestural interaction: *feedback* and *feedforward* [233], which may be designed taking different options into account [56]. The former (feedback) informs the user about the effects of the actions she has already performed. The latter (feedforward) provides information prior to any action, i.e. showing or anticipating the possible future actions. Figure 1.3 shows a sample visualization for such information considering a stroke gesture on a touch panel. The visualization both informs the user on her previous actions and guides her in concluding the interaction correctly.

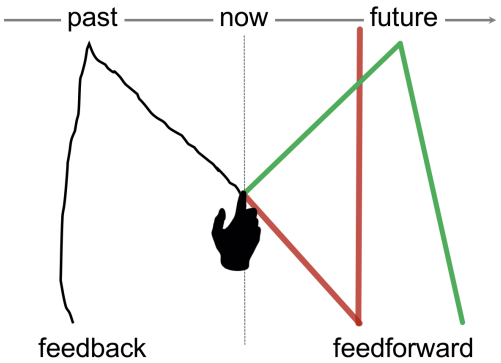


Figure 1.3: A simple guidance system for stroke gestures for a touch interface. It shows the previous touch positions with a black line (feedback) and the possible completions (feedforward). In this example, the system understands either an N stroke (in red) or M (in green).

In order to design and implement such guidance, the developer needs to establish which portion of a gesture has been completed, together with information on the expected conclusions of the movement, which may be more than one. The solutions for having a precise recognition and a usable interface diverge exactly at this point. On the one hand, the classification approaches require that the user completes the gesture for recognizing it, providing a very good accuracy. Since a gesture spans over a perceivable amount of time for the user, the interface (UI) often requires to provide guidance *during* such time, and this is not supported by classification algorithms. On the other hand, different engineering approaches in the literature model gestures through composition and/or declaration [110, 113, 114, 199, 200, 215, 216]. They allow receiving events for the recognition of a whole gesture and its sub-parts, facilitating the reuse of code. These approaches support effectively the development of guidance systems in a UI. However, the sub-part identification is currently achieved through geometric heuristics, which reach a much lower recognition accuracy if compared with the classification techniques in the state of the art. This may represent a treat for the UI overall usability when the gesture types and/or the context of use require a precise recognition.

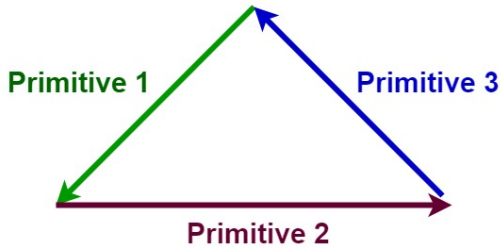


Figure 1.4: Through declarative methods, we can define the triangle stroke as sequence of three tilted lines where each line is a primitive.

In this thesis, we propose two methods for filling the gap between these two fields of research. Both methods accurately recognize gestures relying on HMM classifiers and compositional models, which represent gestures as a sequence of sub-movements described through expressions: the first method maps the composite HMM states to the underlying primitives which constitute it, the other method translates the input stream into a string representing the (partial) gesture and aligns it with a set of reference profiles using a gene-alignment technique. For each sample of a gesture sequence they identify the most likely list of primitives that the user already performed (useful for providing *feedback* in Figure 1.3) and the most likely primitive that the user will perform in the future).

We evaluated the proposed approaches with developers in coding the well-known Octopocus [14] interface (which requires both feedback and feedforward) against a representative of the many gesture recognition libraries that provide only the final label. The results show that our approach required less development time and cognitive load. The gain provided by having the support for the sub-part identification had exceeded the cost of modelling the gesture. The thesis is organized as follows:

- The chapter 2 discusses the related work about classification methods and compositional approaches for gesture recognition and interaction. Besides, the same chapter summarises the properties characterising feedback and feedforward and analyses the different guidance systems for gestural input.
- The chapter 3 describes the concepts we leverage for creating the models and the recognition techniques proposed in this dissertation.
- The chapter 4 introduces the first proposed method, called DEICTIC (DEclaratIve and ComposiTional Input Classifier). This is a declarative and compositional approach for describing stroke gestures according to primitives (ground terms) and a set of compositional operators (iterative, sequence, parallel and choice). By composing HMMs recognizing ground terms through a specific topology for each considered operator, it reaches an accuracy comparable with the state-of-the-art approaches in gesture recognition while supporting sub-gesture identification. The main limitation is the use of global features, which limits the application of DEICTIC in a general case for online recognition, which would require a prior knowledge of the gesture bounding box.
- The chapter 5 discusses the second proposed method, called G-Gene (Gesture-Gene). It addresses the limitations that characterize DEICTIC by introducing another bridge between a simple gesture modelling technique, engineered for supporting feedback and feedforward systems through the gesture sub-parts definition, and the construction of a Hidden Markov Model (HMM) classifier that i) has a reasonable accuracy, ii) supports online recognition and iii) identifies the sub-parts while the user performs the gesture.
- The chapter 6 analyses the results of a developer evaluation of the support provided by DEICTIC and G-Gene in different coding tasks. They consist of defining the gestural interaction support for a simplified drawing application, which exploits three simple stroke gestures (rectangle, triangle and delete). We studied the strategy participants follow, on the one hand, to implement a feedback and a feedforward system when intermediate gesture recognition information is not provided directly by the recognition support and, on the other hand,

to evaluate the information provided by our methods.
- The chapter 7 summarizes the results obtained and describes possible directions for further research.

The following list contains the peer-reviewed publications that are connected to the methods presented in this dissertation:

1. Alessandro Carcangiu. 2017. Gesture Recognition through Declarative and Classifier Approach. In Proceedings of the 22nd International Conference on Intelligent User Interfaces Companion (IUI '17 Companion). ACM, 185-188;

2. Alessandro Carcangiu, Lucio Davide Spano, Giorgio Fumera and Fabio Roli. Gesture modelling and recognition by integrating declarative models and pattern recognition algorithms. In Proceedings of the International Conference on Image Analysis and Processing (ICIAP 2017), Springer, pp 84-95;

3. Alessandro Carcangiu and Lucio Davide Spano. 2018. G-Gene: A Gene Alignment Method for Online Partial Stroke Gestures Recognition. Proc. ACM Hum.-Comput. Interact. 2, EICS, Article 13

4. Matteo Serpi, Alessandro Carcangiu, Alessio Murru, and Lucio Davide Spano. 2018. Web5VR: A Flexible Framework for Integrating Virtual Reality Input and Output Devices on the Web. Proc. ACM Hum.-Comput. Interact. 2, EICS, Article 4;

5. Alessandro Carcangiu, Lucio Davide Spano, Giorgio Fumera and Fabio Roli. 2019. DEICTIC: A compositional and declarative gesture description based on hidden markov models, Int. J. of Hum.-Comput. Stud., (122), 113-132.

Chapter 2

Related Works

Over the last years, the efforts of the research community focused on two main types of development problems related to gesture interface. The first is how to recognize gestures with a high accuracy, through algorithms robust to user's input variability. The second problem is how to describe gestures, using an effective and understandable representation for UI developers. Such description is usually linked to the support for defining in particular feedback and feedforward. In this chapter, we summarise the existing solutions in the literature for both problems.

The recognition problem is usually approached using pattern recognition and machine learning algorithms. As summarised in [48], they provide developers with high recognition accuracy and robustness to noise, allowing to spot complex gestures. Nevertheless, they typically expect the entire movement for recognizing it and these methods need many examples to succeed in the learning phase. Instead, research on gesture description resulted in different declarative and compositional techniques. On the one hand, they reduce the workload in the development of gestural User Interfaces, supporting the definition of feedback and feedforward with intermediate events between the gesture start and end. On the other hand, they reach a poor accuracy level, since they use geometric heuristics for segmenting gestures into sub-parts. In general, these approaches focused on formalizing gesture definitions and their connection to the application behaviour, paying less attention to the accuracy level. Different approaches in the literature [113, 114, 200, 215, 216] are used to describe gestural input through expressions decomposing them in smaller parts, also called primitives. A primitive represents a basic movement which may be part of one or more gestures.

The primary goal of our work is filling the gap between these two fields of research, defining a model which combines the usability and high level of declarative methods with the accuracy recognition and robustness of classification techniques. In the following, we overview the main proposed approaches for recognizing static and dynamic gestures, pointing out those able to identify gesture sub-parts and considering that the definition of gesture sub-part varies in different research field. We introduce the most relevant classification approaches in section 2.1; after that, in section 2.2 we detail the main approaches based on declarative models. Finally, in section 2.3 we analyse the approaches which support developers in building gestural user interfaces.

2.1 Classification Approaches

Vision-based algorithms have proved to be efficient in classification and recognition of gestures resulting the most used method in tracker devices and gestural interfaces. Computer Vision researches propose different tools for gesture recognition, based on the approaches ranging from statistical models, pattern recognition, image processing, connectionist systems, etc. [156]. In general, classifiers can be divided into two groups according to their properties:

Template matching methods , which align the input sequence with a reference one (template).

Machine learning algorithms , which learn to identify similar patterns by training and consist of both structures and parameters. They can be subdivided into two classes, supervised or unsupervised, depending on how the training phase is performed [206, 211].

It is worth pointing out that these methods may be combined with the aim of improving recognition accuracy. Indeed, each technique has benefits and drawbacks, and its accuracy is affected by several factors, e.g., context and scope, user target and recognizable movements. Usually, such solutions consist of two components, the first one has to represent the data received from tracker describing the gestures, the other one has to classify them. The task of the first component is to extract the features from both the training dataset and the user input, which are both acquired through different devices (cameras, infrared or depth sensors, gyroscopes, accelerometers, etc.). The features we can extract from the input stream depend on the input device we select. The task of the second component is to use these features for distinguishing gestures during the classification. In the following, we analyse in more detail the different

solutions proposed to detect gestures and which belong to both classes. We start from the template matching approaches, then we summarise the supervised approaches and finally we describe some unsupervised approaches applied to gesture recognition.

2.1.1 Template Matching Approaches

In this section, we describe the approaches based on template matching, a method originally applied in image processing for finding small components of a figure that match a template image. This concept has been also extended to pattern recognition, where it is used for determining the distance, understood as similarity, between a target template with a predefined template. Substantially, according to the component used in the matching, we may match images, features (e.g. different properties) or areas (e.g. size), expressed as matrices. Moreover, template matching algorithms use a distance metric, such as euclidean distance, Levenshtein distance or Mahanolobis distance, for measuring the similarity between two templates.

As an example, we consider spotting 10 unspecified gestures through a generic template matching classifier. In the beginning, the classifier extracts predefined templates from training samples, for each recognizable class. At run-time, user's movements are converted into a matrix and the model measures the similarity between this latter one with every predefined template, returning the one class with the lower distance from the input template. On the one hand, these algorithms typically need few training samples, on the other hand, they put a heavy workload on the CPU.

Dynamic Time Warping

Dynamic Time Warping (DTW) is an algorithm employed to compute the distance between two temporal sequences which may diversify in speed or length. According to Myers [167], it consists of three elements:

- a distance metric, utilized to represent the similarity between the sequences;
- the local and the global continuity constraints, both used for determining the warping contour;
- an endpoint constraint used to mark both start and and point of the two sequences;

Differently from other methods, in DTW the alignment is performed with different restrictions that limit it [164]. This algorithm is widely employed, particularly in speech and signature recognition systems. In the last few years, it has also successfully applied in gesture recognition. In this section,

we describe some methods proposed to recognized gestures using the DTW algorithm.

In 2003, Zhai and Kristensson presented SHARK, a shorthand-aided rapid keyboarding system, [121, 255], employing a scale and location independent method for digital handwriting recognition. Its main design goal was improving speed-writing on stylus keyboard by shorthand gestures. They represented a word as a sequence of continuous movements, with a start and end point, performed without raising either the finger or the stylus from the keyboard; in other words, every one of these terms is a 2D unistroke, describing the path by which its all letter are connected sequentially. SHARK employed a single dimension DTW, the classic elastic matching algorithm [222, 223], for recognizing shorthand gestures, through the alignment of the user input with every defined word's prototype and returning the one with the minimum distance. A few years later, Holt et al. [225] suggested an extended version of DTW for multi-dimensional time series (MD-DTW) in gesture recognition. Essentially, in that method, the best alignment is established using all the available dimensions. The evaluation tests were performed on a Dutch sign language dataset, showing, on the one hand, a higher robustness to noise and a better performance compared to original DTW. On the other hand, it required a higher computational cost.

Jackknife [224] is a general dynamic gesture recognizer. More precisely, it is a cross-device tool designed to achieve high recognition accuracy on continuous data, requiring few training samples and working with different modalities. In Jackknife, gestures are detected by combining dynamic time warping and a user-independent trajectories representation. The performance evaluation of Jackknife carried out on different input modalities and continuous data using: the $\$1 - GDS$ pen and touch dataset [241], a Wii remote dataset recorded by Cheema et al. [43], the Ellis et al. Kinect dataset [68] and two dataset for acoustic gestures and continuous movements collected by the authors. The experimental results showed, on the one hand, that Jackknife is able to reach an accuracy rate greater than 90% employing only two template samples for each gesture. On the other hand, they show that it cannot be applied to both static poses and gestures independent from the direction. For instance, a circle drawn counterclockwise or clockwise represent two distinct gestures. Such recognizer inspired the work by Pittman and LaViola in [184], where they detected complex hand gestures using Jackknife recognizer and multiwaves.

Choi et al. in [47] proposed a method for reducing the computational

cost in dynamic time warping algorithms. In that system, features and candidate templates are selected by using an orientation histogram, achieving an average accuracy of 97.4%. Zhao et al. [258] relied on Kinect data to detect dynamic hand gestures. The recognition uses DTW, combining skeleton data and hand contour features, reaching an accuracy of 94%. Ruan and Tian [194] introduced an improved DTW algorithm to detect 3D dynamic hand gestures, acquiring movements through a Kinect. The proposed approach represents the user input and gestures with the method of weighted distance, starting from the received skeleton data. Template matching and Kinect data were also considered by Hang et al. [85]. They recognize fingertip unistrokes gestures introducing an improved DTW algorithm that relaxes the endpoint constraint. This solution and the selection of candidates templates using the LB_Keogh lower bound function [107] allow reducing the computational complexity jointly. The representation of fingertip trajectories is based on the vector quantization coding for angle values described in [220]. Another approach focusing on Kinect sensor data was proposed by Doliotis et al. [63], combining hand shapes and DTW algorithm for recognizing dynamic hand gestures.

In literature, we can find several other works based on the dynamic time warping algorithm, designed for hand gesture interaction or action detection systems in different contexts, for instance [74,95] for sign language recognition, in wearable devices [93, 119, 152], in health systems [16, 159] or in-car interaction [69].

Kalman Filter

A Kalman filter [101] is a recursive solution for the discrete-data filtering problem. In his work, Kalman provides a powerful tool which minimizes the mean of the squared error in estimating the state of a process [24]. It offers a valid support in estimating past, present, and future states independently from the nature of the modelled system. In gesture recognition, the Kalman filter is widely used for both preprocessing data, i.e., to reduce noise in tracked trajectories, and predicting the next user's movements.

For example, Allharbi et al. [2] proposed an extended Kalman filter for pre-processing data received from Kinect and applied to a TaiChi rehabilitation system. The Kalman filter is employed to fix incomplete and inconsistent kinematic sensory data, and Allharbi evaluated its performance through different classifiers achieving an accuracy above 97.5%. In [157] Mo et al. introduced a method for segmenting hand gestures based on an improved Kalman filter and TSL skin colour model. The TSL model is a feature where a colour is specified as a combination of three components: i) tint, ii) saturation and iii) luminance.

2.1.2 The "Dollar" Family

Another recognition approach is the so-called Dollar-family, a set of methods based on 2D template matching, which usually requires fewer training samples, but requires the whole input trajectory for the recognition. They compare distances between corresponding locations of resampled trajectories to measure the gesture differences. The original approach was the 1$ recognizer, designed for touch/stylus applications and introduced by Wobbrock et al. [241]. This algorithm compares 2D gesture paths, which are represented as a temporal sequence of points. More precisely, a single gesture path is extracted considering the full user input. Thus, the $1 recognizer can spot only unistroke gestures. Gesture recognition is accomplished through the following steps:

- The whole user input is resampled in M points, allowing to compare gesture paths with different movement speeds;
- paths are rotated over the space of possible angles, in order to find the best alignment between their points;
- the rotated paths are scaled to a reference square and translated in order to have the centre in the origin;
- finally, the algorithm searches the output class with the best score, using the Euclidean metric to measure the distances.

Therefore, this method does not depend on the execution velocity, 180° rotation, scale, and translation. Wobbrock et al. evaluated the performance of their method by comparing 1$ with a classic DTW and the Rubine's algorithm, using a dataset with 4800 samples for 16 gestures. In their experimental tests, 1$ achieved an accuracy of 99%, greater than Rubine algorithm and similarly to DTW algorithm, lowering the processing time.

Through the years, this approach has been improved with various modifications to the processing steps and resulting in different popular solution for implementing 2D stroke recognizers. For instance, Li [131] introduced the Protactor classifier, which is a template-based recognizer focused on single-stroke gestures. In that classifier, the best path alignment is found by calculating the optimal angular distances. The evaluation shows that Protactor outperforms 1$ on accuracy, time and space cost. In the same year, Anthony and Wobbrock proposed the N$-recognizer [7, 8] which extends the 1$ for recognizing 2D multistroke gestures. This method models a multistroke gesture as a set of unistrokes, without assuming any ordering among them. The N$ recognition is based on bounded rotation invariance, supporting multistrokes performing in any order and direction: substantially, the template matching is implemented comparing every multistroke's component permutation with the pre-processed tracked trajectories, thus making it invariant to the execution direction. Multistroke

recognition is also studied in [230, 231]. In these works, Vatavu et al. presented the P$ recognizer. It aims to extend the N$ recognizing to both unistroke and multistroke symbols by representing gestures as a cloud of points. In particular, Vatavu et al. find the lowest distance in a set of point clouds by coupling the Hungarian algorithm within the Nearest-Neighbor (NN) classifier. The recognition is expressed as an alignment problem between two bipartite graphs, that can be resolved by using the Hungarian algorithm. Instead, the best alignment is found through the NN classifier. The evaluation tests showed that P$ offers an accuracy of around 99% with a few training samples. Recently, Vatavu et al. enlarged the $ family recognizers by presenting Q$ [232]. In contrast to previous works, it is also designed for wearable and embedded devices, extending and improving the P$ recognizer. Compared to the latter one, the classification process has been optimized, allowing to reduce the hardware workload and processing time.

It is worth pointing out that none of the mentioned "dollar" approaches was designed for mid-air gestures recognition. Recently, Caputo et al. [36, 37] introduced the 3Cent recognizer, an improved version of "dollar" method for recognizing 3D dynamic hand gestures. Shape matching techniques are combined with the tracked skeleton data, namely the orientation evolution and the fingers description features. This allows detecting gestures with an accuracy of 90% on two recent benchmarks. More precisely, the hand orientation is characterized by the trajectories of a few unit vectors defining it. The authors considered a few key points (index, thumb bases, tips, wrist, and palm), which are assumed to characterize the gestures of interest and estimated all the unit vectors of the directions joining them. Finally, they extracted the finger distances according to the angle related to their movements.

2.1.3 Supervised Algorithms

A classification technique applies a supervised model if it is trained through ground-truth labelled samples. In other words, the algorithm learns to associate each example contained in the training dataset to an output value (a specific class), creating and adjusting an inferred function through which the classifier maps new examples. Supervised learning can be of two types, depending on the output required by the modelled system: i) classification, when the output is a category, otherwise ii) regression, if the output is a real value.

Supervised classifiers offer different benefits. On the one hand, we know

the available output classes, therefore their functioning can be evaluated and analysed, similarly to the template matching methods. On the other hand, it is worth pointing out that the mapping is determined from the training dataset, consequently, noisy examples can undermine the performance of these models. As an example, we can consider recognizing 10 gestures through a generic supervised classifier. At first, the classifier is trained to recognise these 10 classes; its inner parameters are adjusted using the labelled samples included in the training dataset, which balance the tracked feature values. At run-time, the set of features is extracted from the new gesture samples, which represents the tracked user's movements and they are sent to the classifier. The trained model will return one of those ten classes as output or, more frequently, the probability of belonging to each of the classes, according to the received input and its inner structure.

Such techniques usually guarantee a high accuracy and robustness to noise. Therefore, supervised learning classifiers are widely employed in many pattern recognition fields, for example, speech or action recognition, spam detection, information retrieval, and so on. A large part of machine learning classifiers has been addressed on supervised algorithms, like Hidden Markov Models (HMM), Neural Networks (NN), k-Nearest Neighbor (kNN), Random Forests (RF), etc.

Finite State Machine and Hidden Markov Model

In this section, we describe first Finite State Machines (FSMs) and then we discuss Hidden Markov Models (HMMs). The first ones are also called finite automata and, substantially, they are abstract machines with a finite number of states, a starting state, and a set of transitions [90]. A transition, normally, connects a pair of states s^i and s^j, describing the condition or action that allows moving from s^i to s^j. There are several typologies of FSM which differ according to the arrangement of transitions; i.e., in the ergodic model, all states are connected to each other, in the left-right model we can move forward over the states that compose the machine and so on. Such methods are efficient to describe discrete and sequential systems in which the current state depends from previous states.

Hidden Markov models represent an extension of FSMs. They are also known as Markov sources or probabilistic functions of Markov chains. They are probabilistic models, including a distribution generated for the possible observations that depend only on the current state of the Markov process. In addition, they include a transition probability between states, but the current one is not observable in such processes (which explains the

adjective hidden). Researchers employ HMMs for the statistical modelling of non-stationary signals, i.e., speech recognition or image sequences, or to represent and analyse discrete and continuous series [67, 187, 260]. We considered such structures in our work considering their flexibility in structure (we manipulate their topology for the composition) and the existence of well-known algorithms for training, recognition and state likelihood evaluation. A deeper description of HMMs is included in the section 3.2.

The first researchers introducing HMMs in gesture and action recognition were Yamato et al. [245]. They created and trained one HMM for each available action category, converting time-sequential images in a sequence of symbols. Each model processes these symbols, returning the likelihood that the tracked movements belong to the corresponding class.

Starner and Pentland [217] applied HMMs to create a system for recognizing sentence-level American Sign Language (ASL) in real-time. In their work, static hand poses are tracked both from solidly coloured gloves or natural skin tone using a camera. They are represented by an eight-element feature vector consisting of x and y coordinates, axis angle of the least inertia and eccentricity of bounding ellipse. Moreover, they applied a statistical grammar model to enforce the training and the recognition, achieving an average recognition rate of 91.9%. Static hand gestures recognition and HMMs are also studied in [197] where Sagayam and Hemanth applied Artificial Bee Colony (ABC) algorithm to optimize HMM's performance: ABC is an optimization algorithm which simulates foraging behaviour of honey bee swarm proposed by Karaboga [104]. In the evaluation tests, Sagayam and Hemanth reported an average recognition rate of 73.59%.

In the literature, we find several works in which HMMs are used for recognizing dynamic gestures. Lee and Kim [128] proposed an ergodic threshold model for an HMM that calculates the likelihood of an input pattern and indicates the matched gesture. In this work, each recognizable gesture is detected through a simple left-right HMM, representing the user input by temporal characteristics. In the experimental tests, Lee and Kim noticed an overall recognition rate of 98.1%. Sezgin and Davis [205] represented sketches as a sequence of basic strokes which are recognized by applying an approach similar to the ones we present in this dissertation. On the one hand, strokes are recognized relying on Hidden Markov models; on the other hand, compared to our approaches, this method does not: i) avoid the collection of new samples for each available sketch; ii) generate HMM as composition of more basic HMMs. User study proves that increasing

the number of strokes the used method reduces the computational cost of this solution, contrary to conventional methods. Min et al. [155] codified user hand input coding eight discrete directions through characters and representing gestures as string chaining them. They used HMM to recognize twelve 2D strokes acknowledging an average recognition rate above 90%, considering complete gestures. We use a similar approach for online recognition, i.e. iteratively recognizing the most likely gesture *during* the performance. Marcel et al. [147] dealt with hand gestures detection using Input-Output Markov chains, which combine neural networks and HMMs. Differently to the other Markov sources, in this model every state is associated with an input and output state neural network. Such solution relies on HMMs training properties with the discrimination efficiency offered by Neural Networks. The features applied to IOHMM represent the skin-colour of the user's hand and face, extracted from a sequence of video images. In contrast to the previous methods, which are designed to track one hand, Kao and Fahn [103] focused on two hand gestures. They defined five basic strokes, to be performed by any hand, described as a sequence of centroid points: moving upward, downward, leftward, rightward and no action; the resulting twenty-four possible combination of gestures are recognized through HMMs, in which the average recognition rate is more than 96%. The GestureCommander, proposed by Lucchese et al. [140], is a touch-based method for mobile devices, designed to predict gestures and to show feedback. It consists of two connected HMM systems, one for each task. When the user is performing a gesture, the underlying framework generates visual feedbacks according to the three most likely predicted gestures with an average accuracy of 96.5%. The input movements are described using a subset of the Rubine features, namely direction, curvature and sine of the initial angle (all of them can be extracted without knowing the whole input movement).

Deo et al. [60] detected hand gestures by coupling convolutional networks and Markov sources. This method, that is designed for in-vehicle gestural interfaces, employs both a CNN to extract features from videos and an HMM to label the tracked hand movements. Deo et al. evaluated their system using the VIVA dataset [71], and achieved an accuracy rate of 77.5%.

HMMs found an application also in wearable trackers considering their flexibility, e.g., in [108, 118, 183]. Keskin et al. in [108] proposed a system able to capture and recognize real-time 3D hand strokes, performed through coloured gloves. This system used left-right HMMs to model gestures, and Baum-Welch algorithm to re-estimate its parameters and deal with segmentation ambiguity. It reached an accuracy above the

98%. Kratz et al. [118] combined Nintendo Wii controller with HMMs for recognizing 3D accelerometer gestures, while Park et al. [183] implemented an energy-efficient gesture recognition system for wearable devices based on multi-situation HMM models, an extension of HMM able to fit multiple mobility situations.

Support Vector Machines

In this section, we describe the Support Vector Machine (SVM), a non-probabilistic supervised learning algorithm widely employed for classification and regression analysis. The standard version of SVM [229] allows binary linear matching, assigning new examples to either one or the other category: it defines a hyperplane, in high or infinite dimensional space, to map examples as points in space which are subdivided by a clear gap that is as wide as possible [218, 226]. Over the years, researchers have proposed different methods to make SVM applicable in multi-class classification problems, i.e., one-vs-all, Directed Acyclic Graph (DAG) [185], error-correcting output codes (ECOC) [62] etc. [73].

SVM-based techniques proved their accuracy in recognizing dynamic or static gestures. Aoki et al. [9] employed SVM to recognize real-time strokes for vision-based gesture interfaces (VGI), such as TV menus or similar. Usually, VGIs recognize both the shape of stroke and the orientation based on the start position allowing to decrease the number of strokes the users need to remember. They achieved unistroke recognition combining SVM and Distance to Border (DtB) features; the latter ones represent the distance from the edge of the figure to the figure's bounding box. The accuracy of the model proposed by Aoki et al. is ranges between 84.57% and 99.43%. Yuan and Barner [253] showed an algorithm to recognize 9 two-dimensional strokes: it combines PCA and non-linear SVM using shape and motion figures as features. This work proposed to extend both multi-class and binary SVM with a hybrid kernel, where the Euclidean distance is replaced by the weighted sum of the shape distance and dynamic time warping to measure the similarity between sequences. Yuan and Barner evaluated the proposed method in three different cases on 9 different gestures, achieving an accuracy above 90%. Instead, in [239], SVMs were applied for an offline recognition of 2D strokes. In their work, Willems et al. proposed eight different configurations of features based on stroke shape, such as length, the angle of the bounding box, the area and so on. They compared the performance of the different subsets against global features using SVM, multi-layered perceptron (MLP) and DTW. The result was that the combination of shape features and SVM strongly improves recognition performances compared to global features.

Gosh and Ari [77] aimed at recognizing hand poses in sign language applications by using multi-class SVM. The method is robust to illumination, rotation and position variations, and it represents hand images as a combination of located contour sequences (LCS) and block-based features. Three different datasets are employed in experimental tests, with an accuracy above the 95%. Another work focused on supporting sign languages was proposed by Kumar et al. [123]. It recognizes two main components of sign languages simultaneously, manual signs and finger-spelling. These components represent, respectively, gestures and words, both consisting of hand or fingers movements. Kumar et al. proposed a framework exploiting SVMs for distinguishing between manual signs or fingers-spelling, Bidirectional Long Short-Term Memory Neural Networks (BLSTM-NN) classifiers to recognize the distinguished gestures (one for each component) and the Leap Motion device for tracking the user's movements.

Various Kinect-based methods employ SVMs to spot hand poses. Dominio et al. [64, 65] achieved hand static gestures recognition introducing two new features based on depth data acquired from Kinect. First of all, they located the user's hand and its components, namely the fingers and the palm, from the depth and colour data. These regions were located using PCA and a circle fitting on the palm. Successively, this model computes the finger distance from the hand and the curvature of the hand shape. These features are the input of a multi-class SVM, recognizing 10 hand poses in real time. In the experimental tests, the underlying system achieved a recognition rate ranged between 95% and 100%. In [148], they extended the method supporting Leap Motion data. Keskin et al. [109], recognized hand poses using a random decision forest (RDF) to segment hand depth map into its different parts, and an SVM to classify the segmented hand parts. That system was evaluated on the America Sign Language digit dataset, and it achieved an accuracy of 99.96. Okada and Otsuka [180] proposed a method for associating gesture with words. The work proposed detects and associates gestures relying on an SVM, representing the user's input through three set of features: trajectory signal feature set, hand motion data and primitive gesture pattern phases. In the experimental tests, they compared these three sets of features, obtaining the best accuracy (63.4%) with the gesture phase features.

Moreover, CV and HCI communities apply SVM recognizers in both action and expression classification. Among these works, some are designed for tangible acoustic interfaces (TAIs). More precisely, TAI is a particular type of tactile interface, where the interaction between human and computer is carried out through various acoustic or imaging sensing

technologies. In other words, user's gestures are analysed and localised by exploiting the propagation of sounds and video frames.

Camurri et al. [31] analysed expressive gestures in TAIs studying the interaction between human and a Google Earth application on tangible acoustic interfaces. They implemented an SVM-based expression hand gesture detector for a TAI, combining acoustic and visual data. This system relies on the EyesWeb Expressive Processing Library [33], a flexible platform for supporting multimodal and cross-modal processing, used for both segmenting hand movements from a tactile surface and extracting expressive features from the acquired trajectory (namely time duration, peak velocity, impulsiveness, spatial length and directness index). Expressions gestures are recognized combining two SVMs and the Laban's theory. Substantially, according to this theory, there is a relationship among humans actions, space and time that can be represented through different binary components. In their work, Camurri et al. focused on two dimensions: Space, where the motion can be either flexible or direct, and Time, the action is quick or sustained. The gestures and the interaction are classified according to these dimensions, using a single SVM with likelihood estimation for each component.

Among the action recognition works, Ji et al. [97], recognised offline cooking gestures coupling visual local features, depth image information, and a multi-class SVM. User actions are tracked through the Kinect and local features are estimated using an extended FAST detector and the compact histogram of orientation gradient in spatiotemporal space (CHOG3D). Such descriptors are employed to detect and describe feature points in three dimensions.

On the contrary, Schuldt et al. [201] used SVM in action recognition exploiting local events in videos, such as size, frequency and velocity of motion patterns, as features. SVM is employed also for low-power classifiers in wearable devices, resulting suitable for working with different kind of data: for instance for recognizing real-time gestures through SVMs and electromyography (EMG) [18], or detecting hand gestures through a first-person camera [204], exploiting a random forest-based hand segmentation algorithm to track user movements.

Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models that vaguely reproduce working principles of biological neural networks. McCulloch and Pitts formalized them in [150] as a conceptual model able to perform computational tasks in the same way as a human brain. The main components of the ANN are the neurons. They are equipped with input and output channels: when a signal is sent, they change their state according

to their activation function applied to the received data and produce an output depending on the input and state. Therefore, a stereotypical ANN is an interconnected system which consists of neurons layers and weighted edges. The first ones can be distinguished into the input, hidden and the output layer, while the second ones join neurons to each other conditioning the final output [17, 23, 83, 192]. The ANNs represent another approach based on supervised learning algorithm, where weights and activations are learned during the training phase.

In the literature, we find several variants of ANNs, which differ for structure, activation function or learning rule. ANNs and their derivative work, such as Convolutional Neural Network (CNN), Deep Neural Network (DNN) and Recurrent Neural Network (RNN), are widely used in pattern recognition and in this section we analyse some of these methods employed in gesture recognition field.

Recurrent Neural Network A Recurrent Neural Network, or recurrent net, is a particular class of artificial neural networks, specialised in learning sequential or time-varying patterns [145]. Formally, it is a neural network with feedback connections, namely the edges between nodes describe a direct graph along a sequence. The arrangement of connections in recurrent nets affects their properties. The described graph can be either cyclic or acyclic, corresponding respectively to infinite and finite impulse networks [151]. The first ones are organized as a fully interconnected net, allowing to get back in the start state. The other ones consist of a partially connected net called unrolled graph, where only forward steps are supported (nodes have no backwards arcs). The engineering community proposed several structures suitable to different problems, such as self-gated recurrent nets (SGRNN), Elman's network, Jordan's Network (JRNN), long-short-term memory network (LSTMN) etc.

Similarly to convolutional nets, recurrent networks have found a wide use in all pattern recognition applications, in particular, to detect and recognize user movements and emotions. The state-of-the-art offers various RNN-based approaches focused on gesture recognition. It is worth pointing out that most of them are focused on either static or dynamic hand gestures. In 1991, Murakami and Taguchi were the firsts to apply Elman's recurrent net in sign dynamic hand gestures recognition [166]. Their method classifies sign language movements passing to RNN the angle and the positions of fingers and hands, acquired by DataGlove. About at the same time, Cracknell et al. evaluated the performance of three different techniques, including RNNs, with dynamic 3D gestures [49]. RNN was also exploited by Muraqa and Abu-Zaiter in [146] for sign language: they compared

Elman's recurrent net and a fully connected RNN to detect hand postures. They used the position of the five fingertips and the wrist extracted from colour images as features, achieving a recognition rate of 95.11% with the second proposed method.

Two streams recurrent networks (2S-RNN) are used by Chai et al. in their framework [42] to detect gestures in a continuous input stream. Firstly, input images are segmented into separated gestures, locating the hand by exploiting a fast-CNN detector, and using a particular movement, the user put hands down, to mark the end of a gesture from the beginning of the next gesture. Successively, the framework uses the 2 streams networks for labelling segments with the predefined gesture detected. Neverova et al. [171] used hand RGB-D images, skeletal motion data, and an audio stream to detect gestures. The features are extracted using multilayer perceptrons (MPL), for hands images and skeletal data, and a bag-of-word (BoW) for the speech input. In more detail, MLP is another kind of supervised feedforward neural networks based on multiple layers and non linear activations; multiple and single layer perceptrons represent the most basic architectures in artificial neural networks. Instead, a BoW is a vector of occurrence counts of words, namely a sparse histogram over the vocabulary, that can be applied to image or speech classification by treating their features as words. After that, these feature sequences are fed to a recurrent net for the classification. Chen et al. in [45] employed skeleton data to detect hand gestures: fingers and global motion features are extracted and sent, along with the skeleton sequence, to a recurrent network that analyses the received input. A low power system based on RNNs was introduced by Shin and Sung in [207]. They proposed two different solutions to achieve dynamic hand gestures classification with wearable devices: the first one combines convolutional networks and RNN working on the received video signals, the other one exploits the acquired accelerometer data through an RNN. In particular, each solution uses the fixed-point algorithm to optimize the input signals, reducing the memory storage and thus improving the system performance. There are several works which detect air-gestures combining wearable devices and recurrent nets. For instance, in [235] there is a decrease in the CPU load to detect human movements through wearable devices, employing a recurrent network. The accelerometer and gyroscope data is fed to a self-gated RNN to classify human action movements. In the experimental tests performed on the HTC [252] and the HAPT [191] datasets, the proposed system reaches an average accuracy rate (93.01%), comparable to state-of-art techniques but with lower memory requirement and lower computational cost. Xu and Xue [244] apply inertial data, collected

through smart-watches, to a long short-term memory (LSTM) in order to recognize dynamic arm actions, such as time watching or shoulder patting. LSTM network, introduced by Hochreiter et Schmidbhuber [88], is a special kind of RNN designed to address the long-term dependency problem in recurrent terms. Indeed, if on the one hand original RNNs are able to connect information to the present task, on the other hand the gap between these information must be small. Xu and Xue achieved a recognition rate ranged between 99% and 90%, depending on the considered class samples (faster or lower execution).

Some methods based on recurrent nets are designed for both mobile and wearable devices. Czuszynski et al. detected hand gestures combining linear optical data and RNN [51,52]. Li et al. proposed SonicOperator [130], a system for in-air hand gestures based on ultrasounds. In SonicOperator, the device's speaker and microphones are employed to generate and receive ultrasounds, while features are obtained from the Fast Fourier Transform, which is applied to reflected waves, and sent to RNN for recognizing hand movements.

Instead, Naguri and Bunesco [168] compared long short-term memory (LSTM) and CNN on 3D hand movements tracked by Leap Motion. The system presented by Naguri and Bunescu is divided into two modules:

- The gesture detector identifies the subsequences of frames that correspond to one of the predefined gesture. They implemented it as an LSTM operating on post-processed input data;
- The gesture classifier that consists of a mean-pooling layer and a Softmax model. The former is connected to the output of a gesture detector and converts input segments in a fixed size input; then the fixed input is sent to Softmax model for classification.

In [254], Yuan et al. presented a set of command-like multi-touch gestures for users with limited physical capabilities: a recurrent net is employed for the recognition, which is applied to the trajectory and the angle features computed from measured touches. In their work, Yuan et al. reached an overall accuracy of 96.1% on 10 different gestures.

Convolutional Neural Network The convolutional neural network, also known as convolutional network, is another extended version of the ANN. They are designed to process data having a know grid-like topology, namely vectors in one or two dimensions, such as e.g., time-series or image data. Differently from the other artificial neuron networks, CNN applies convolution operation to the input in place of general matrix multiplication in least one of its layers. More precisely, the convolution is a specialized kind of linear operation which emulates the response of a neuron to visual

stimuli [80]. CNN is part of feedfoward artificial neural networks; it means that in CNNs the inner connections follow a feedforward disposition.

Recent researches introduced CNNs in the gesture recognition field. In particular, the vast majority of these CNN-based methods has been applied to either dynamic or static hand gestures. They are diversified according to both the features applied for representing input data and the devices used for tracking movements. The first work to explore CNN in gesture recognition was by Nowlan and Platt in [176]. Given a video sequence, they determine if the user's hand is closed or open in real-time employing CNNs: poses are spotted by locating the hand and using the last three received frames. The system proposed has an accuracy of 99.1%, demonstrating that convolutional networks can be used also in gesture recognition. Lin et al. [133] proposed a framework which accomplishes offline hand poses recognition without using hand features, such as fingertips or contour, for describing every gesture. Firstly, this system extracts the user's hand, from the received image, using a skin model trained by a Gaussian Mixture Model (GMM). Then, it calibrates the hand by its position and orientation. The calibrated images are the input of a CNN trained to recognize seven different hand poses, achieving an accuracy of 95.96%. Yingxin et al. [250] introduced an extended version of CNNs for hand static gestures. They modified the structure and data preprocessing methods used in CNN, improving its robustness with various illuminations.

Lamar et al. proposed a method of feature extraction based on PCA for hand postures and used it with a neural network system, called T-ComboNET [127]. Principal Component Analysis (PCA) was introduced in 1901 by Pearson and developed independently by Hotelling in 1933. It is a multivariate method which allows reducing the dimensionality of a dataset formed of a large number of interrelated variables while retaining as much as possible the variation present in the dataset [99]. In experimental tests, T-ComboNET achieved a recognition rate of 99.4% with 42 static postures and 34 hand motions. Moreover, PCA has demonstrated to be useful tools in feature extraction, several extensions of PCA convolving other techniques have been proposed over the years, like clustering methods or neural networks. For example, Shao-Zi et al. in [129] combined sparse auto-encoder (SAE) and PCA with Convolutional Neural Network (CNN) to define a learning feature approach for human action recognition. Park and Cho [182] utilized PCA and a neural network to recognize static gestures using impulse radio ultra-wideband (IR-UWB). In [22] Birk et al. proposed an approach which is able to recognize 25 international hand alphabet gestures in real-time; this technique extracts offline the

features from training images through PCA and uses them to train a Bayes classifier, which employs these latter ones to label the incoming images. The authors noticed an offline recognition rate such around 99%. PCA was also exploited by Martin and Crowley in [149] where they proposed a technique which allows the user to interact with a system by hand gestures and consists of three components: the first tracks the user's hand, the second classifies hand postures in a space defined by a PCA, while the last part determines the gesture in real-time, starting from the recognized poses. Aleotti et al. [1] acknowledged arm gestures using functional PCA (FPCA) for both unsupervised clustering of training data and gesture recognition. Such system is applied in a small humanoid robot to reproduce the recognized gestures.

Lu and Little in [139] track and recognize athlete's movements by applying firstly HOG descriptor to input and then projecting it in a linear subspace by using PCA. HOG is a feature descriptor that represents an image, or an image patch, using the distribution of directions of gradients, namely the derivative X and Y of an image. Such descriptor allows to highlight edges and corners, regions where the magnitude of gradients typically is large. Amin and Yan [4] employed PCA to reduce the feature space obtained from Gabor filters, performing gesture classification with a method of fuzzy-c-mean clustering. Similarly, [94] and [82] coupled PCA and Gabor filters to describe user input and classified static hand poses through support vector machine. Munoz-Salinas et al. in [165] introduced the concept of depth silhouette in gesture detection, using the principal component analysis to compress silhouettes and support vector machines to learn temporal patterns. In another work [243], principal component analysis was combined with the percentage of cumulative energy to analyse and spot both static and dynamic hand gestures using inertial data.

In 2017, Amir et al. [5] described a low-power, event-based and online hand gesture recognition system using a deep convolutional network. Gestural input was tracked through a Dynamic Visions Sensor (DVS), which sends events only when a pixel value changes its magnitude; these events are passed to the CNN hosted on TrueNorth, an asynchronous, lower-power, and neuromorphic processor. The peculiarity of this system is the high accuracy (96.46%), achieved with low latency and low power devices.

There are other works focused on hand gesture recognition with low power consumption. In [54], Dekker et al. combined CNN and frequency-modulated continuous-waves (FMCW) radar, representing the user's hand with spectrograms and supporting low power hand dynamic gesture recog-

nition for consumer devices. In a similar way, Zhang et al. [257] detected dynamic hand gestures relying on continuous waves and 3D convolutional networks. Compared to [55], the system proposed by Zhang et al., called Latern, involves a recurrent net in order to improve the performance. It is worth pointing out that FMCW signals are not altered from lighting, noises or atmospheric conditions contrary to other types of features or techniques.

Molchanov et al. studied the application of convolutional networks in drivers' hand gesture recognition. In 2015, they used colour and depth data in a 3D CNN classifier, consisting of two components: a high-resolution and a low-resolution network [160]. This method reached an accuracy of 77.5% on the VIVA challenge dataset [71]. In the next year, they proposed an extended version of the previous work, which employs recurrent 3D convolutional networks, embedding RGB-D data, infrared data and optical flow as features [161].

Several works analysed the use of CNN in detecting and classifying hand movements. In [61], Devineau et al. suggested a new convolutional network architecture for 3D dynamic hand gestures relying on hand skeleton data. In that case, the gestural input is received from an Intel RealSense camera, which describes the user's hand through 22 joints: it returns four joints for each finger, while the others represent the centre of palm and wrist respectively. The architecture proposed by Devineau et al. is a multi-channel CNN that processes in parallel these sequences of hand joints positions, classifying complete hand gestures. They evaluated the implemented system using the DHG dataset from the SHREC2017 - 3D Shape Retrieval Contest [153]; the recognition accuracy achieved is 91.28% in the 14 gesture classes case and 84.35% in the 28 gesture classes case. Kopiski et al. combined depth data extracted by Time-of-Flight (ToF) sensor and convolutional networks to detect mid-air hand gestures [116]. The suggested solution relies on a new feature generation technique which converts the point clouds extracted from depth data in a suitable size for CNN. Strezoski et al. [219] compared the performance of different types of CNN on hand gestures using the Marcel dataset [147].

In literature, we can find different works employing waveforms as features in gesture recognition applications. In the last years, researches have proposed approaches combining wireless signals and CNN. Kim et al. [112] tracked the user's hand movements using impulse-radio (IR) signals and a wireless sensor. This sensor consists of a transmitter and a receiver: the former generates the signals while the latter receives the waveform reflected from the hand. Kim et al. classified waveforms by

CNN, according to their amplitude and phase. In a recent work, Ma et al. [143] presented SignFi, a sign language recognition system based on wireless sensors. In this technique, hands input is obtained from WiFi packets and expressed as Channel State Informations (CSIs): CSI is utilized in wireless communication to represent the signal propagation quality, from the transmitter to the receiver, describing scattering, fading and so on. Once the CSI measurements are received, these are pre-processed with the aim of removing noise. Finally, SignFi submits them to a 9-layer CNN for the sign classification. Ma et al. evaluated the system in three different conditions, reporting an accuracy ranging between 94.81% and 98.01%. The application of convolutional networks in sign language was studied also by Rakowski and Wandzik [189], who assessed the performance of three state-of-the-art deep CNN architectures for recognizing hand signs representing alphabet letters and classifying 60 common signs. The classification tasks are performed respectively on the American sign language fingerspelling dataset [186], and the 1 Milion hands dataset [115]. Neverova et al. [171, 172] detected sign language gestures applying only RGB-D images and upper-body skeletal motion data to convolutional networks.

In 2013, Ji et al. in [96] applied CNNs in action recognition. This work proposed a three dimensional CNN for recognising 3D movements. The novel model is obtained using a 3D kernel, allowing to capture spatial and temporal features from adjacent frames. Ji et al. compared the performance of their method with the other state-of-the-art techniques, achieving an average accuracy of 90.2% with the KTH dataset [201].

Time-Delay Neural Network The Time Delay Neural Network is another specialised class of neural network, as its name suggests. Compared to NN, it classifies patterns with shift-invariance, avoiding the need of segmenting data during training. This type of net models the context at each layer of the network, receiving inputs from a contextual window of outputs from the previous layer. It was first designed for speech recognition [236] and it has found wide employment in other recognizing temporal patterns, including gesture recognition.

In 2001, Yang and Ahuja proposed a recognition hand gestures algorithm [247] based on trajectories, where the user's movements are extracted from video sequences in two phases and recognized by TDNN. In the first step, they used a multi-scale segmentation method for partitioning every frame image into regions; then, consecutive frames are compared, highlighting the regions with similar pixel correspondences. Moreover, these pixels are concatenated and grouped based on their 2-view motion

similarity from which the motion trajectories are extracted. Then, it relies on a time-delay network to recognize motion patterns from trajectories and regions, achieving an accuracy of 96.21% with 40 hand gestures from American Sign Language. The performance and the motion extraction were further enhanced by Yang, Aruja and Tabb in [248]. These are the first works where a TDNN is employed in recognizing gestures. Another work focusing on sign language recognition using time-delay networks is presented by Bhowmick et al. [20]. In this method, the gestural input is acquired by webcam, using a skin colour segmentation model to extract and track user's hand in the received frames; the authors described the segmented hand through a set of features that consists of orientation, gesture trajectory length, velocity, and acceleration. These features are then applied to an MLP and a TDNN to recognize, respectively, static and continuous gestures. In experimental tests, they achieved a recognition rate of 92.5% for static gesture and 87.14% for dynamic movements.

Modler and Myatt, in [158], proposed a gesture-based method for controlling sound parameters in an interactive music system. In their method, gestures are defined as a cyclic sequence of two poses, such as index moves up and down or flat hand moves up and down. The method obtains left-hand poses from images, extracting the features through 2D spatial Fourier Transformation and high-cut filtering to reduce and smooth data. Modler and Myatt employed a time-delay net to recognize the gestures. TDNNs are also used in low power gesture recognition solutions: Liu et al. presented Virtual Trackpad [137], an online wireless hand gesture recognition device based on electromyography signals for tracking hand movements. It is able to detect 10 common hand gestures, extracting the features from four differential EMG channels and applying them to time-delay neural network. The device generates an event when a gesture is recognized, achieving an accuracy of 94% with low-latency and low resource consumption.

Random Forest

The Random Forests (RFs) algorithm, or random decision forests, are a group of supervised ensemble methods which use multiple models with the aim of improving the prediction in classification, regression or other tasks. In summary, the basic component of RFs is the decision tree, that is a predictive model consisting of nodes (decision or end) and edges, represented by a tree-like graph and organized on multiple layers. In a decision tree, the predicted value to return is defined according to the reached leaf and the followed path. There are several versions of random forest, which differ in optimization, features, classification and in the design of forest [138].

Breiman wrotes an exhaustive article discussing random forest [29].

We can divide random forests approaches according to both features used or gesture types recognised, akin to precedent techniques. Liang et al. [132] developed a gesture recognition system that extracts user's hand poses from RGB-D images. Song et al. [213] relied on the ensemble methods to recognize in-air static hand gestures in real-time from colour data. Their work is applicable to whichever device equipped with RGB camera, and the classification consists of three steps, performed by likewise random forests: First, the hand is classified into three levels of depth, the next component classifies the hand according to shape, while the other one detects the location of fingertips and wrist for fine-grained performance.

Different works recognize gestures by using skeleton data in random forests. Canavan et al. [35], recognize hand static gestures using data received from a Leap Motion. They described the input through six features, such as fingers binary representation and max fingers range, starting from the joints contained in the received frames. The computed features are then fed to forests classifiers, achieving an accuracy of 100% with the University of Padova Microsoft Kinect and Leap Motion dataset [177], and 98.36% with a dataset of static poses created by the authors. Similarly, Joshi et al. [100] focused on skeleton data, introducing a multi-class RFs-based framework for continuous gesture detection, in which user input is represented combining 3D joint position with colour and appearance-based features. The evaluation of that framework is conducted using the NATOPS [214] and the ChaLearn [70] datasets, achieving an average recognition rate respectively of $87,35\%$ and $88,91\%$. The appearance data and the random forests find application also in [26,106] for detecting static poses, while [136] recognizes stroke gestures by applying RFs to segmented trajectories.

Balli Altuglu and Altun [13] evaluated the efficiency and efficacy of random forests to classify touch gestures, using different feature sets (gesture duration, mobility, pressure data, Hurst exponent, touch coordinates etc.). They evaluated the approach on two datasets: i) Corpus of Social Touch (CoST) [178] with 14 gestures, and ii) Human-Animal Affective Robot Touch (HAART) [179] that includes 7 gestures. This system participated in the Recognition of Social Touch gestures Challenge 2015, achieving an accuracy of about 50% and 70%, depending on the dimensions of training and test sets. A similar classifier method is presented by Gaus et al. in [75] for the same challenge. In that case, the authors coupled random forests with Boosting methods for labelling the touch gesture classes, using the Binary Motion History, histograms, and pressure data as features. They

recorded an accuracy of 67% for HAART dataset and 59% on the CoST dataset respectively.

The approaches described until now acquired the user movements by either camera, depth sensor or combined devices like Kinect and Leap Motion. Various methods based on other sensors exist and they have been presented in the later years. An example is the system by Zhang et al. [256], which defined a hand gesture recognition system for wearable devices, based on an electromyograpich (EMG) armband, a device able to track the electrical activity produced by skeletal muscles distinguishing finger configuration, hand shapes and wrist movements. Such approach is designed to be independent of the wearing position, further improving sensor's robustness. Fundamentally, the work suggested by Zhang et al. is composed of four basic elements: i) the first component filters and segments received signals by pre-processing; ii) the second decomposes pre-processed electromyograph data into Intrinsic Mode Functions (IMFs), (IMF representation allows to extract instantaneous frequencies using the Hilbert transform); iii) the next part extracts time and frequency domain from both EMG and IMFs; iv) finally, the last component is the random forests classifier for predicting the position of the wearable device and to detect the dynamic hand gesture using position and features. Another system was proposed by Smith et al. [210] for human-car interfaces. The input is received using an mm-wave radar sensor, which allows detecting precise features of fine motion, namely range, acceleration, energy total, energy moving, velocity, velocity, spatial dispersion, etc. These features are then applied to an RF classifier, allowing real-time recognition and achieving an average accuracy of 90%. Normani et al. [175] presented a low-power, offline, hand gesture recognition system that tracks user input through Lensless Smart Sensors, combined with infrared filters and five hand-held LEDs. For each received frame, it computes a set of features based on the Discrete Fourier Transformation. The system involves a random forest to recognize gestures, showing an accuracy ranged between 90% and 91%.

In gesture recognition, random forests are not employed only as classifiers. They are applicable even to either segment hand static poses from depth images, such as [141, 227], or in curve modelling to distinguish between strokes movements (drawing gestures) and hover movements [25].

2.1.4 Unsupervised Algorithms

In this section, we summarize different approaches which involve unsupervised classifiers in gesture recognition solutions. At the beginning of

the section about classifiers, we mentioned some peculiarities that concern unsupervised algorithms. They are characterised by a procedure in which they autonomously learn to distinguish regular patterns in the training data. In opposition to supervised algorithms, they are trained using unlabelled samples, subdividing and structuring data according to common features, in order to discern more about the samples. In other words, they learn automatically to distinguish the available classes that compose the training dataset [87]. Generally, unsupervised learning problems can be grouped into clustering and association problems: the former is related to the discovering of the inherent groupings in the data, while the latter is useful to discover the features that describe large portions of data. The main approaches are k-means for clustering problems and Apriori algorithm for association problems.

The unsupervised learning approach is employed either for a complex classification task that humans would not deal with correctly, or when the available training datasets are huge and not subdivided into classes, thus it is expensive to label samples manually. On the one hand, unsupervised algorithms relieve workload from humans both defining classes and for the training phase. On the other hand, it is not possible to analyse and evaluate the produced structure. In addition, the classifier is trained without using labels, thus there is no simple way for developers or designers to understand which and how many classes will be recognized, especially when the contents of the dataset are unknown. Consequently, the uncertainty about the detectable classes makes the unsupervised classifiers poorly suited for designing gesture interfaces. It is worth pointing out that a proper user interface has to know a priori the input vocabulary, and that notion is valid for both keyboard/mouse input and gestures. Such knowledge supports the development of guidance systems with the aim of providing feedback and feedforward, advising the user during the interaction.

Different works have proved the usefulness of unsupervised algorithms in gesture recognition systems. A vast part of these methods called semi-supervised algorithm, combine unsupervised and supervised classifiers trying to mitigate the aforementioned problem. The first ones are employed to segment both data and received videos or to extract features from the acquired input; the other ones are applied to the computed features to recognize gestures. For instance, Ge et al. [76] used a Distributed Locally Linear Embedding (DLLE) unsupervised algorithm for determining the inner structure properties of the acquired images. Then, it classifies static and dynamic hand gestures sending the estimated properties to a probabilistic neural net. Ma et al. [142, 221] recognize hand gestures employing

an extended version of the auto-encoder neural network, extracting edge features from Kinect images. Simao et al. [208, 209] designed a method for segmenting continuous data stream for finger and arm gesture recognition systems. For tracking motions through a wearable device, they relied on an unsupervised approach to divide input stream into either static or dynamic segments, registering an error rate of 2.70% with a sliding window of 20 frames. Liu et al. in [134] implemented a framework for transferring the learning based on an unsupervised sequence model: the Markov Random Field (MRF). They apply it to capture the dependency information between frames, joining the learned parameters with a Conditional Random Fields in a gesture recognition system.

Other works applied unsupervised algorithms on action recognition in different contexts, such as assisted living applications [162], human-robot interaction [12], or to detect Karate movements [84]. Moreover, unsupervised methods were also employed by Glowinski et al. to evaluate expressive motions [78, 79]. Substantially, they aimed to find the minimal representation necessary for describing affective gestures. Therefore, they extracted and compared a set of non-verbal gesture features, such as smoothness, symmetry, head leaning, energy etc. and they determined the minimal representation using the PCA, reducing features dimensions. After that, they compared the sets of features for classifying emotions with a two-steps clustering for classifying emotions.

2.1.5 Classification approaches using primitives

As we already mentioned, machine learning approaches applied the notion of sub-part definition in different research work. Such solutions are relevant for our work since they would be useful for designing interfaces. In this section we discuss them, highlighting their benefits in developing user interfaces and the differences with respect to our approaches.

Classification methods that identify a set of sub-parts (or primitives) common to different gestures have been proposed for either increasing the recognition rate or to reduce the training set size in learning-based approaches. Consequently, they are not thought to help UI developers. Primitives can be broadly defined as a set of distinguishable patterns from which either a whole movement or a part of it can be reconstructed. Different and specific definitions of "primitive" have been considered in the literature: they may represent basic movements (e.g., raising a leg, moving an arm to the left), static poses, or characteristic patterns of low-level signals like the Fast Fourier Transform. In the following, we give representative examples of each interpretation of the primitive concept.

In [249], primitives are identified using a bottom-up clustering approach, aimed at reducing the training set size and at improving the organisation of unlabeled datasets for speeding up its processing. Gestures are then labelled with sequences of primitives, which is close to a representation useful for building UIs. However, since primitives are identified automatically, they are difficult to understand for designers while creating feedback and feedforward systems, since they are salient for the recognition but they may be not for the interaction. In [44] primitives are defined in a context-grammar established in advance using a top-down approach, which is more suitable for UI designers since they can select meaningful primitives. However, such approach is affected by the opposite problem: designers may not be able to identify those easily distinguishable from each other, since usually they do not have a clear understanding of the underlying classification algorithms. Natarajan and Nevatia in [169] used primitives together with a three-level HMM classifier architecture for recognising i) the primitives, ii) their composition and iii) the pose or gesture. However, in this case, unsupervised learning was used for defining both primitives and their composition, which is not suitable for building UIs.

Vamsikrishna et al. [228] proposed a palm and finger rehabilitation system which combines discriminant analysis (DA), SVM and HMM. This system is able to recognize both rehabilitative gesture sequences and its components, analysing 3D gestural input acquired through Leap Motion. They employed DA and SVM to recognize isolated gestures, while gesture sequences are identified through HMMs. The method proposed by Vamsikrishna takes advantage of sub-parts to provide online feedback, helping users to perform rehabilitation movements with an average accuracy of 99.09%. On the one hand, this work shows an example of the potential of primitives in user interface design. On the other hand, in contrast to our work, they do not support the composition and the extension of the gesture dataset. Deng and Tsui [59] proposed a solution for segmenting isolated gestures in a continuous input stream through HMMs. They are able to recognize 16 different pre-defined trajectory primitives, expressed as line with a certain slope, employing a simple left-right Markov chain. Such an approach was evaluated using 50 samples, each one consisting of 4 distinct strokes, with an accuracy of 94.5%. The idea of representing a gesture as a sequence of atomic components underpins the solution proposed by Walter et al. [237]. They proposed a method which is able to automatically segment and cluster continuous input stream in a sequence of atomic poses, without using labelled samples. The segmentation and detection are performed analysing the received trajectories: a sequence of primitives, namely a gesture, is described through a mixture of Gaussian

components. They defined a different mixture of components for each gesture using an unsupervised algorithm. On the one hand, the method suggested by Walter et al. improves the performances in unsupervised gesture recognition systems, but on the other hand, it is difficult to apply in real UIs, considering the limited scope of the vocabulary.

A set of primitives that suits better the designer's understanding includes 3D properties of the movement trajectory. For instance, in [170] primitives are identified in a 2D video and used for classifying 3D movements. The primitives are functions on the 2D features that represent the user's state. Yang et al. [246] relied on primitives to detect a 3D motion trajectory. More in detail, the approach proposed by Yang et al. parses the whole user input into four types of trajectory primitives, based on their shapes. The four primitives recognisable are:

1. Primitive type A is a straight line including at least three consecutive points which are collinear.

2. Primitive type B consists of three or more sequential points which describe an arc on a plane, namely these points are not collinear but coplane.

3. Primitive type C consists of four or more points describing a right-hand helix if they are not coplane and the direction of every four sequential points is positive.

4. Primitive type D describes a left-hand helix; in contrast to primitive C, in each one of the four sequential points the direction is negative.

After parsing the trajectory in a sequential series of primitives, such approach further subdivides parsed primitives into sub-primitives according to a shape feature that describes the curvature (type B) or the torsion (type C and D). Then, the sequence of sub-primitives is detected by using trained HMMs, one for each class recognizable. Our approach G-Gene parses input user's movements in a similar way to Yang et al. solution; however, in contrast to their approach, G-Gene works online and provides a gesture modelling language supporting the definition of new gestures. Holte et al. [89] proposed a representation more linked to geometric features in the 3D space for identifying primitives; however, both approaches require the understanding of the underlying mathematical representation, which is not feasible for UI designers that usually do not have such skill.

The system proposed by Araga et al. [11] has the peculiarity of decomposing real-time hand complex gestures in a sequence of poses. More precisely, that system couples a hand posture classifier with a JRNN: the

former detects the poses in received video frames using RGB-D data, the latter assigns a predefined label to poses in the input sequence. They compared the suggested architecture against conventional methods, and their results show that the method outperforms the others if movements are performed slowly. John et al. in [98] labelled gestures using recurrent networks, while Ng and Ranganath combined radial basis function neural network and Markov sources to recognize, respectively, poses and complex gestures [173].

Kratz and Wiese [120] proposed a method to improve gesture segmentation algorithms using gesture execution phases and inertial measurement units as features. In their work, gestural execution is subdivided into three distinct phases: start, middle and end. They employed crowd workers to label these phases recognizing them by SVM classifiers and comparing the obtained results with heuristic approaches. Gesture segmentation is a significant issue in user interfaces development. On the one hand, this approach is helpful in defining more flexible UIs, showing feedback and feedforward based on the recognised phases. On the other hand, the approach proposed by Kratz and Wiese does not support the creation and definition of new recognizable gestures. The approaches discussed in this dissertation support developers to both define new strokes easily and track the movements the user has just performed.

To our knowledge, Kim et al. [111] proposed the most similar approach to the work discussed in this dissertation. It decomposes gestures into application-specific "primitive strokes", and uses a distinct HMM for modelling each stroke; each gesture is then modelled by a composite HMM obtained by concatenating the corresponding stroke models. Similarly to [122], this technique is valid for describing stroke sequences, which corresponds to the *sequence* operator in our proposed model language. Instead, as we will show in chapter 4, DEICTIC is able to define more complex composite gestures, including iterations (iterative operator), alternative paths (choice operator) and parallel stroke recognition (parallel operator). In addition, the method in [111] requires a re-training step with samples of the complete gesture for avoiding degradation in the recognition performance.

In this section, we reported and analysed different vision-based methods for classifying gesture in real-time or offline. If on the one hand they offer a higher accuracy, on the other hand they are not designed to provide additional information about the level of completion of gesture; in other

words, they do not support those feedback and feedforward mechanisms which help users during tasks. On the contrary, the approaches proposed in this dissertation support developers to both track the movements the user has just performed and recognize gestures accurately. In particular, these methods rely on HMMs to accomplished gesture recognition. In a HMM, the current state depends only on the probability of being in the incoming states; experimental tests showed that Markov models allow to determine the progress of a gesture execution in real-time when they are applied to local features. A deeper description of HMMs is included in the section 3.2.

2.2 Gesture Description Models

In this section, we summarise different compositional approaches based on heuristic gesture recognition. We point out that most of them do not include a formal evaluation of the recognition accuracy. Declarative models mainly aim to support developing gestural interfaces in different ways. For instance, by simplifying the creation of gesture recognizer, introducing a simple language tool for easily defining gestures, handling and processing the input received from various kind of devices, splitting a gesture in more sub-parts, generating events when a gesture and/or its primitives are recognized, assisting developers to define the interface behaviour according to the user input, and so on. In other words, description models allow reducing the developers' workload in building gesture interfaces. Generally, in these models, gesture recognition is accomplished employing heuristics rather classification approaches. On the one hand, this solution is more appropriate for improving the generation of feedback and feedforward in gestural interfaces. On the other hand, it lacks in accuracy and robustness.

Declarative approaches allow splitting a gesture into several sub-components. There are different compositional approaches based on heuristic gesture recognition. Kammer et al. [102] introduced GeForMT, a language for formalizing 2D multitouch gesture for filling the gap between the high-level complex gestures and data from low-level devices. For instance, the pan gesture, and the low-level device events generated according to the user input. In GeForMT the gestures are defined using an Extended Backus-Naur form grammar, and they are represented as a sequence of touches. The gestures are formalized using five different components:

- The pose function. In GeForMT, the user input is grouped into continuous contacts called pose functions, that describe the shape of the tracked touch. GeForMT is designed for handling finger and

hand input respectively represented as 'F' and 'H'.
- The atomic gestures, namely the primitives used to build more complex gestures. They describe the movements of the tracked touch and are categorized into point, hold, move, line, circle, and semicircle.
- composition operators, they are used for describing the temporal evolution of the gesture; Kammer et al. implemented three operators: sequence(,), parallel(*) and asynchronous (+).
- The current focus, which specifies the object, or objects, connected to a specific atomic gesture.
- The area constraints, which specify the relative movements among the different primitives. The possible relations are CROSS, indicates the overlapping, SYNC, employed when two primitives have to move in parallel, JOIN, describes a converging motion, and SPREAD, represents two atomic gestures that depart from each other.

Therefore, gestures are the result of the grammar productions of these five components. For example, figure 2.5 depicts the creation of the two fingers rotate gesture. The left part of 2.1 shows the expression which defines the underlying gesture. The rotate gesture is obtained from the composition of two different primitives: i) $1F(HOLD(o))$, describes the hold touch on a particular object; and ii) $1F(SEMICIRCLE(o))$, expresses the rotational movement of the other finger. The asynchronous operator + specifies that the second primitive is performed only while the first is still in progress. Differently from the approaches presented in this dissertation, GeForMT does not implement the choice operator, thus it is not able to describe the same gesture with alternative expressions. In our approaches, we use a subset of the primitives implemented in GeForMT, wide enough for defining a large set of different combined stroke gestures.

```
1F(HOLD(o)) + 1F(SEMICIRCLE(o))
```

Figure 2.1: The expression which defines the rotate gesture in GeForMT [102]

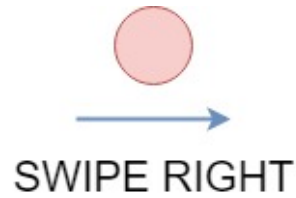
Another approach for multitouch events is proposed by Scholliers et al. [199]. They defined Midas, an architecture that supports the declarative definition of gestures using multiple tracking devices. In Midas, gestures are expressed and recognized according to a set of logical rules. In that case, a single rule represents a primitive and consists of:

- A name, which identifies the primitive.
- A prerequisite fact, which defines the input pattern to be recognized (for example the 2D position, the speed etc.).
- An action, which is the operation performed when the underlying primitive is detected. It represents the UI behaviour connected to this primitive.

A rule contains both the definition of a primitive and the UI behaviour. In addition, this architecture requires the developer to define a priority level for each rule, in order to avoid the overlapping between gesture with similar primitives, for instance, the single click and the double click. In Midas, complex gestures are expressed as the combination of two or more primitives by using four different types of operators:

- The temporal operator, which describes the time relations among the primitives, allowing to build complex gestures whose different components occur with the specified temporal relationship.
- The spatial operator, similarly to the temporal one, defines the spatial constraints that connect the sub-components.
- The list operator, which allows considering a set of events within a specific time frame.
- The movement operator, which is used in combination with the list operator. In summary, it verifies that a certain property is valid for each list's component.

Figure 2.2 depicts an example of a right swipe gesture created using Midas. The right part shows the expression which defines the swipe. It is expressed as a sequence of cursor events, performed using the same finger, within a small time interval and in which the user progressively moves the finger to the right. The approaches described in this dissertation employ an equivalent set of operators.




```
(defrule RightSwipe
    ?eventList[] <-
        (ListOf (Cursor (same: id)(within: 500)(min: 20)))
    (movingRight ?eventList)
=>
    (assert (Swipe (events ?eventList))))
```

Figure 2.2: The swipe right and its related expression defined by using MIDAS [199].

Inspired by Midas, Hoste et al. proposed Mudra [91], an extension of

Midas for multimodal interfaces. This architecture is able to unify the input stream coming from different devices, such as the skeleton data acquired by Kinect, cross-device multitouch information via TUIO and Midas, speech, and accelerometer data. Substantially, it can extract meaningful information from raw data, by processing and encapsulating multiple feature streams according to the semantic interpretation of high-level events. In other words, Mudra combines into a single software architecture the handler of low-level events and the definition of high-levels. When Mudra receives an input frame from one of the possible trackers, this is converted into a uniform representation that describes at high-level the user input. Mudas extends Midas in different ways. First of all, it expands the rule language by supporting more features, such as speech, hand movements, and multi-users. The underlying architecture strengthens the declarative language by providing new operators, for instance, the negation of an event and new attribute constraints. However, similarly to Midas, it couples the gesture expressions with the UI behaviour. Compared to Midas, our approaches are independent of the input stream but, on the other hand, we do not provide a multimodal interface for handling different input devices.

Khandkar et al. introduced the Gesture Description Language (GDL) [110] a flexible and extendible domain-specific language for multitouch applications, which supports developers in using gestures independently from the tracking device. Substantially, it allows both to declaratively define new gestures hiding their low-level implementation, and to use them in multiple hardware platforms. In GDL, the gesture expressions consist of three components:

- The name, employed to uniquely identify the gesture within the application;
- One or more validation blocks, which describe the temporal evolution of the gesture evaluating the raw data to detect the gestures. Each block corresponds to a primitive and consists of a simple boolean function.
- One or more return types, which represent the data notified to the application logic only when the gesture is entirely recognized. The returned data is notified using callbacks. It includes different information, such as touch positions, the number of touches, directions and so on.

Figure 2.3 represents the expression for defining a Lasso gesture, which allows selecting a set of multiple objects (visible in the left part). The right part depicts the code defining the gesture called Lasso: the gesture

is performed when a single touch forms a closed loop with the specified dimensions. GDL is a compositional model that facilitates the reuse of gesture recognizers for building other expression gestures. Unlike the other discussed approaches [91, 199], in GDL the gesture definition does not include the UI behaviour. Compared to our models, it is not able to generate events when a single block of a gesture is recognized. Therefore it is not possible to bind a handler to gesture's sub-parts.

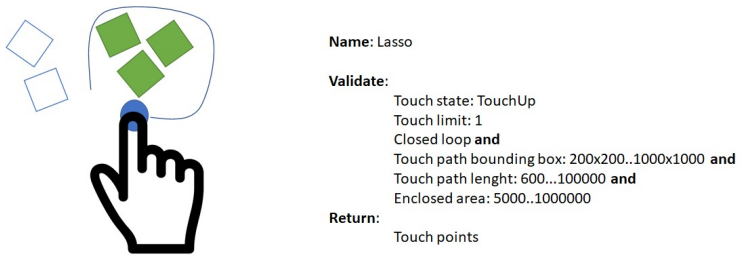


Figure 2.3: A "Lasso" gesture for selecting objects defined using a GDL [110] expression.

More structured and expressive declarative methods are Proton++ [114] and GestIT [215, 216], which we consider as two of the most complete declarative and compositional models for gestures. They clearly separate the concerns of UI description and behaviour, and they define a set of operators that are both understandable and effective for designers. Proton++ is a framework allowing developers to declaratively describe custom gestures, separating the temporal sequencing of the events from the code related to the UI behaviour. It aims to recognize multi-touch gestures in mobile or touch-screen applications. The framework was designed to reduce developers' workload in creating and extending the set of recognizable gestures. On the one hand, Proton++ provides to developers a graphical editor for building and modifying gesture expressions. On the other hand, it prevents conflicts between composed gestures, avoiding the addition of two o more similar gesture definitions. It highlights the common components between two or more gesture expressions, supporting the developers to either remove the ambiguous expression or assign a different probability to similar gestures. The gesture recognition is accomplished according to the detected touch events. Proton++ converts the acquired user input in a touch event stream, generating a new event when one of them is handled by the defined regular expressions. When one or more

gesture is recognized, this framework invokes their callbacks selecting those with the highest confidence scores. Gestures are defined as regular expressions, more specifically as a sequence of multi-touch events, where literals are identified by a triple composed of:

1. The event type, e.g. 1 touch down, move and up.
2. The touch identifier, e.g. 1 for the first finger, 2 for the second etc.
3. The object hit by the touch, e.g. the background, a particular shape, a widget, etc.

Proton++ allows developers to declaratively describe custom gestures through regular expressions, using the concatenation, alternation and Kleene's star operators. Figure 2.4 depicts an example of gesture definition through Proton++. The gesture is a simple two-hand rotate (pan) gesture, consisting of three different components, namely start, move and end. Each one corresponds to a different colour in the figure. Developers build the entire expression by composing different touch events. These latter ones are represented in the E_T^O form where: i) E specifies the event type that can be D for touch down, M for touch move and U for touch up; ii) the O indicates the touchable objects, s for the triangle shape in the figure or a for any object; and iii) T is an integer identifying the touch. The gesture expression is obtain composing such literals through the regular expression operator. Considering figure 2.4, the red part relates to the start part of the gesture and describes when the user touches the screen with two fingers. Then, the user can move iteratively the two fingers, diverging or converging the hand according to the rotation that she/he wants to be applied. The green part of figure 2.4 shows that component. Finally, the gesture is completed when the user raises both fingers from the screen, represented in the blue part of figure 2.4.

An improved version of the framework, presented in [113], supports tracking a set of computed attributes associated with an expression literal. For instance, developers can define a heuristic for validating the on-screen finger trajectory and bind it to touch move events. The framework raises the associated events (i.e. it recognises the literal) only if trajectory is accepted by the heuristic e.g., it moves north, north-west, south etc. GestIT [215,216] follows a similar approach, including operators for defining more advanced gestures. As discussed in [216], they are a superset of those defined by regular expressions. Gestures are modelled through expressions defining their temporal evolution, combining two main elements: ground and composite terms. The approaches proposed in this dissertation rely on GestIT, and its operators, to model complex and basic gestures as

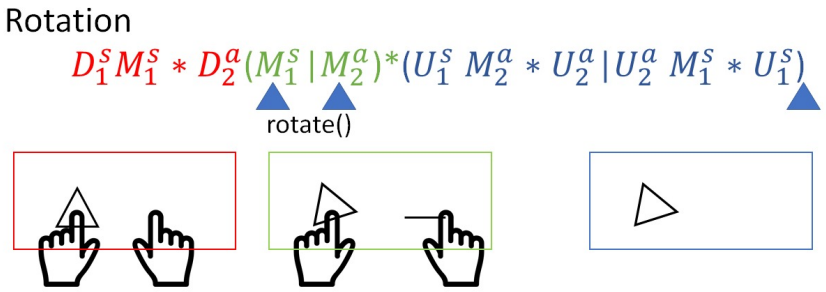


Figure 2.4: The rotate gesture defined in Proton++ [114].

a combination of small Markov sources. We analyse in more detail its components in section 3.3.

Another framework which supports developing gestural interface is GISpL [66]. GISpL is a formal language designed for different interaction modalities, including multitouch, mouse, tangible tokens, mid-air gestures and pointable devices. It allows developers to unambiguously describe the behaviour of gestural interfaces using a JSON-based syntax. In particular, GISpL represents a high-level interface between a gestural application and input devices: figure 2.5 shows the processing pipeline in GISpL. This framework can receive input from different kinds of devices. The user input, called also input event, is filtered by a list of regions, which are employed to extract a set of features, namely simple mathematical properties of the raw input data, such as the count of different object in a region, the matching accuracy between a predefined path and the one travelled by a given input object etc. Finally, features are employed in order to determine if a defined gesture is correctly performed. When a gesture is recognized, GISpL sends to the target application a specific event. On the one hand, GISpL supports the reuse of the gesture definition in other applications and the separation between the gesture recognizers and their effects on the interface. On the other hand, contrary to Proton++ and GestIT, GISpL does not define compositional operators, and it does not support the definition of more complex gestures describing their temporal evolution. Hoste and Singer [92] provide a useful comparison between these techniques.

Cuenca et al. proposed Hasselt [50], a declarative language for rapid prototyping based on finite state machines. It is designed to be an alter-

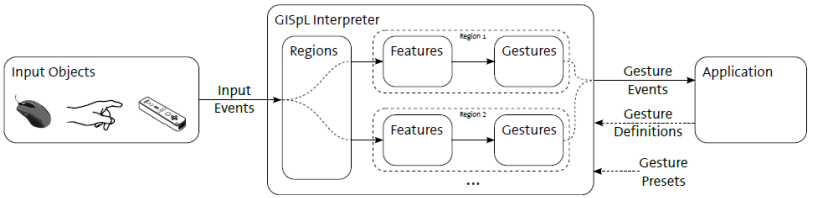


Figure 2.5: The processing pipeline which characterized GISpL [66].

native to event-driven languages, such as GestIT or Proton++, avoiding the 'callback soup' problem. More specifically, it allows defining gestures as a combination of one or more events, providing to bind these events to their handler automatically. Therefore, in Hasselt, each composite event is handled by generating an FSM. Every node contained in the FSM can be associated with a certain handler. This solution is suitable for providing feedback and feedforward in gestural interfaces.

Primitives can be employed to facilitate the definition of complex gestures. An example is a tool implemented by Krupka et al. in [122]. More specifically, Krupka et al. designed a language, and a set of tools, supporting developers to build and recognize poses and hand gesture definitions using a 3D camera. In their language, a gesture is a sequence of static poses that are characterized, and distinguished from the other, applying six basic proposition related to the fingers and palm center:

1. The palm and fingers pointing direction, e.g. 'the index finger point up while the thumb is on pointing left'. The direction of the object is quantized into: 'Left', 'Right', 'Up', 'Down', 'Forward' and 'Backward'.

2. The palm relative orientation, which describes the direction pointing from the wrist to the base of the middle finger.

3. The finger flexion, open or folded. A finger is considered folded whether its tip is near the hand.

4. The finger tangency, which expresses the relative distance among a pair of fingers as either 'touching' or 'not touching'. For instance, consider index and thumb fingers: this pair is considered very closed if their distance is lower than a certain threshold, therefore the proposition will be 'thumb (or index) is touching index (or thumb)'. Vice versa, the developer uses the proposition 'is not touching'.

Figure 2.6: The description of the 'Rotate Right' gesture using the three tools designed in [122], respectively: the first part (a) shows the poses defined using the visual gestural builder; the second (b) is the gesture description in C# codex; in the third part (c) the same gesture is represented in the XAML format.

5. The finger relative position. Given a pair of fingers, e.g. a and b, it defines the direction of a in relation to b.

All the possible propositions, and their combinations, supplies a suitable and complete language for defining complex gestures. The gesture expressions are saved in XAML format, and this underlying framework provides to developers three different means for building a gesture, a XAML text interface, a C# interface, and a visual gesture builder. Figure 2.6 shows the definition of a new gesture by using each solution. The represented gesture is a simple 'Rotate Right' that is composed of two poses: i) starting with the index above the thumb, and ii) ending with the index finger right of the thumb. A developer defines a gesture using a visual tool, namely modelling the static poses manually, or the C# syntax and XAML format, specifying each proposition and connecting the defined poses. The sequence of static poses is recognized using a set of convolutional table ensembles (CTE) classifiers trained on a large annotated dataset. In the evaluation tests, Krupka et al. noticed an average accuracy of 96%. This work is similar to our approach, allowing to declaratively define a gesture as a sequence of sub-movements. However, it does not support the definition of more complex gestures, such as iterating the same sub-component, describing the same gesture with a different path, and recognizing parallel strokes.

2.3 Gesture Interface Design

Over the last years, gesture interaction has become widely popular since the introduction of the iPhone in 2007. Compared to canonical input, such as through mouse or keyboard, a gesture may last for several seconds including the execution of several and complex movements. Gesture interaction may require a high cognitive load to both novice and expert users. In particular,

errors are more likely to happen in absence of a guidance system. Indeed, in this case, users easily fail in executing or forget a gesture. In this section, we summarize the frameworks and methods proposed for implementing guidance systems in gestural UI. These systems rely on feedback and feedforward for advising and supporting the users during tasks. Different works proved that they are a strong mechanism for learning gestures and reducing errors.

An efficient feedback and feedforward mechanism for stroke gestures may require the recognition of both the whole gesture and also its sub-parts. On the one hand, the feedback informs the user about the recognition process and the system's state, such as status of the executed movement path, error messages on incorrect movements or the current position on a correct movement path. In both cases, feedback is displayed during or after the execution of a gesture. Feedback mechanism is able to convert low-level information into graphical or text data. According to Bau and Mackay [14], it consists of three steps:

- Acquire recognition rate for each class from the recognizer; depending to the classifier, it can return discrete or continuous data;
- Convert and filter data;
- Represent feedback to users;

Through the feedback a user understands whether the performed movements are correct or not, and she may adapt the following actions if necessary. On the other hand, feedforward mechanism advices the user on possible continuations for the interaction. It is employed for showing the association between a particular gesture and the relative system behaviour, prior to the execution or the completion of the underlying gesture. This mechanism can also be used for guiding the user in performing tasks, for instance highlighting the possible ways for concluding the current gesture. Bau and Mackay [14] subdivides feedforward systems according to:

- Level of detail, which can range from show a minimal hint to describe the whole gestures.
- Update rate, which is closely linked to the level of detail. Similarly to feedback, feedforward can be displayed prior to or during the execution.

Delamare et al. [56,57] proposed a design space which defines and organizes those design option that can be applied to any dynamic guide for gesture-based interaction. In their design space, the feedback and feedforward mechanisms are described along four component groups (axes), which define the behaviour of a guiding system:

When This group describes the temporal characteristics of feedback and feedforward mechanisms according to three temporal steps, in which

they may intervene: i) the beginning, or trigger, related to when the user starts the gesture; ii) the execution, namely when the user is performing the gesture; iii) and the end, when the gesture is performed.

What In this group, Delamare et al. defined the content displayed through the feedback and feedforward mechanisms. The feedback mechanism may notify information about the state of the performed gestures (user's evaluation) and the gestures that are recognized or intended (system's evaluation). The feedforward mechanism guides the user about the set of gestures managed by the system (providing information about all the available gestures, a subset of them or only on one gesture).

How It characterizes the means used for showing the feedback and the feedforward to the user. They depend on both the sensory input type used and the visual modality adopted.

Where It describes the spatial relationship between the places where the user executes the gesture and where the feedback and feedforward mechanism are displayed.

Different properties characterise deictic and semaphore gestures; on the one hand the first type is employed mainly to select interface items, on the other hand, the second type is mostly associated to send commands and functions. Therefore, these differences are reflected also on the adopted feedback and feedforward mechanisms. Pointing systems rely on target expansions techniques to help the user in the selection. According to Guillon et al. [81], these methods consist of two basic movements:

- The expansion algorithm, which determines how to decompose the space among the selectable items (an example of expansion algorithm is the Voronoi tessellation);
- The visual aid displayed, which Guillon et al. [81] described along three axes:
    1. The dynamicity axis, namely the update rate of these aids. i) static if the target expansion algorithm does not change the aid during movements; ii) discrete when aid is updated at prefix points; and iii) continuous if the changes follow the user movement.
    2. The expansion observability axis, this point indicates if the target expanded area is displayed (explicit) or not (implicit).
    3. The augmented element axis, determining which interface item is expanded (cursor, target item or space).

One of the first gesture guiding mechanisms was proposed by Kurtenbach et al. in [125]. In this work, they combined two interactive mechanisms (crib-sheets and contextual animations) helping the user to learn gesture commands. In more detail, the system proposed by Kuternbach et al. employs a pop-up cheat sheet to display the list of available functions and their associated gestures according to the context. The user can see this graphic component at any time by performing a particular gesture (press-and-hold gesture). Besides, they coupled the crib-sheet with a series of animations that the user can visualize by clicking on their icons. These animations demonstrate how gestures should be performed, enforcing the learning mechanism. In contrast, this system presents different limitations, firstly it does not support demonstrations of geometrically parameterized gestures nor highlights it geometric gesture nuances. In addition, this solution takes for granted that users know which is the press-and-hold gesture to bring up the crib sheet.

Bau and Mackay introduced OctoPocus [14], a dynamic guide for continuously supporting users during the interaction in 2D gestural interfaces. It helps the user in learning, performing and remembering gesture sets by combining feedback and feedforward. In this scenario, feedforward notifies the user's current options, while the feedback shows how well the current gesture has been recognized. More precisely, the feedforward mechanism is implemented using a set of templates for each recognizable gesture class; the feedback system requires a continuous or discrete value from the recognition algorithm. Bau and Mackay combined the Rubine's algorithm [195] and the incremental turning angle representation to detect gestures and their progress. Substantially, when the user starts the execution of a command, OctoPocus displays a map of all available paths (feedforward) that may be followed for completing a correct interaction near the cursor, or touch, position. In this way the user can see and understand the correlation between gestures and commands while she is performing the gesture, facilitating her learning process. Once she starts to make a gesture, OctoPocus continues updating the dynamic guide: on the one hand, the less likely gesture paths gradually disappear according to the gestures detected, on the other hand, it highlights the performed movement with a black line (feedback). Figure 2.7 shows the underlying feedback and feedforward mechanism in a three-gestures scenario (cut, copy and paste). The left part shows the initial state when she starts the execution and the whole gestures are displayed. Instead, the right part of figure 2.7 illustrates how OctoPocus updates the dynamic guide varying the template path's thickness, indicating the evolution in the recognition It represents the tracked user trajectory with a black line, while it highlights

the more likely gestures with a thicker stroke. It is worth pointing out that the dynamic guide of OctoPocus may be displayed at any time or only when it notices a hesitation, for example when the user moves either the cursor or finger slowly. Therefore, it is compatible with both novice or expert users.

Moreover, in a follow-up work, Bau and Mackay proposed a technique for detecting the scale factor for a gesture, in order to anticipate the recognition [10], but a general solution is still missing. The works presented in this dissertation focused on providing such intermediate information through an automatic classifier generation starting from the gesture definition, which does not require the developer to understand the recognition technique.

Bragdon et al. proposed gestureBar [28], a learning multistroke gestures system for walk-up-and-use systems. It fosters learning of the possible interactions by exploring them without any prior introduction or training step, relying on a feedback and feedforward mechanism. Substantially, gestureBar consists of two components:

- A toolbar, that shows the whole commands and their icons: when the user clicks on one item, the corresponding commands are not executed, rather the application opens a new tab, the practice area, showing in which state the application will be (feedback) and explaining how to perform the gesture (feedforward).
- The practice area, in which the necessary steps for completing a

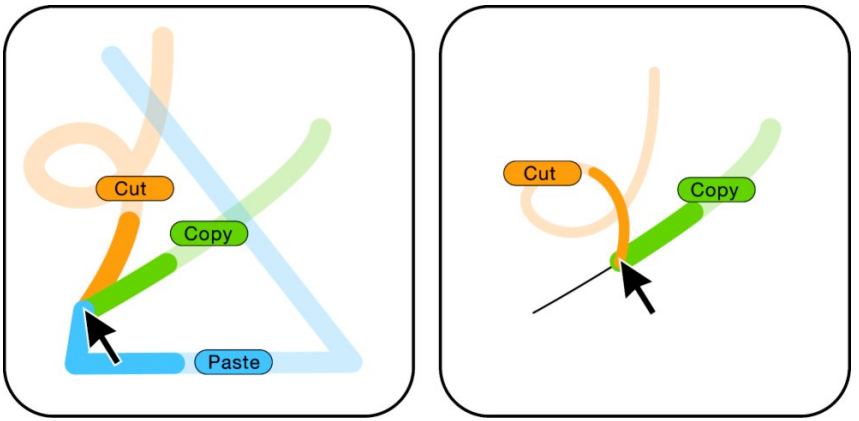


Figure 2.7: The evolution of feedback and feedforward guide in OctoPocus [14].

certain gesture are illustrated. This solution supports the user to learn the required steps. In this area, the user can also simulate the execution of a gesture: if the executed gesture matches a gesture, the application encourages the user. Vice versa, it notifies the wrong movements.

A similar feedback system is employed by Schwarz et al. in [203]. They proposed an architecture that supports users displaying continuous feedback about uncertainty for touch interfaces. It aims to avoid the cases when the user reaches an undesired state after performing the wrong command since she has misenderstood or missed the correct movement. The underlying architecture is based on prior work in modelling uncertainty using Monte Carlo [202] optimization. At run-time, that architecture tracks all the likely states according to user movements. At the same time, it reduces the number of alternative interfaces fusing the most likely interfaces into a single interface that communicates uncertainty and allows for disambiguation. In other words, the architecture proposed by Schwarz et al. notifies the user the most likely application's states in line with her current input. When the user completes the task, the command that corresponds to most likely gesture is executed, but the architecture allows the user to return to the previous state or to select another state.

LightGuide has been proposed by Sodhi et al. in [212]. It is a dynamic guide for gestural interfaces, similar to OctoPocus. It guides the user during the interaction in order to accomplish a certain gesture, by using continuous feedback and feedforward. More precisely, it helps the user in mid-air gesture by projecting guidance hints on her hand. These hints are displayed using an overhead projector and reply the movements which compose the gesture. In their work, Sodhi et al. evaluated different type of visual hint, including: i) follow spot, a 1D visual arrow which changes its size according to user movements; ii) hue cue, it uses negative and positive spatial colouring to indicate direction and the space a user should occupy; iii) a 3D arrow for visualizing the correct direction; iv) a 3D pathlet metaphor, which illustrates to user a small segment of the gesture; it shows the whole movement to perform, notifying the relative position of the user in the execution of the movement. The feedback and feedforward system is employed to:

- Show the gesture's progress state;
- Display the next movement to perform;
- Notify if the movement has been executed correctly, showing the distance between the performed movement and the right trajectory.

Inspired by LightGuide and OctoPocus, Alt et al. developed ShapelineGuide [3]. Similarly to other dynamic guide analysed up to now, even ShapelineGuide employs feedback and feedforward on the executed gestures, in order to support the user during the interaction. More precisely, it combines the advantages of OctoPocus and LightGuide for supporting mid-air gestures on large display applications. On the one hand, ShapelineGuide is designed for 3D gestures, differently from OctoPocus. On the other hand, compared to LightGuide, it is able to handle multiple gestures and more than one part of the body. Therefore, the feedforward mechanism constantly informs the user on the state execution of gestures by visualizing hints in real-time and displaying a text label to describe the corresponding action. Akin to OctoPocus, feedforward and feedback mechanism relies on different colours to notify the gesture progress. When the user starts the execution of a gesture, it highlights the next partial segment using a light colour and updating the graphical components, according to the tracked movements. In particular, that system applies different colours according to the hand which will be used to accomplish the gesture: i) orange for the left-hand, ii) green for right-hand, and iii) yellow for multiple gesture visualization. Besides, ShapelineGuide provides further feedbacks employing a set of icons to make the user interface comprehensible for people that are unfamiliar with the commands. In particular, they categorized icons in two blocks, the first consists of generic commands, such as zoom, rotation and so on, the other block suggests which hand she has to use to accomplish the gesture. An example of interaction in ShapelineGuide is depicted in figure 2.8.

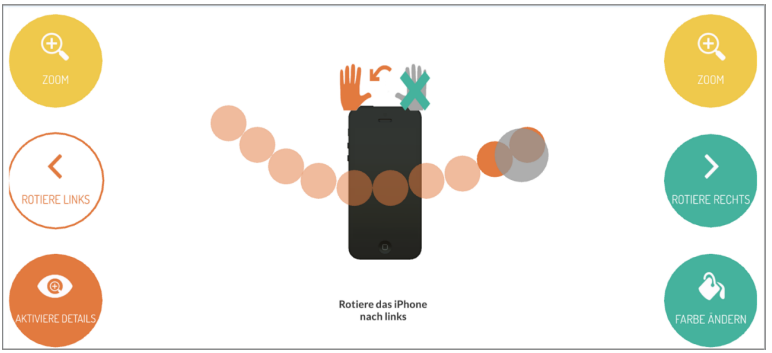


Figure 2.8: An example of gesture guiding in ShapelineGuide [3]: the hand-icons, located above the smartphone, indicate wich hand the user has to use; the gesture's path is described using a sequence of orange circle subdivided in more segments.

Guillon et al. [81] presented Expansion Lens, a new target expansion technique for pointing tasks. This method employs the Voronoi tessellation as expansion algorithm, and a circular area centred on the cursor through which the user can see the target expanded area. In another work, [58], Delamare et al. extended OctoPocus to support 3D gesture guiding system, showing the set of 3D in-air hand gestures as 3D pipes. Similarly to ShapelineGuide, this solution helps the user to understand the execution of a gesture, displaying which movements have been correctly recognised. A typical problem in 2D-3D guiding systems is finding the best positioning for the suggestions in the UI layout, in order to avoid the overload of the scene. In OctoPocus3D, Delemare et al. addressed this problem using depth cues, that is chosen among a set of the visual cues for 3D space perception suggested by Ware in [238]. OctoPocus3D relies on depth cues to show different feedback and feedforward mechanisms. Similarly to the original OctoPocus, the solutions proposed by Delemare et al. renders the available paths in two different parts: i) a coloured prefix and a transparent suffix; besides, when the user completed the gesture, the user path becomes green if it is right performed or red in the other case. During the execution, the user can rotate the 3D scene visualizing the 3D gesture paths which are available in a certain moment. In addition, it provides a digital representation of the user's hand as a white sphere, allowing the user to perceive if she is in front of or behind the path to follow. In addition, the UI background is designed to support the user by increasing her linear perspective and the contrast of the 3D scene.

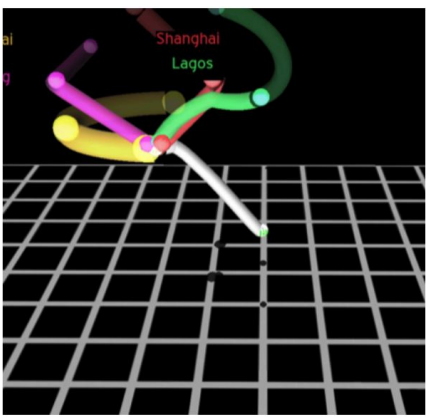


Figure 2.9: The guiding system displayed using OctoPocus3D [58]

Rovelo et al. proposed Gestu-Wan [193], a gesture guidance system for mid-air gesture in walk-up-and-use displays. In more detail, it aims to

facilitate the execution of mid-air gestures without prior training. In the solution presented by Rovelo et al., gestures are described as sequences of static poses and structured hierarchically. When the user starts the interaction, the guidance system provides the display of a graphical representation of the possible postures. The set of the showed postures depend on the leaf node reached inside the gesture hierarchy. Once a posture is matched, the system updates aid which is notified by i) turning in green the detected posture, and ii) shifting the gesture hierarchy showing the other available postures until a gesture is not completed.

The mechanism implemented in LightGuide is particularly useful for programming movements that require accuracy and proper technique. The problem of learning movements is also addressed by Anderson et al. in YouMove [6]. This is an architecture which supports users to learn physical movement sequences with an augmented reality mirror, where the user movements are acquired through tracking devices such as Kinect, and so on. First of all, YouMove provides an editing interface through which trainer can both record the sequence of movements that compose the training, and define the main poses and their global movement parameters (joint positions). The learning process is accomplished by guiding the users on the execution of training exercises, based on the distance between the tracked user's joints with those recorded by the trainer. In this case, the dynamic guide highlights those user's body parts which are in the wrong position. Besides, this system advices the user by illustrating those joint that she needs to move in order to cue upcoming movements. YouMove is designed to reduce the level of aid according to the expertise acquired by the user. Figure 2.10 shows an example of the posture guide implemented in YouMove: the figure illustrates a scenario where the system notifies the incorrect joint positions (feedback).

Daliri et al. introduced a system [53] that helps users in performing bend gestures correctly on flexible devices. It relies on a feedback and feedforward mechanism for advising the user about the correct location, direction and angle of each available gestures. In addition, the system proposed by Daliri et al. notifies the user when a gesture is performed incorrectly and helps her with the aim of correcting the mistake. More precisely, they implemented three different visual feedback guides:

Central Circle : in this design, the feedforwards are located at the edge of the circle, and are represented through arrows; each of them corresponds to a gesture, and its orientation and colour indicate the gesture's direction. The feedback is displayed using the empty space in the middle of a circle to show the state of the gesture.

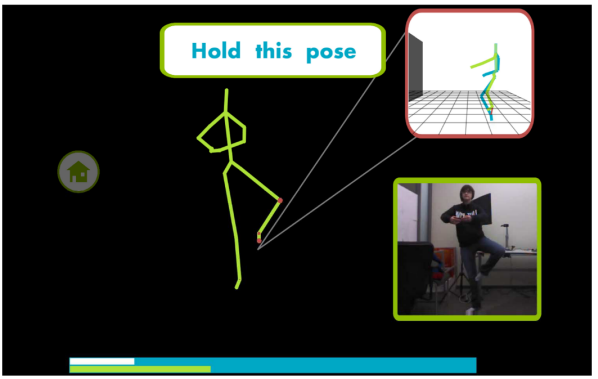


Figure 2.10: The feedback and feedforward mechanism employed in YouMove [6].

Arrows , while the feedforward system is similar to the central design, the feedback is implemented by using arrows.

Clear Sheet , it combined feedback and feedforward in a single text menu.

In this section, we described the main approaches for recognizing gestures based on declarative methods. It is worth pointing out that declarative models reduce the workload in building gesture interfaces in different ways, for example by simplifying the creation of gesture recognizer or the description of new gestures. In particular, declarative methods aim to support feedback and feedforward mechanism by employing heuristics rather classification approaches. If on the one hand this solution makes available the recognition status of gesture and its sub-parts at any time, on the other hand it offers a lower accuracy than vision-based methods. The two approaches discussed in this thesis filled the gap between these techniques, supporting developers by tracking the movements the user has just performed and recognizing gestures accurately. In particular, DEICTIC relies on GestIT's operators to describe stroke gestures and define HMMs composed; we use GestIT since they are a superset of those included in Proton++.

Chapter 3

Background

The goal of the approaches discussed dissertation is supporting the development of a gesture guiding system by combining accurate classification techniques, e.g. HMMs, and GestIT [215, 216] operators. On the one hand, HMMs sono adatte al riconoscimento online, in quanto lo stato attuale del modello dipende dallo stato precedente; questa soluzione Ã¨ utile per riconoscere parti di gesture. On the other hand, GestIT provides a set of operators that are a superset of those included in Proton++[ref].

In this section, we described the main approaches for recognizing gestures based on declarative methods. It is worth pointing out that declarative models reduce the workload in building gesture interfaces in different ways, for example by simplifying the creation of gesture recognizer or the description of new gestures. In particular, declarative methods aim to support feedback and feedforward mechanism by employing heuristics rather classification approaches. If on the one hand this solution makes available the recognition status of gesture and its sub-parts at any time, on the other hand it offers a lower accuracy than vision-based methods. The two approaches discussed in this thesis filled the gap between these techniques, supporting developers by tracking the movements the user has just performed and recognizing gestures accurately. In particular, DEICTIC relies on GestIT's operators to describe stroke gestures and define HMMs composed; we use GestIT since its operators are a superset of those included in Proton++[ref]. In this section, we reported and analysed different vision-based methods for classifying gesture in real-time or offline. If on the one hand they offer a higher accuracy, on the other hand they are not designed to provide additional information about the level of completion of gesture; in other words, they do not support those feedback and feedforward mechanisms which help users during tasks. On the contrary, the approaches proposed in this dissertation support

developers to both track the movements the user has just performed and recognize gestures accurately. In particular, these methods rely on HMMs to accomplished gesture recognition. In a HMM, the current state depends only on the probability of being in the incoming states; experimental tests showed that Markov models allow to determine the progress of a gesture execution in real-time when they are applied to local features. A deeper description of HMMs is included in the section 3.2.

In this chapter, we first define the concept of interactive gesture. After that, we summarise the definition of a Hidden Markov Model, in order to point out the main recognition properties we exploit in our methods and explain the notation we use dissertation. Finally, we describe GestIT, the declarative method we used as a starting point for our notation, analysing its organization and benefits.

3.1 Gesture

According to Kurtenbach and Hulteen [21, 124], a gesture is "a motion of the body that contains information". This definition includes speech, face movements, tactile input, etc. Generally, in the HCI field, gestures represent an additional input modality allowing users to use movements for interacting with a system or application. Besides, some research work relies on movements and expressive gestures to describe, and understand, emotional aspects of human motion and interaction between other persons and machines. These works aim to better understand such aspects, in order to improve the user experiences in interactive multimedia systems. In [126, 234], Laban introduced the theory of effort, where he formalized the dynamic nature of human movement in different contexts, for instance, dance and theatre, describing the connections among movements, space and time. The extraction of expressive gestures from dance, or similar activities, is also studied by Camurri et al. in [32, 34] by determining the features characterising the communication in music and dance performance. The authors presented a unified conceptual framework for analysing expressive gestures, which is able to automatically extract expressive gestures from physical movements and sound by adopting a layered approach:

Layer 1 collecting the input data, acquired from different devices and sensors, some techniques for background subtraction, motion detection, motion tracking and so on;

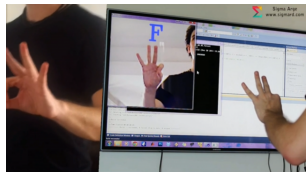
Layer 2 including several computer vision methods employed in order to detect movement or trajectory of points;

Layer 3 extracting motion descriptors and expressive cues to represent human action;

Layer 4 extracting automatically expressive gestures by means of a series of supervised algorithms, for example, for classifying human movements in term of basic emotions.

Over the last years, different gesture classifications have been proposed to categorize gestures [30, 105, 163, 240].

For our purposes, it is relevant to distinguish gestures according to the movements they describe: static or dynamic. The former category represent body postures that are not conditioned from the temporal dimension. In other words, the user is asked to mimic the requested pose, without taking into account the movement sequence. They are used, for example, in sign language recognition systems as shown in figure 4.9(b). Dynamic gestures define a set of movements that the user must execute accurately for providing input to a device. Compared to static gestures, they are characterized by a wider syntax allowing developers to describe more complex gestures. Therefore, a large part of gestural interfaces exploits dynamic gestures, for instance in rehabilitative physiotherapy, guiding the patient to correct the performance of movements, or in video-gaming environments. Figure 4.9(a) depicts an example of dynamic gesture performed using hands.



(a) An example of static gesture recognition.

(b) This figure shows an example of interaction through dynamic gesture: a videogame system convertes into commands the users movements tracked using a Kinect device.

The approaches proposed dissertation focus on stroke gestures, which are a particular type of dynamic gestures employed for representing figures in two or three dimensions. Stroke gestures can be distinguished in two different classes, depending on the number of movements that compose them:

- Unistroke gestures, which are performed through a single uninterrupted movement without pauses. In order to be recognized, devel-

opers may enforce the start position (the black dot) and the path direction (the arrows). An example of unistroke gesture is depicted in the left part of figure 3.1.

- Multistroke gestures, consisting of multiple separated strokes. In figure 3.1 each stroke is represented as a segment that starts from the grey dot.

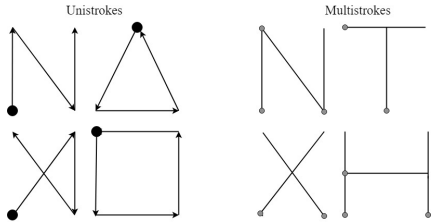


Figure 3.1: The two types of strokes, unistroke and multistroke.

3.2 Hidden Markov Models

The recognition solutions proposed dissertation rely on HMMs to model user input, determining the progress of a gesture execution. In this section, we summarize how they work, showing and describing the main properties through some examples.

A Hidden Markov Model (HMM) is a probabilistic model that maps a sequence of observations into a corresponding sequence of labels. It has been used in the literature for solving various types of pattern recognition problems, due to their effectiveness in modelling the correlations between adjacent symbols. For instance, HMMs are widely used in speech recognition [19] for finding the sequence of phonemes which form the actual uttered sound, or in biological analysis for modelling proteins, DNA sequencing, and alignment [251]. They represent an extended version of finite state machines, in other terms, a special case of weighted automata defined through a finite set of states and a set of transitions with the associated weights. HMMs satisfy the *Markov assumption*, stating that the probability of being in a particular state at the time t depends only on the probability of being in its incoming states at $t-1$. It means that the model determines the state at the time t according to a finite set of previous states. The models that satisfy this assumption are called Markov chains. There are many types of Markov chains, which differ on the number of considered past states for firing transitions. In the easiest form

of a Markov chain, the current state depends only from the immediately previous state 3.1.

$$P(s_i|s_1, s_2, \ldots, s_{i-1}) = P(s_i|s_{i-1}) \quad (3.1)$$

(Markov Assumption)

Formally an HMM is a quintuple $\lambda(S, V, A, T, B)$, that can be defined by:

- A finite set of states S, including a start and a end state denoted, respectively, as s_0 and s_f.
- A vocabulary of values for the observable events V. This vocabulary can be composed by either discrete (e.g. a finite set of labels) or continuous values (e.g. a set of real number).
- A transition probability matrix A. Given two states $s_i, s_j \in S$, $A_{i,j}$ is the probability of firing the transition from s_i to s_j. Among these states, we denote the initial state as s_0 and the final state as s_f. Obviously, the dimensions of that matrix is connected to the number of states which compose the HMM (if the HMM has n states, A will be a square matrix of order n).
- A topology T, which determines the inner structure of the HMMs conditioning the connections between states.
- A sequence of observation likelihoods B. Given a value $v \in V$, $b_i(v_j)$ is the probability of observing the value $v_j \in V$ being generated from the state $s_i \in S$. If the observable events have continuous values, $b_i(x)$ is the probability density function for generating a value from a state $s_i \in S$. The probability of observing any value in s_0 and s_f is zero.

Each state represents one of the possible events that has an impact on the modelled environment. These events are hidden i.e., they are not directly observable in the world. For example, consider an HMM of n states that is designed to recognize gestures. In a gesture interaction domain we do not observe the gesture performed by the user, but only the input device data. scenario, we use these events to represent the recognition states of a gesture. Therefore, the probability of the state at the time t will be between the value 0 (the gesture has not been recognised) to $n - 1$ (the user has completed the execution of that gesture). Besides, these events influence the set of observable events. According to this sentence, the observable events at the time t depend only from the actually state 3.2. The observable events define the values measurable in the modelled environment and they can be acquired through different types of sensors. For instance, in the previous example, the observable events

at the time t describe the set of features extracted from the user input as they are tracked by an input device (e.g. a touchscreen or a Microsoft Kinect).

$$P(v_i|s_1,\ldots,s_i,\ldots,s_{i+k},v_1,\ldots,v_i,\ldots,v_{i+k}) = P(v_i|s_i) \tag{3.2}$$

(Obvervation independence)

In an HMM, the relationship between an observable event value and a given internal state is modelled either by probability distribution or a probability density function, respectively in the discrete and continuous case 3.3. A probability distribution is also employed for defining both the prior probability at the time $t = 0$ (i.e. when we move from the initial state S_0) and the transition probability between two different states. The weights which characterize an HMM represent these transitions and the probability of firing them. Therefore, considering a single state, the sum of the weights on all its transitions to any other state is 1. According to this properties, the weight matrix satisfies equation 3.3.

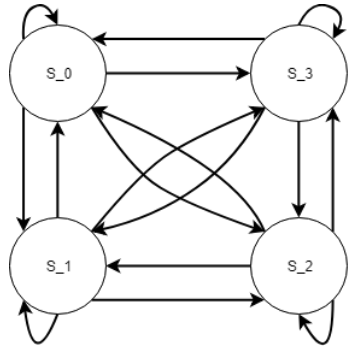
$$\sum_{j=0}^{f} A_{ij} = 1 \quad \forall i \tag{3.3}$$

(Outgoing probabilities)

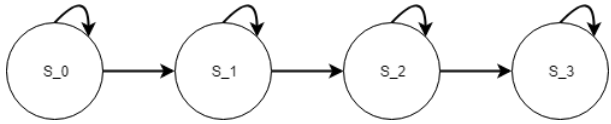
In general, not all states of an HMM are connected. The arrangement of the transitions depends on the adopted topology, which has an impact on the HMM effectiveness. There are several types of HMM topologies [242], each one offering advantages and disadvantages according to the considered domain. The main topologies are:

- Ergodic topology. It is the most complex structure, in which all states are fully connected. It can generate any sequence of observations (figure 3.2(a)).
- Self-loop topology. Each state is connected with both itself and the next state. It extends the simplest model, the linear chains, where each state is connected only with the next one (figure 3.2(b)).
- Left-right topology. This model mixes the ergodic and self-loop topologies. It represents a linear chain where it is not possible to go back. For example, if at the time t, the model in figure 3.2(c) is in state S_2, in $t + 1$ it will be either in S_2 or in the next state S_3.

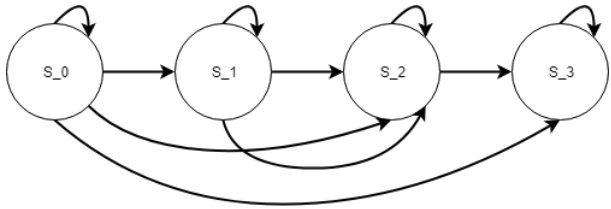
HMMs can be used as a supervised classifier, by learning the probability distributions through a set of labelled samples. More in detail, there are



(a) Ergodic model



(b) Self-loop model



(c) Left-right model

two algorithms for the training: Baum-Welch [15] or Viterbi learning [188]. In their original form, both methods do not employ labelled samples, thus they may be used as unsupervised training algorithms. However, they can be extended in order to assign labels during the training phase. On the one hand, the Baum-Welch algorithm assigns soft labels by using Expectation Maximization (EM) [27], and updates the distribution parameters by employing the Maximum Likelihood Estimation (MLE) [154] method. On the other hand, Viterbi relies on the Viterbi algorithm [72] for assigning labels to each sample, updating the distribution according to the assigned

labels. The Viterbi algorithm is also employed in the evaluation phase.

ALGORITHM 1: Forward-Backward Algorithm [196]

Input: ev, a sequence contains n evidence values - *prior*, the transitions propabilities refers to the state s_0 (namely $A(s_0)$)

Output: sv, a vector of probability distributions

fv, the vector of forward messages for the steps $0, ...n$

b, a representation of backward message, initially all ones

sv, a vector of smoothed estimates (it represent the returned vector)

$fv[0] \leftarrow prior$

for $i = 1 \in n$ **do** $fv[i] \leftarrow$FORWARD$(fv[i-1], ev[i])$ (namely$P(s_{i-1}|V_{1:i})$) ;

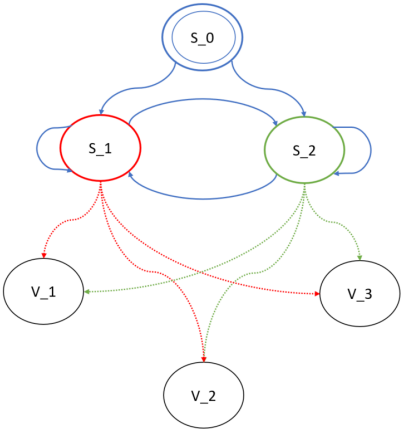
for $i = n$ *downto* 1 **do** $sv[i] \leftarrow$NORMALITATION$(fv[i] \times b)$;
$b \leftarrow$ BACKWARD $(b, ev[i])$ (namely$P(v_{i+1:n}|s_i)$) ;

return sv

Figure 3.2 shows an example of a simple HMM with three states (start + two states) and three observable events. At the beginning, we are in the start state S_0. At the time $t = 1$, there is a high probability to go in S_1, because we have the 68% of possibilities to go in state S_1, and the 32% of probability to move in the other state S_2. Supposing that we have reached the state S_1, now the observed value is determined according to the probabilities in the table "Observation-States". Following the likelihoods contained table, the most likely emission is associated to V_2. After determining the observable state at time $t = 1$, from the current state, we can either move to the other state S_2 (with a probability of 80%) or remain in the same state (20%). scenario, we can determine the probability that a particular sequence of observations belongs to this model by using the forward-backward algorithm(1). For example, the probability related to sequence (V_2, V_1) is 14.85%, obtained following the forward-backward algorithm:

- At the first step, we apply the forward algorithm that returns this distribution $fv = [[0.39, 0.04], [0.0141, 0.1344]]$;
- Then, the backward algorithm relies on the vector of forward messages for computing the distributions over past states, returning $[[0.35, 1][0.3, 1]]$
- Finally, the probability distributions are mapped to a single value, that indicates the posterior probabilities of the sequence (V_2, V_1).

In the literature, it is possible to find different types of specialized HMMs, which are designed to deal with different types of problems. Some



Start Probability	S_1	S_2
	0.68	0.32

Transitions States	S_1	S_2
S_1	0.2	0.8
S_2	0.4	0.6

Observations-States	V_1	V_2	V_3
S_1	0.15	0.65	0.2
S_2	0.4	0.1	0.5

Figure 3.2: The inner structure of a simple HMM with three states and three observable states.

examples are the profile-HMMs, that we rely for supporting G-Gene 5 in measuring the distance between two strings. approach, we achieve a real-time recognition comparing the strings that represent the user input with the others that represent one of the correct ways for achieving a particular gesture. Commonly, profile-HMMs are employed for modelling character sequence profiles, widely used to model and analysing biological sequences. They have a strictly acyclic linear left-to-right structure in their transitions and offer a high effectiveness and robustness to noise in representing sequence profiles [251]. For these reasons, profile-HMMs are used for gene sequencing. In our approach, we use them for aligning the stroke input string representation to the most likely gesture profile. More in detail, a profile-HMM determines an alignment between two strings (gestures in our case), finding the most likely correspondence between their characters. They define the alignment using three operations:

1. A *match*, representing the correspondence between a character in the reference and the input string;
2. An *insertion*, representing an additional character in the input string, not included in the reference;
3. A *delete*, expressing that a specific character contained in the reference is not included in the input string.

A profile-HMM reflects such operations in its structure. For each character in the profile string, it has three basic hidden states: a match, an insertion, and a delete state. The transitions follow a left-to-right topology,

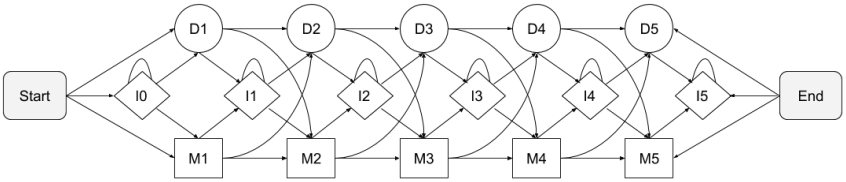


Figure 3.3: The inner structure of a profile-HMM. The Mi, Ii and Di are respectively the match, the insertion and the delete state for the i-th symbol in the reference string.

connecting all the hidden states representing the three operations of two adjacent symbols in the profile string.

Figure 3.3 shows a sample inner structure for the profile-HMM describing a string with 5 characters. The HMM contains a match, an insert and a delete hidden state for each symbol in the profile, together with the start and final state. From an operation state corresponding to the i-th character, the HMM allows firing a transition either to an operation on the $i+1$-th symbol or to insert characters in the i-th position. It does not allow going back to the $i-1$-th symbol (left-to-right topology). The emission probability is uniform in the insertion and delete states for all characters, while is forced to 1.0 in the match state for the corresponding character. We arbitrarily defined the set of the transition probabilities (see Figure 3.3) from a match to the next delete or insert state (0.05), from the delete or insert state to the next match (0.15), from a match to the next match (0.9), from a delete to next match (0.7) and the insert iteration (0.7).

3.3 GestIT

section we detail the model language GestIT describing its main components, both ground and composite terms. Besides, we illustrate some examples of gesture expressions obtained using GestIT.

GestIT [215, 216] is a framework which is designed to support the declarative and compositional definition of new gestures, independently from the devices used to track the user movements (Kinect, Leap Motion, touch screen, and so on). It defines the temporal evolution of a gesture through expressions, following an approach similar to Proton++ [113, 114]. These expressions are modelled using a set of operators, which represent a

superset of those included in Proton++ and that are defined by regular expressions [216]. In other words, in GestIT a gesture is a composition of primitive gestures, defined combining two main elements: ground and composite terms.

3.3.1 Ground Terms

A ground term is the smallest block for defining a gesture: it describes an atomic event which cannot be further decomposed. It is associated with a value change of a *feature*, such as the pixel coordinates of a touch on the screen or the position and rotation of a skeleton joint. The atomic events are recognized by using heuristics; such solution, on the one hand, supports the temporal composition of more complex gestures, but on the other hand, it has lower performance compared to machine learning approaches. In general, the expressions in GestIT includes three types of ground terms:

Start it is related to the start part of the gesture. It defines the movement that the user performs to start the gesture. Usually, it is represented through a static movement or a speech command, minimizing both the input ambiguity and the continuous input segmentation problems. At the time $t = n$, the framework that implements GestIT verifies the temporal execution of those gestures for which the start term has been recognized in $t = n - 1$.

Move this type is employed to describe the movements which compose gestures. For example, they describe a circular movement of the right hand or a swipe gesture performed by touch. GestIT supports different features, allowing to define the movements using temporal coordinates, direction, covered distance, speed, etc. It is worth pointing out that if the user misses the correct movement, the framework generates an error event resetting the gesture in the start position.

End as its name suggests the end term describes the end part of a gesture. Similarly to the start term, it represents a static pose or a speech command. The execution of a gesture is accomplished when all its components are recognized.

3.3.2 Composite Terms

Composite terms are used for defining more complex gestures by relating the ground terms through a set of operators:

Iterative g^*, the iterative operator supports the repeated execution of the gesture g an indefinite number of times. Generally it is combined

with the disabling operator in order to avoid an infinite gesture definition.

Sequence $g \gg h$, this expression represents a sequence of the movements that must be executed in the specified order, from left to right: first g, then h.

Disabling $g[> h$, it defines a gesture (g) that stops the recognition of another one (h), for example an iterative movement.

Parallel $g \times h$, the parallel operator describes the execution of two or more movements at the same time. In that case, the gesture $g \times h$ is recognized if the user performs both movements simultaneously.

Order Independence $g| = |h$, the order independence is the opposite of sequence operator, and it is employed when its components can be performed in any order. When the user completes the two movements, thus the framework recognizes $g| = |h$.

Choice $g|h$, in GestIT the choice operator is used to define a gesture that can be performed in different ways. case, the gesture $g|h$ is recognized when the user completes either the first (g) or the second (h) component.

In DEICTIC we use these operators to define a simple modelling language for stroke gestures.

We illustrate two simple examples of building gesture expression in GestIT. The first expression 3.4 defines the gesture for selecting an object in the user interface. The user can perform the grab gesture by using either the right or the left hand. Therefore, the gesture definition contains the choice operator. Regardless from the used hand, the gesture begins with the closing of the dominant hand sH[closed] (the start term). This movement defines the object selection. After that, the user can move the closed hand for e.g., moving the selected object (mH^*)(the move term) and the gesture terminates (the second static pose) when the user opens the hand (sH[open]). In the second expression 3.5, we define the pinch gesture that involves two simultaneous touches. The gesture begins when the device detects two touches without any order ((sT_1[down]$| = |sT_2$[down])). After that (sequence), the user can move iteratively both fingers on the screen ($mT_1^* \times mT_2^*$). The two fingers may be moved simultaneously, thus the sub-gesture includes the parallel operator. Finally, the pinch gesture is completed when she raises one of the two fingers, disabling the iterative movement (sT_1[up]$| = |sT_2$[up]). The order independence operator ensures that the user can lift fingers in any order.

$$(sH_r[\text{closed}]) \gg (mH_r^*[> sH_r[\text{open}])|(sH_r[\text{closed}]) \gg (mH_r^*[> sH_r[\text{open}]) \tag{3.4}$$

(Grab gesture)

$$(sT_1[\text{down}]| = |sT_2[\text{down}]) \gg ((mT_1^* \times mT_2^*)[> (sT_1[\text{up}]| = |sT_2[\text{up}])) \tag{3.5}$$

(Pinch gesture)

In the approach discussed dissertation, we start from the modelling experience we acquired in defining GestIT and we try to fix its main drawback, which are shared with Proton++: they both use heuristic recognition approaches for ground terms, which do not guarantee a good recognition accuracy. We advance the state of the art field by showing that the modelling technique can be used for automatically creating highly-accurate HMM classifiers supporting the identification of gesture sub-parts during the execution and the prediction of the most likely gesture completion.

Chapter 4

DEICTIC

Over the years, different machine learning-based solutions have been proposed to detect gestures online; a large part of them supports developers by providing a high accuracy for different types of gestures. In general, the recognition process requires the complete gesture sample before recognizing it. On the one hand, including classifiers techniques solves the accuracy problem, on the other hand, it does not support guidance systems. In this chapter, we describe the first method that we proposed for filling the gap between classifier approaches and declarative techniques. This method, called DEICTIC (DEclaratIve and ComposiTional Input Classifier), achieved a highly accurate stroke recognition by combining HMMs and GestIT [215, 216] operators independently from the devices employed to acquire the user input.

We start introducing the definition of the stroke gesture modelling language in the section 4.1. Then, in section 4.2, we describe the composition algorithms for building composite HMMs. After that, the section 4.3 summarises the advantages and limitations of DEICTIC for the UI development. Finally, in the section 4.4 we describe the experimental tests performed to measure the accuracy on two stroke gesture datasets.

This chapter is an extended version of the work in [40, 41].

4.1 Gesture Description

In this section, we introduce a simple modelling language for stroke gestures that follows an approach similar to other declarative approaches (e.g., GestIT [215, 216], Proton++ [113, 114], etc.). It starts from the definition of a ground term set and it obtains more complex gestures

through composition. We have been inspired by commonly-used UI widget toolkits, which allow developers to create their interfaces exploiting simple, general purpose objects that shape complex visualisations through the composition. The language is both understandable for gesture designers and supports the automatic classifier generation from gesture models. We focus on stroke gestures, assuming that the user's input is expressed drawing two-dimensional paths on a screen or in mid-air.

In DEICTIC, simple and complex gestures are represented as expressions: literals represent the basic elements (ground terms) that can be combined for obtaining more complex paths. The temporal evolution is defined by the composition operators semantics. They allow creating complex gestures defining how a composite gesture evolves. In the following sections, we define both the ground terms and the composition operators.

4.1.1 Ground terms

The ground terms in DEICTIC are simple building blocks for defining strokes: points, lines and arcs. On the one hand, they guarantee a good level of expressiveness, since they allow modelling both linear and curve paths. On the other hand, they are a simplified representation of 2D paths that keep the language concise and understandable. We do not consider this as a limitation since the user's input has a coarser granularity.

Points They define the starting position of the stroke (e.g., the user touches the screen) simply specifying x and y coordinates in the plane. We represent a point with the notation $P(x, y)$. A stroke must always start with a point, and its coordinates define the current stroke position. Multi-stroke gestures have multiple point terms in their definition.

Lines They define a linear movement of a specified offset in the x and y axes, starting from the current stroke position. We represent a line with the notation $L(\Delta x, \Delta y)$.

Arcs They are quarters of a circle, starting from the current stroke position and finishing at the specified offset, following a clockwise or counter-clockwise direction. We represent clockwise arcs with the notation $A_{\circlearrowright}(\Delta x, \Delta y)$, and counter-clockwise arcs with $A_{\circlearrowleft}(\Delta x, \Delta y)$. If $|\Delta x| \neq |\Delta y|$ the arc is resized according to the offset ratio.

Figure 4.1 shows a graphical representation of the four primitives. The current position is represented by a filled circle, the stroke path with an arrow. The first defines a stroke starting at $(3, 3)$; the second defines a

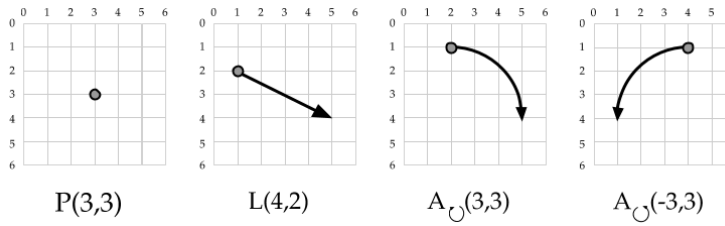


Figure 4.1: DEICTIC sample primitives.

line that, assuming that the current position is (1,2), moves 4 units on the x and 2 units on the y axis; the third defines an arc that, starting from the position (2,1) moves 3 unit on both axes, in a clockwise direction; the fourth defines an arc that, starting from the position (4,1) moves 3 units on both axes, in a counter-clockwise direction.

4.1.2 Composition operators

Starting from ground terms, we define complex expressions composing terms through a set of temporal operators, namely iterative, sequence, parallel and choice.

Iterative Operator E^* repeats an expression E an indefinite number of times. In order to maintain the compositional properties of the resulting gesture classifier, an iterated expression must either start with a *Point* ground term, or begin and finish in the same position. We will discuss such requirement more in deep in Section 4.2.2.

Sequence Operator The expression $E_1 + E_2 + \cdots + E_n$ defines a set of sub-strokes that must be executed in sequence, from left to right. Each expression considers as the current point the last position of the previous one.

Choice Operator The expression $E_1|E_2|\ldots|E_n$ defines a set of alternatives for performing a stroke. The entire expression is completed when one among the sub-strokes is executed.

In order to maintain the compositional properties of the resulting gesture classifier, the alternatives must either end their path at the same point or conclude the stroke. In the first case, the choice expression models a set of alternative paths for reaching the same point. In the second case, it either concludes the entire gesture or is followed in sequence by a point

primitive. We will discuss the technical motivation for this requirement in Section 4.2.4.

From an expressiveness point of view, if the alternatives must end at different points, it is still possible to model the stroke with DEICTIC, but the gesture continuation must be distributed for each alternative. We show an example in equation 4.1: the two lines in the choice end in different points. In order to satisfy the requirement, it is sufficient to distribute the sequence with the arc on both lines.

$$P(0,0) + (L(1,1)|L(-2,-2)) + A_{\circlearrowleft}(3,3) = \\ P(0,0) + (L(1,1) + A_{\circlearrowleft}(3,3)|L(-2,-2) + A_{\circlearrowleft}(3,3)) \quad (4.1)$$

Parallel Operator The expression $E_1 \times E_2 \times \cdots \times E_n$ represents a set of strokes that can be performed at the same time. It is useful for modelling e.g., multitouch gestures, where the user controls more than one stroke on the same screen. The operator does not fix any particular ordering between the two gestures but, in order to complete the recognition of the entire expression, both gestures must be performed.

4.1.3 Modelling examples

We complete the discussion of the description language showing a set of modelling examples. We refer to the stroke gestures of two well-known datasets, namely the 1$ gesture dataset [241] and the N$ gesture dataset [7,8]. They contain respectively different examples of single stroke and multiple stroke gestures. We used these datasets for evaluating the performance of our approach in Section 4.4.

The first example we discuss is depicted in Figure 4.2, a multi-stroke gesture representing a pitchfork (ψ). It consists of two different strokes, one describing a vertical line, intersected by a counter clock-wise half circle. For modelling the straight line, it is sufficient to specify the starting point and a line primitive going in the negative direction along the y-axis (s_1). The half circle consists of a sequence containing a starting point positioned on the left of the line, and two arc primitives: one moving down-right and one going up right (s_2). The final gesture should not take into account the stroke order, so we define a choice between s_1 followed by s_2 or s_2 by s_1. The selected points for modelling the gesture are arbitrary, the important part is the relative size of the primitives since in the recognition process we centre and normalise the size of both models and samples. Figure 4.2

shows graphically the resulting ideal definition and a set of real executions of the considered gesture.

$$\begin{aligned}\textbf{Pitchfork} : \psi =& s_1 + s_2 \mid s_2 + s_1\\ s_1 =& P(0,4) + L(0,-4)\\ s_2 =& P(-2,4) + A_{\circlearrowleft}(2,-2) + A_{\circlearrowleft}(2,2)\end{aligned}$$

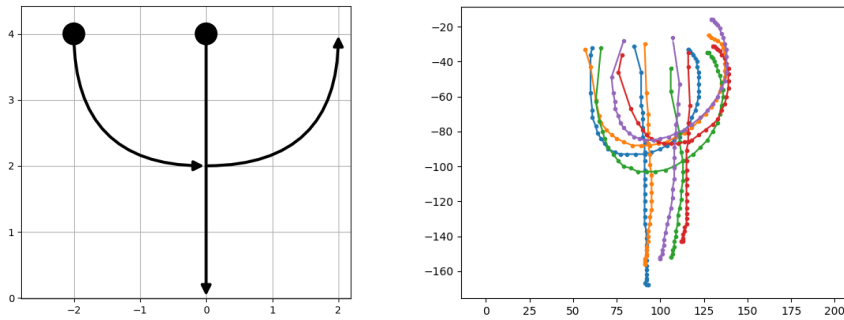


Figure 4.2: Modelling the pitchfork (ψ) gesture from the N$-dataset [7,8] with a DEICTIC expression. We show a graphical representation of the ideal model, together with 10 real gesture samples.

We apply the iterative operator for closed paths, for repeating a gesture an indefinite number of times, or when the stroke begins with a point. As an example, we consider the circle gesture in Figure 4.3 (◯), consisting of a single stroke containing four consecutive arcs. The repetition of the entire circle an indefinite number of times requires the user to lift the finger or the stylus from the screen and put it down again for repeating the circle, since it contains a point term ($P(0,0)$) at the beginning of the sequence. Instead, if we repeat only the arcs, we can model a repeated counter-clockwise movement on a closed circular path, that may be used e.g., as a rewind command (defined as ⏮in Figure 4.3) in a video player, mimicking the interaction with a real handle. It is possible to iterate the sequence of arcs in ⏮, since the gesture starts and ends at the same point (the origin).

Finally, we may allow our user to perform the circle gesture with one hand and the pitchfork with the other one, simply putting them in parallel ($\psi \parallel$ ◯). In this case, the temporal ordering is completely up to the user:

Circle: $\bigcirc = P(0,0) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3) + A_{\circlearrowleft}(-3,3)$

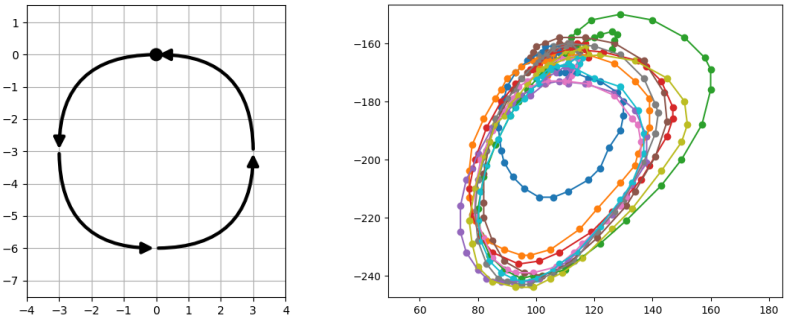


Figure 4.3: Modelling a circle gesture from the 1$-dataset [241] with a DEICTIC expression. We show a graphical representation of the ideal model, together with 10 real gesture samples.

she may start drawing the circle and then the pitchfork or vice-versa.

4.2 Building HMMs from the gesture definitions

In defining DEICTIC, our objective was to combine the declarative description advantages for UI developers together with the recognition accuracy and the robustness to input fluctuations of the state-of-the-art classification techniques. For deriving a classifier from a declarative gesture model, we use Hidden Markov Models (HMMs), provided their ability to describe stochastic temporal processes and their internal graph representation of states.

Differently from the other work in the literature, we do not use HMMs for recognizing the whole gesture when the user completes the stroke, but we use them for recognizing also the gesture sub-parts, defined through a declarative approach by the UI designer. This marks a difference from the other research on classification based on gesture primitives, which are usually segmented through an unsupervised learning step and hard to understand and use for UI designers.

In this section, we discuss how we exploit the definition of HMM (see

eq. 3.2) for recognising the ground terms (lines and arcs) introduced in the description language. After that, we describe an algorithm for creating an HMM associated with a composite term, starting from the ones that are associated with each operand. Finally, we discuss the algorithm that, starting from the definition of a gesture and a training set of lines and arcs, creates an HMM able to recognise complex gestures.

4.2.1 Ground terms

For defining a ground term in DEICTIC, we need to train an HMM able to recognise a particular segment (e.g., a straight line or an arc). This requires establishing the set of features that better describe the path and the set of values they can take. The selected features, according to their definition domains, fix the possible observations of the resulting HMM.

Besides the observation features, for training the HMM we need to define its number of states and the transition topology. Temporal processes like gestures and speech recognition are well suited for the left-to-right (or Bakis) topology [111, 135], which includes arcs between two states s_i and s_j only if $j \geq i$ or, equivalently, if the transition probability matrix A is upper triangular. In this method, we use a Bakis topology for the ground terms, but the composition works also if different topologies are used (e.g., ergodic) [67].

Once the topology and the observation domain are established, the last step is the HMM training for learning the parameters in A (the transition probability matrix) and B (the vector of the observation distributions). The learning phase is performed relying on the well-known algorithms described in 3.2, such as Viterbi learning [188] or Baum-Welch training [15] (we use the latter in this work, combined to the EM and MLE algorithms). The resulting HMM can be used for the recognition of a specific segment in different gesture definitions.

4.2.2 Iterative operator

The iterative operator allows recognising the same gesture an indefinite number of times. Starting from an HMM that defines a gesture g, the HMM for the gesture g^* can be defined adding a transition from all states connected with the ending state s_f to all states that are connected with the start state s_0, in order to create a loop in the HMM topology. The observation distribution vector remains the same.

Figure 4.4 depicts the HMM construction. The states filled in green ($F_0 \dots F_n$) belong to the forward star of s_0, which is the starting state of the HMM associated with the gesture g. Similarly, the backward star of the final state is represented by the states filled in red ($B_0 \dots B_m$). The HMM corresponding to the g^* gesture is obtained connecting each red state with each green state, allowing recognition loops. The original transition probability from a red state to the final state is distributed uniformly among all the transition to the green states and the final state. In figure 4.4, the s_0 is the starting state with its forward star $F_0 \dots F_n$; s_f is the final state with its backward star $B_0 \dots B_m$; the blue rectangle represents all the other states. The iterative operator is defined by the red arcs connecting each B_i with all F_j.

Algorithm 2 defines more in detail how to obtain an HMM for recognising the iteration of a gesture g^*, given the HMM for recognising g. In the algorithm definition, we distinguish the states and the transitions of an HMM using a superscript notation (e.g., s_0^g is the initial state of the g HMM, while s_f^i is the final state of the i HMM).

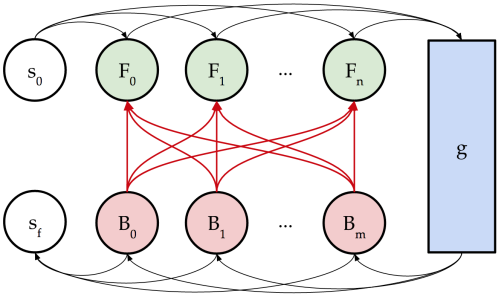


Figure 4.4: The topology of an HMM for the iterative operator.

ALGORITHM 2: Iterative Gesture HMM

Input: A HMM $g(S^g, V^g, A^g, B^g)$ recognising the g gesture
Output: A HMM $i(S^i, V^i, A^i, B^i)$ recognising the g^* gesture

$i \leftarrow \text{Clone}(g)$;
$\alpha \leftarrow \text{count}((s_0^g, u) \in A^g)$;

for $(u, s_f^g) \in A^g, (s_0^g, v) \in A^g$ **do** $A^i_{u,v} \leftarrow \frac{A^i_{u,f}}{\alpha+1}$;

for $(u, s_f^g) \in A^g$ **do** $A^i_{u,f} \leftarrow \frac{A^i_{u,f}}{\alpha+1}$;

return i

4.2.3 Sequence operator

The sequence operator allows recognising two gestures in the specified order. If we have an HMM for recognising the gesture g and one for recognising the gesture h, we define an HMM for $g+h$ simply connecting the end state of g with the start state of h from the topology point of view. However, since the starting and the final state in an HMM must be unique, those of the original models cannot be connected directly. Therefore, the idea is to bypass the final state of g and the starting state of h, adding a transition from the backward star of the former to the forward star of the latter. We delete the arcs in the backward star of s_f^g and in the forward star of s_0^h. Such operation is depicted in Figure 4.5: the red states represent the backward star of s_f^g $(B_0^g \dots B_m^g)$, while the green states represent the forward star of s_0^h $(F_0 \dots F_n)$. The sequence $g + h$ is defined connecting each red state with each green state. More in detail, in figure 4.5, s_f^g is the final state of the HMM associated with g, while $B_0^g \dots B_m^g$ is its backward star; s_0^h is the starting state of the HMM associated with h, while $F_0^h \dots F_n^h$ is its forward star; the blue rectangles represent all the other states of both HMMs. The sequence operator is defined by the red arcs connecting each B_i with all F_j.

In the general case, two ground terms may use two different feature sets. For instance, it is possible that a term considers the 2D position of the finger on a screen, while another one considers the direction angle. If the features used by g are different from those used by h, we support the composition creating a generic observation value for the composite HMM, which is a vector consisting of the union of all features considered by both g and h. Each state in the composed model must specify an emission probability for all features used in both g and h. Since a state of the composed HMM $g + h$ derives either from g or h, the observation distribution vector entry corresponding to each state must be "completed" with the features that were not considered in the original HMM. This can be achieved adding a uniform distribution across all the possible values of the features that are not considered in that state.

In the previous example, if g considers the 2D position of the finger on a screen, while the h considers the direction angle, we must complete the observation distribution vector in $g + h$ by adding the emission probability distributions for the direction angle in correspondence of the states that originally were defined in g, and for the 2D position in correspondence of the states that belonged to h. In both cases, the HMM should "ignore" the new features, therefore we add a uniform distribution across all the possible values for the angles of g and all the positions in h.

Algorithm 3 shows how to define $g + h$ given an HMM recognising g and one recognising h.

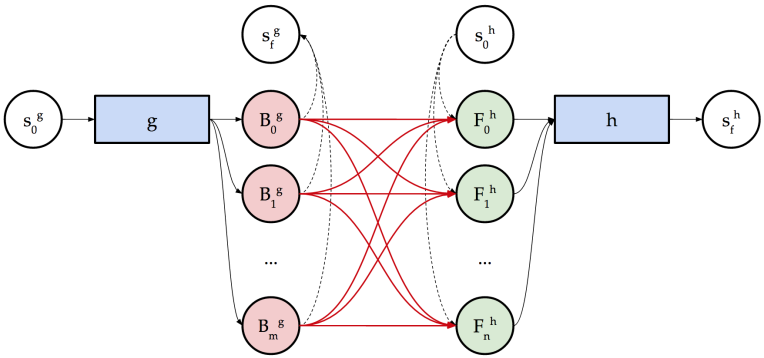


Figure 4.5: The topology of an HMM for the sequence operator.

ALGORITHM 3: Sequence Gesture HMM

Input: $g(S^g, V^g, A^g, B^g)$ and $h(S^h, V^h, A^h, B^h)$ recognising respectively the g and h gesture

Output: $s(S^s, V^s, A^s, B^s)$ recognising the $g + h$ gesture

$n \leftarrow |S^g| - 1$; $m \leftarrow |S^h| - 1$; ;
$s \leftarrow \text{EmptyHMM}(n + m)$;

$S^s \leftarrow (S^g \setminus s_f^g) \cup (S^h \setminus s_0^h)$; $V^s \leftarrow V^g \cup V^h$; $A^s \leftarrow 0_{n+m,n+m}$;
$B^s \leftarrow \text{CompleteDistributions}\ (B^g, B^h, V^s, V^g, V^h, n + m)$;

`/* copy the transitions from g */`
for $(u, v) \in A^g, v \neq s_f^g$ **do** $A^s_{u,v} \leftarrow A^g_{u,v}$;

`/* copy the transitions from h */`
for $(u, v) \in A^h, v \neq s_0^h$ **do** $A^s_{u+n,v+n} \leftarrow A^h_{u,v}$;

`/* connecting the ending states in g with the starting states in h */`
$\alpha \leftarrow \text{count}((s_0^h, u) \in A^h)$;

for (u, v) s.t. $A^g_{u,f} \neq 0$, $A^h_{0,v} \neq 0$ **do** $A^d_{u,v+n} \leftarrow \frac{A^g_{u,f}}{\alpha}$;
return S^s

function *CompleteDistributions* (B^g, B^h, V^s, V^g, V^h,n)
  $B \leftarrow \text{EmptyDistributionVector}(V^s, n)$;
  for $u \in A^g$, $v \in V^s$ **do**
    if $v \in V^g$ **then** $B_u[v] \leftarrow B^g_u[v]$;
    else $B_u[v] \leftarrow \text{unif}(v)$;
  end
  for $u \in A^h$, $v \in V^s$ **do**
    if $v \in V^h$ **then** $B_{u+n}[v] \leftarrow B^h_u[v]$;
    else $B_{u+n}[v] \leftarrow \text{unif}(v)$;
  end
  return B

4.2.4 Choice operator

The composition of two gestures g and h in choice allows the recognition of either g or h. In order to build the composite HMM for $g|h$, we put the original models in two separate recognition lines: there is no transition between the states originally belonging to g and the ones belonging to h. The only contact points are the starting states, which scatter the recognition lines, and the final state that collects them.

The schema is depicted in Figure 4.6: we connect the starting state with the forward state of both s_0^g and s_0^h. We set the transition likelihood to one-half of the value one in the original HMM, in order to have the two gestures in choice equally likely. The elements in the backward star of s_f^g and s_f^h are connected with the final state, with one-half of the original transition probability. We remove the arcs in the forward star of s_0^g and s_0^h together with those in the backward star of s_f^g and s_f^h. As happens for the sequence operator, the observation distribution vector for both recognition lines must be completed with respect to the features exploited by the other operand with a uniform distribution over all possible feature values. In this way, both recognition lines belonging to g and h ignore the values that were not originally in their own feature set. It is worth pointing out that composing gestures in choice decreases the probability assigned by the original HMMs to both g and h instances since it splits the recognition line for modelling the selection uncertainty. If different levels of choices are nested, this may degrade the recognition sensibly.

We can apply a simple optimisation in the composition when the choice is specified at the first level of the expression tree. In this case, an explicit composition of the choice operands is not needed: we simply select the operand that assigned the maximum probability to the considered sequence and we use it for both state labelling and likelihood computation. Since the complexity of the forward algorithm is quadratic on the number of hidden states, working on smaller HMMs decreases the recognition time.

Such optimisation is frequent in real-world gestural applications since many of them support the selection of a gestural command from a given set. In addition, this would allow also to include garbage models (arbitrary gestures that the application should ignore) without requiring an explicit modelling in an interactive recognition scenario.

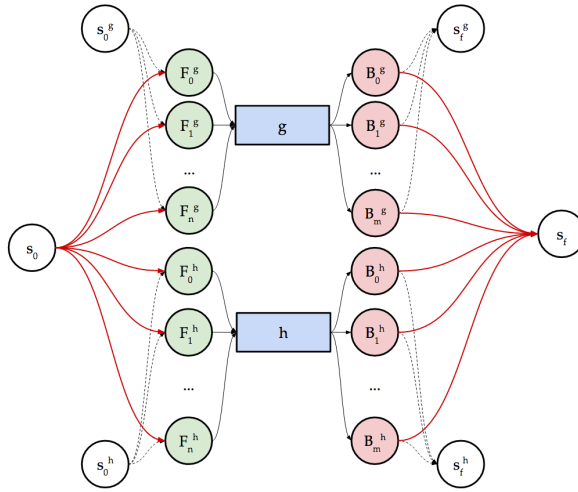


Figure 4.6: The topology of an HMM for a choice operator. The two original HMM are put in two separate recognition lines.

ALGORITHM 4: Choice Gesture HMM

Input: $g(S^g, V^g, A^g, B^g)$ and $h(S^h, V^h, A^h, B^h)$ recognising respectively the g and h gesture

Output: $c(S^c, V^c, A^c, B^c)$ recognising the $g + h$ gesture

$n \leftarrow |S^g| - 2$; $m \leftarrow |S^h| - 2$; ;
$c \leftarrow$ EmptyHMM$(n + m + 2)$;

$S^c \leftarrow (s_0^c \cup S^g \cup S^h \cup s_f^c) \setminus \{s_0^g, s_f^g, s_0^h, s_f^h\}$; ;
$V^c \leftarrow V^g \cup V^h$; $A^c \leftarrow 0_{n+m+2,n+m+2}$;
$B^s \leftarrow$ CompleteDistributions (B^g, B^h, V^s, V^g, V^h, $n + m$);

`/* create the recognition line for g */`
for $(u, v) \in S^g \setminus \{s_0^g, s_f^g\}$ **do** $A^c_{u,v} \leftarrow A^g_{u,v}$;

`/* create the recognition line for h */`
for $(u, v) \in S^h \setminus \{s_0^h, s_f^h\}$ **do** $A^c_{u+n,v+n} \leftarrow A^h_{u,v}$;

`/* connect the starting state with the` g `and` h `recognition lines */`
for (s_0^g, v) *s.t.* $A^g_{0,v} \neq 0$ **do** $A^c_{0,v} \leftarrow \frac{1}{2} \cdot A^g_{0,v}$;
for (s_0^h, v) *s.t.* $A^g_{0,v} \neq 0$ **do** $A^c_{0,v+n} \leftarrow \frac{1}{2} \cdot A^h_{0,v}$;
`/* connect the` g `and` h `recognition lines with the ending state */`
for (v, s_f^g) *s.t.* $A^g_{v,f} \neq 0$ **do** $A^c_{v,f} \leftarrow \frac{1}{2} \cdot A^g_{v,f}$;
for (v, s_f^h) *s.t.* $A^h_{v,f} \neq 0$ **do** $A^c_{v+n,f} \leftarrow \frac{1}{2} \cdot A^h_{v,f}$;
return S^c

4.2.5 Parallel operator

The parallel operator supports the simultaneous recognition of two gestures, performed independently. If we consider two gestures g and h, such independence requires that either g and h have a disjoint set of features or those in the intersection come from different data sources. For instance, the definition of two hand trajectories have clearly an intersection in their feature sets since at least the hand position is considered in both gestures definition. However, it is possible to compose them in parallel if we assign the first-hand trajectory to the right hand and the second to the left, or vice-versa.

Considering the creation of a composite HMM for $g \times h$, the independence leads to two important assumptions:

1. A transition event in g is independent from all transitions in h and vice-versa.

2. The observation of a value in V^g is independent from the observation of a value in V^h.

The composite HMM must represent all the possible combinations of states existing in g and h. Therefore, it contains a state for each pair (s^g, s^h). We include a transition between two states in $g \times h$ if it is valid in both g and h. Considering two states (s^g_i, s^h_j) and (s^g_x, s^h_y), we add a transition between them only if $A^g_{i,x} \neq 0$ and $A^h_{j,y} \neq 0$. The transition probability is $A^g_{i,x} \cdot A^h_{j,y}$, since the two events are independent. Finally, the observable values of $g \times h$ are the concatenation of those observable from g and h and they are independent from each other.

Algorithm 5 summarizes the procedure for building the HMM. Figure 4.7 shows the topology for the composition of two left-to-right HMMs, both consisting of 3 states. The state names in the parallel HMM correspond to the pair of states in the original HMM.

We can avoid using a quadratic number of states when the parallel composition occurs at the first level of the expression tree. In this case, we maintain separate the two HMMs, providing as state labels the pair (g_i, h_i) where g_i and h_i are respectively the state label assigned by g and by h to observations in the sequence. The sequence probability, since the two gestures are independent, is the product of the probability assigned by g and h.

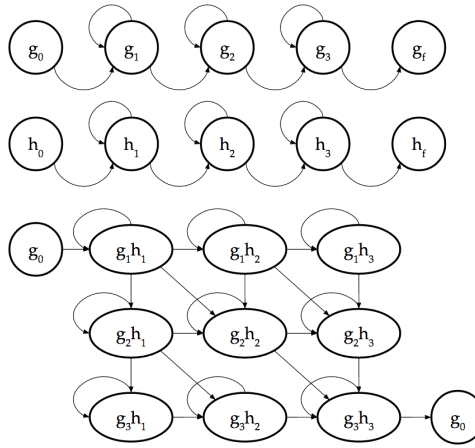


Figure 4.7: The topology of an HMM for a parallel operator, considering two Bakis terms

ALGORITHM 5: Parallel Gesture HMM

Input: $g(S^g, V^g, A^g, B^g)$ and $h(S^h, V^h, A^h, B^h)$ recognising respectively the g and h gesture

Output: $p(S^p, V^p, A^p, B^p)$ recognising the $g \parallel h$ gesture

$n \leftarrow |S^g| - 2$; $m \leftarrow |S^h| - 2$; ;
$c \leftarrow \text{EmptyHMM}(n \times m + 2)$;

$S^p \leftarrow (S^g \setminus \{s_0^g, s_f^g\}) \cap (S^h \setminus \{s_0^h, s_f^h\})$;
for $u \in (S^g \setminus \{s_0^g, s_f^g\}), v \in (S^h \setminus \{s_0^g, s_f^g\})$ **do** $B^p_{(u,v)} \leftarrow \text{indep}(B^g_u, B^h_v)$;
`/* setting transitions from the starting state */`
for $(s_0^g, v) \in A^g$, $(s_0^h, u) \in A^h$ **do** $A^p_{0,(u,v)} \leftarrow A^g_{0,u} \cdot A^h_{0,v}$;
`/* setting transitions to the final state */`
for $(v, s_f^g) \in A^g$, $(u, s_f^h) \in A^h$ **do** $A^p_{(u,v),f} \leftarrow A^g_{u,f} \cdot A^h_{v,f}$;
`/* parallel transitions in` g `and` h `*/`
for $(u, v) \in A^g$, $(x, y) \in A^h$ **do** $A^p_{(u,x),(v,y)} \leftarrow A^g_{u,v} \cdot A^h_{x,y}$;
return S^p

4.2.6 Creating an HMM from a gesture description

Having defined the algorithms for composing HMMs according to the gesture description language, we summarise here how we obtain an HMM from a gesture expression. We consider as classification features the x and y position of a stroke over time.

The first step is obtaining the normalised version of the expression: starting from the description, we calculate a bounding box for the gesture

definition, we centre it on the origin and we normalise the bounding box height and width in order to obtain a square enclosing the gesture definition.

The second step is training the HMMs for the ground terms. We assume having a training dataset for the left to right lines, one for clockwise arcs (starting from 0 to $\frac{\pi}{2}$) and one for counterclockwise arcs (starting from $\frac{\pi}{2}$ to 0). All samples are normalised in the same way we described for gesture descriptions.

For each line or arc term in the gesture expression, we apply to the corresponding training dataset a scaling, rotation and translation transform, in order to match the considered ground term in the normalised gesture definition. In addition, the raw training data is uniformly resampled in space (using the same approach described in [241]), which means that for each sample we use n equidistant samples. In this way, we are able to train a forward (Bakis) HMM for recognising each ground term in a complex gesture using the Baum-Welch [15] algorithm. We exploit the same (transformed) training dataset for each primitive. We establish the number of states for each HMM considering the relative length of the ground term in the whole gesture. The generation procedure defines a parameter representing the number of states per length unit s. Therefore, if l is the normalised length of the line or arc, the number of states of the generated HMM is $l \cdot s$.

After this step, we have an HMM trained for all ground terms included in the gesture description. In order to obtain the final HMM, we apply the composition algorithms described in the previous sections.

In the recognition phase, we first apply to each gesture a preprocessing step, consisting of the same transformations we applied to the gesture model definition (centring, scaling, translation, rotation and resampling). Then, the recognition probability is computed through the forward algorithm [187].

4.3 Developing gesture interfaces with DEICTIC

In this section, we discuss a set of properties of the composition algorithms, which are important for the development of UI requiring feedback and feedforward, as discussed in the Introduction.

The gesture definition language, together with the HMM generation procedures are suitable for building a generic gesture library, which enables interface designers and developers to rapidly create classifiers simply by

writing an expression. They do not need to provide any training example, since the primitive dataset may be shipped together with the library support. We publicly shared a Python reference implementation of the approach on GitHub [38] that allows writing gesture expressions similar to the ones reported in this dissertation for building classifiers through the operator overload. As we better detail in Section 4.4, the resulting HMMs are robust enough for recognising with high accuracy stroke gestures from different state-of-the-art datasets.

The composite HMMs support the identification of the ground terms states inside them, for estimating the completion level of a gesture. This enables showing which parts have been completed and the possible ways of completing the gesture. Indeed, we can identify which ground term is currently handling the values coming from the tracking devices from the most likely state in the HMM. It is easy to prove by induction on the composition algorithms that either a state belongs to a single ground term, or to an n-tuple of ground terms the user is allowed to perform at the same time. The latter case is related to the composition through a parallel operator. Therefore, if we are able to find the most likely state sequence that may produce the tracked feature values, we are also able to identify which ground terms have been performed or are currently progressing in the recognition process. This is an instance of the well-known decoding problem in the HMM theory, which can be solved using the Viterbi algorithm [72].

Another positive effect in developing gestural UIs derives from the mapping between states and ground terms. Given an HMM, we can predict the future distribution of its internal states through the forward algorithm [187]. Therefore, such mapping allows predicting the ground terms we will most likely encounter in the future. Such information is what the designer needs for creating effective *feedforward* systems.

Triangle: $\triangle = P(0,0) + L(-3,-4) + L(6,0) + L(-3,4)$

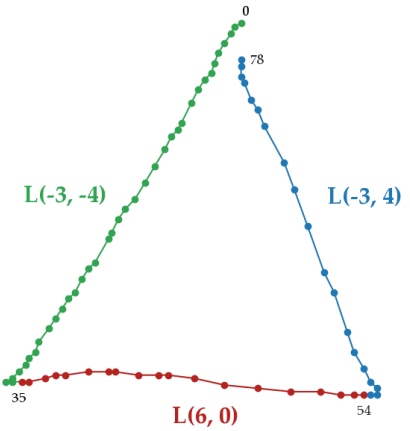


Figure 4.8: Ground term decomposition of a triangle gesture sample from the 1$-dataset. Each dot corresponds to a set of x, y coordinates of the sample point list. Points having a different colour corresponding to different ground terms.

Figure 4.8 shows the ground term segmentation obtained applying the Viterbi algorithm to a triangle gesture sample in the 1$-dataset, modelled with the reported DEICTIC expression. For each point of the sequence, the algorithm assigns the most likely state in the HMM. The point is green if such state belongs to the first line term, red if it belongs to the second line term, blue if it belongs to the third one. Such segmentation is promising for supporting intermediate feedback and feedforward in gestural UIs.

Our approach is currently limited in providing such information in the general case since the features we use for classifying gestures need a preprocessing phase for supporting a position and scale independent recognition, which cannot be executed without tracking the whole stroke. A solution may be using a representation of the stream that works both with incremental updates and with different sizes and positions of the same shape. We will investigate it in future work. However, such preprocessing phase may be avoided in case the stroke position and the scale are known, as happens in the interface discussed in section 6.1, where the user may execute gestures only inside the cells of a grid.

Our approach differs from other composition techniques on HMMs [111] since we neither retrain nor we fine-tune composite HMMs using samples

of the whole gesture. We train only ground terms using primitive samples, which may be shipped together with the recognition code, independently from the gesture set. This means that designers who define their own composite stroke gestures starting from predefined primitives are not required to collect a training dataset. The evaluation in Section 4.4 shows that this is possible without a sensible degradation of the recognition accuracy with respect to other state-of-the-art approaches.

The time spent for recognizing a gesture with a composite HMM depends on the number of ground terms it contains. The model definition allows associating a probability to a sequence of feature values, through the forward algorithm [187]. Its computation complexity is $O(N^2T)$, with N the number of hidden states and T the sequence length. While T does not depend on how we create an HMM, the composition has a clear impact on N. If we assume that all ground terms have a comparable number of states, N grows linearly with the composition operator count. A special case is represented by the parallel operator, which squares the number of states. Overall, this is a limitation in our approach: composing HMM requires increasing the number of states. Training an HMM directly on complete gesture samples (which we call ad-hoc HMMs in this method) usually allows finding a good trade-off between the recognition accuracy and the number of states, which is connected with the likelihood computation performance. However, it is worth pointing out that the ad-hoc solution requires a training set for each composed gesture, while DEICTIC requires training only the ground terms, and this speeds up the training phase.

4.4 Recognition accuracy evaluation

In this section, we show that DEICTIC achieves a recognition accuracy comparable with the other state-of-the-art approaches.

We implemented the algorithms described in Section 4.2 in Python, relying on an existing HMM library called Pomegranate [200] for evaluating the proposed compositional approach. We published the DEICTIC software package for creating and composing gesture recognisers on GitHub [38].

In order to recognise ground terms, we collected a set of 14 training example using a Leap Motion for i) left-to-right lines, ii) clockwise arcs (from $\frac{\pi}{2}$ to 0) and counter counter-clockwise arcs (from 0 to $\frac{\pi}{2}$). The samples are shipped with the library and we used them for training the ground terms HMMs as described in Section 4.2.6.

In order to evaluate the recognition rate of the HMM described with DEICTIC, we conducted three experiments, starting from two datasets

from the literature:

1. **Recognition of single-stroke gestures**. We considered the 1$-dataset introduced in [241]. It contains 330 repetitions of 16 single stroke gestures, shown in Figure 4.9, left part. We repeated the classification task using both DEICTIC and HMMs trained with samples from the dataset.

2. **Recognition of multiple-stroke gestures**. We considered the N$-dataset introduced in [7, 8]. It contains 600 repetitions of 14 multi-stroke gestures, shown in Figure 4.9, right part. We again repeated the recognition with both DEICTIC and HMMs trained with samples from the dataset.

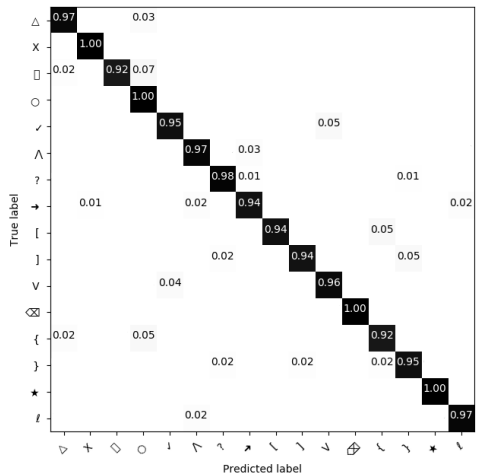
3. **Recognition of synthetic sequences**. In order to test the performance of the composition operators, we created test sequences composing random sequences from both datasets according to the operator semantics and we classified them with DEICTIC.

We modelled each gesture in both datasets using a DEICTIC expression and we generated the corresponding composite HMM. For the sake of brevity, we report in this section the test results, while we included the gesture models for single and multiple stroke gestures respectively in Table 4.4 and Table 4.5, 4.6. Before starting the classification task, we preprocessed each gesture instance centring it in the origin, normalising its size and resampling it to 20 samples per unit.

For evaluating the recognition accuracy, we fed each preprocessed instance to all composite HMMs. The HMM having the highest recognition probability (computed using the forward algorithm) assigned the label to the current sample. Table 4.1 shows the results. It is worth pointing out that none of the instances in both datasets was used for training a DEICTIC HMM. The recognition rate for both datasets is comparable with state-of-the-art classification algorithms applied to the same dataset in the literature. We refer to the rates reported in [231] for user-independent recognition, shown in Figure 4.10.

Considering the single-stroke gestures in the 1$-dataset, the 1$ algorithm [241] has the best performance (97.1%), followed by the P$ [231] (96.6%). DEICTIC (96.2%) is positioned immediately after, outperforming Protractor [131] (95.5%) and N$ [7, 8] (95.2%). Table 4.1 (above part) shows the DEICTIC confusion matrix on the single-stroke dataset. The classification performance is robust in all considered gestures, ranging from the maximum recognition rate (100%) for X, circle, delete and star, to the minimum (92%) for the rectangle and left brace.

1$ Dataset



N$ Dataset

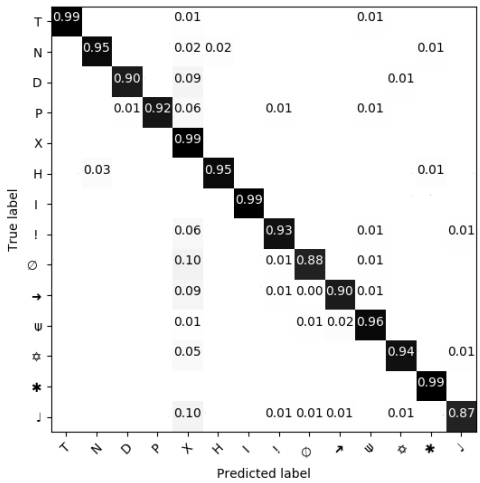


Table 4.1: Confusion matrix for the 1$-dollar [241] (above) and the the N$-dollar [7,8] (down) gesture dataset. Rows represent the ground-truth class, while the columns represent the class assigned by DEICTIC. Ground terms contained 6 states, working with gestures resampled to 20 samples per normalised unit. The 1$-dollar contains 330 samples for each one of the 16 gestures (5280 samples in total). The N$-dataset contains 600 samples for each one of the 14 multistroke gestures (8400 samples in total).

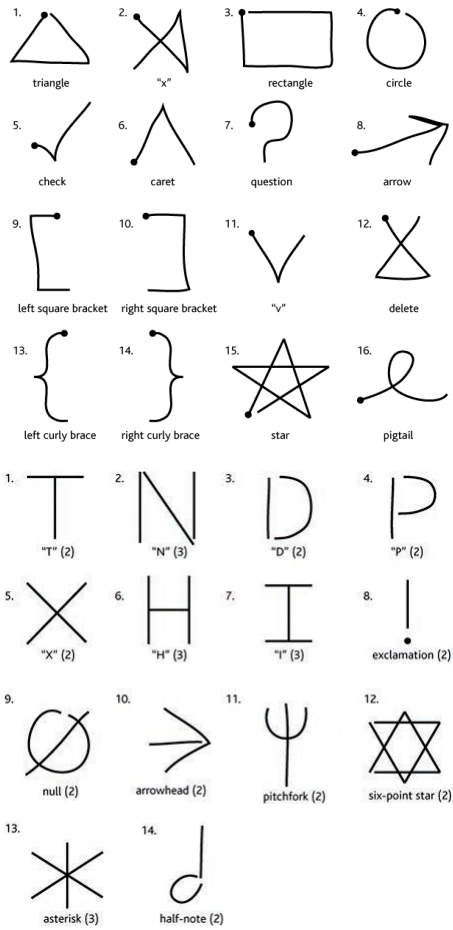


Figure 4.9: Gesture sets in the two evaluation datasets. The top part shows the single stroke gestures (figure taken from [241]), while the other part shows the multiple stroke gestures (figure taken from [7, 8])

Considering the multiple-stroke gestures in the N$-dataset, the best performance is reached by the P$ algorithm [231] (98.0%), followed by the N$ [7, 8] (96.4%). DEICTIC (94.0%) outperforms Dynamic Time Warping [231, 241] (93.4%) and approaches based on angular cosine [131] (91.3%) and Euclidean [121] (91.5%) distances. Table 4.1, down part shows the confusion matrix for the multi-stroke dataset, which shows again the robustness of the classification for each gesture, with a recognition rate

ranging from 99% for the X, to the 87% for the half-note. Some instances of all two-stroke gestures were confused with X.

Such consistency in the recognition rates shows that DEICTIC has a comparable accuracy with respect to state-of-the-art approaches in gesture recognition while maintaining the advantages of declarative and compositional modelling.

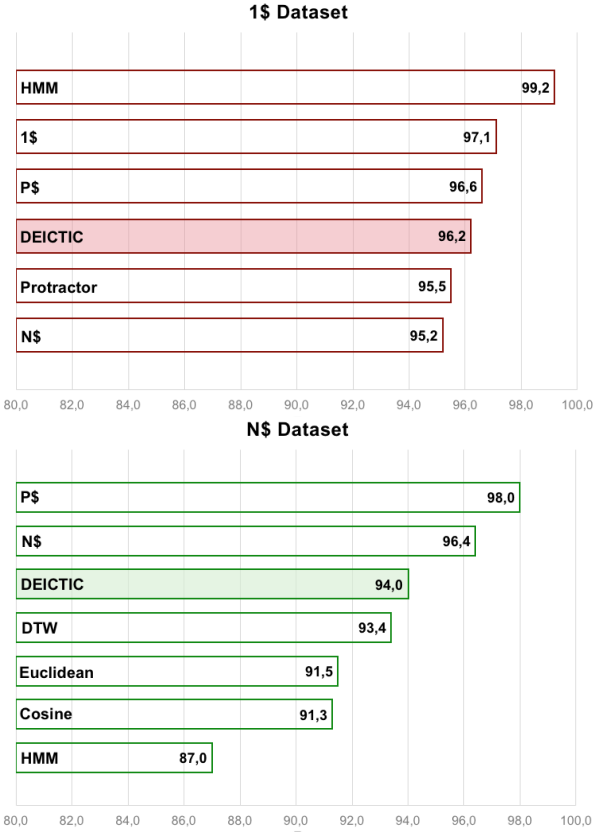


Figure 4.10: Accuracy of state-of-the-art approaches on the 1$-dataset [241] (top), and on the N$-dataset [7, 8] (bottom).

We added to the approaches from the literature in Figure 4.10 the recognition results obtained by HMM classification, without applying our compositional approach for building them (ad-hoc HMMs). This allows evaluating the impact on the recognition performance introduced by training only ground terms and applying the composition algorithms, against training HMMs on whole gesture samples (e.g., training with

samples of each side of a triangle gesture against providing the entire triangle stroke). More in detail, for each gesture we created a dedicated Bakis HMM, having the same number of states with respect to its DEICTIC counterpart. We run a ten-fold cross-validation for ensuring the relatability of the results.

Ad-hoc HMMs performed very well on single-stroke gestures, the recognition rate was higher than 99% and having a significant difference with DEICTIC ($t(15) = 3.98$, $p = .001$). Considering this dataset, the composition technique lowered the recognition level by about 3%.

We did not register the same performance on the N$-dataset: the mean recognition rate was about 87%. Ad-hoc HMMs had particular difficulties in distinguishing D strokes from P and many gestures are confused with the half-note and vice-versa. In contrast, DEICTIC had a lower recognition rate with respect to the single-stroke dataset (about 94%), but such decrease was not significant ($t(22) = 1.76$, $p = .09$). Our approach performed significantly better than ad-hoc HMMs ($t(13) = 2.25$, $p = .04$), increasing their recognition level by about 7%. The results of the experiments show again that the proposed composition technique maintain a good performance if compared with ad-hoc HMMs on single-stroke gestures, while it increases the recognition rate on multi-stroke ones.

It is worth pointing out the advantages introduced by DEICTIC specifically for the training phase. Ad-hoc HMMs requires complete gesture samples for learning the emission and transition distributions for all HMM states. In our experiment, it means using the 90% of the dataset for learning how to recognize the remaining 10% (297 samples for the 1$-dataset and 540 for the N$). If the gesture set changes, e.g., by adding a new gesture, we would need to collect other samples. In contrast, DEICTIC uses only 14 samples for each one of the ground terms, and they are the same for both the 1$ and the N$ datasets. No samples from these datasets were used in the training phase. This means that developers would not need to collect additional data for supporting different gestures.

In addition, DEICTIC always trains a constant number of states (6 in our experiments). Since the time complexity of the Baum-Welch [15] algorithm for training is $O(D \cdot T \cdot N^2)$ where D is the number of training samples, T the sample size and N the number of states in the HMM, ad-hoc HMMs training requires much more time. In our experiment, we passed from about 2 minutes for training DEICTIC to one hour for a single fold for ad-hoc HMMs.

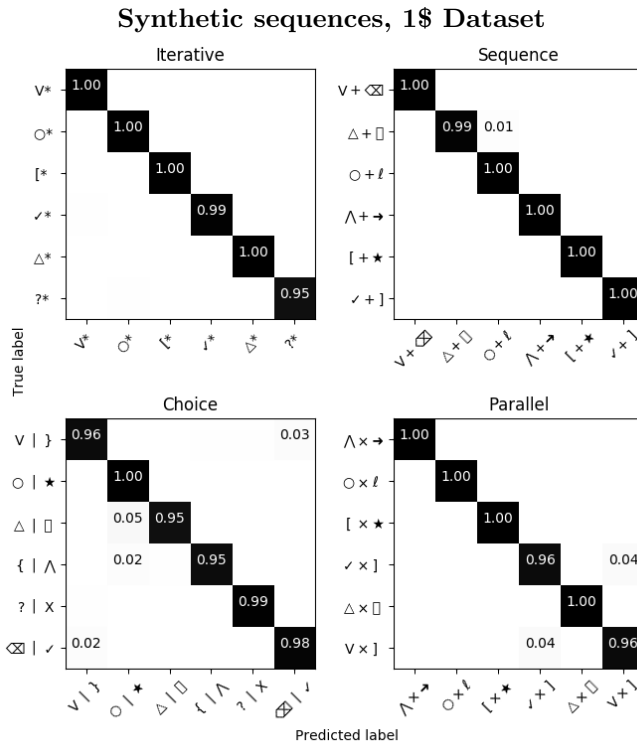


Table 4.2: Recognition of synthetic sequences

Besides the recognition of gestures in the dataset, we measured the performance with a set of synthetic sequences, produced in order to test all composition operators. We randomly selected the gesture pairs (or single gesture for the iteration) and, according to the operator semantics, we generated the composed sequence starting from the original data as follows:

- **Iterative**. Starting from a single sample, we randomly repeated it from 1 to 5 times.
- **Sequence**. We associated each sample of the first gesture to a randomly selected sample of the second one, without repetitions. The synthetic sequence is simply the concatenation of the two samples (in order).
- **Choice**. Starting from the gesture pair, for each synthetic sequence we randomly selected one sample either of the first or the second one, without repetitions.
- **Parallel**. Starting from the gesture pair, we selected two samples as

described for the sequence. We juxtaposed them shifting randomly up or down the rows of the second gesture and filling the blanks with random values. The latter operation guarantees that the gestures may start at different times with respect to each other.

After the sequence generation, we created the DEICTIC expression and we trained the corresponding HMMs. Table 4.2 shows the recognition performance on single-stroke composite gestures, while Table 4.3 reports the results on multi-stroke sequences. In both cases, the recognition performance is good.

In conclusion, the experiment highlighted different results obtained by our approach. First of all, DEICTIC has been able to recognise new gestures, significantly different from the samples included in the training set of each ground term. This is important for interface designers, which would be able to create gesture recognisers exploiting components, as they already do with UI widgets. Secondly, they would achieve a recognition rate comparable to other approaches in the literature.

Finally, considering the properties discussed in Section 4.3, it is possible to reconstruct the sequence of the most likely ground terms associated with a particular gestural input. Such information is not trivial when gestures are composed in choice or parallel since the designer would have the possibility to associate different feedback and feed-forward reactions to different ground terms. Such level of granularity is not supported by other methods used for recognising gestures.

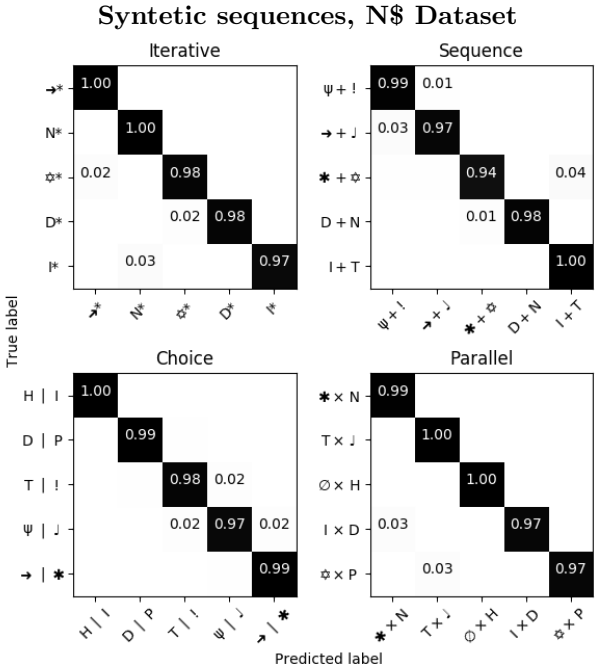


Table 4.3: Recognition of synthetic sequences

	Gesture	Expression
△	Triangle	$P(0,0) + L(-3,-4) + L(6,0) + L(-3,4)$
X	X	$P(0,0) + L(3,-3) + L(0,3) + L(-3,-3)$
□	Rectangle	$P(0,0) + L(0,-3) + L(4,0) + L(0,3) + L(-4,0)$
◯	Circle	$P(0,0) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3) + A_{\circlearrowleft}(-3,3)$
✓	Check	$P(0,0) + L(2,-2) + L(4,6)$
⋀	Caret	$P(0,0) + L(2,3) + L(2,-3)$
?	Question m.	$P(0,0) + A_{\circlearrowleft}(4,4) + A_{\circlearrowleft}(4,-4) + A_{\circlearrowleft}(-4,-4) + A_{\circlearrowleft}(-2,-2) + A_{\circlearrowleft}(2,-2)$
→	Arrow	$P(0,0) + L(6,4) + L(-4,0) + L(5,1) + L(-1,-4)$
[	L. bracket	$P(0,0) + L(-4,0) + L(0,-5) + L(4,0)$
]	R. bracket	$P(0,0) + L(4,0) + L(0,-5) + L(-4,0)$
V	V	$P(0,0) + L(2,-3) + L(2,3)$
⌫	Delete	$P(0,0) + L(2,-3) + L(-2,0) + L(2,3)$
{	L. brace	$P(0,0) + A_{\circlearrowleft}(-5,-5) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(5,-5)$
}	R. brace	$P(0,0) + A_{\circlearrowleft}(5,-5) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(-5,-5)$
⋆	Star	$P(0,0) + L(2,5) + L(2,-5) + L(-5,3) + L(6,0) + L(-5,-3)$
ℓ	Pigtail	$P(0,0) + A_{\circlearrowleft}(3,3) + A_{\circlearrowleft}(-1,1) + A_{\circlearrowleft}(-1,-1) + A_{\circlearrowleft}(3,-3)$

Table 4.4: 1$-dataset Gesture Modelling DEICTIC

	Gesture	**Expression**
T	T	$P(-2,0)+L(4,0)+P(0,0)+L(0,-4)\|$
		$P(0,0)+L(0,-4)+P(-2,0)+L(4,0)$
N	N	$P(0,4)+L(0,-4)+P(0,4)+L(4,-4)+P(4,4)+L(0,-4)$
D	D	$P(0,6)+L(0,-6)+P(0,6)+L(2,0)+A_{\circlearrowleft}(3,-3)+A_{\circlearrowleft}(-3,-3)+L(-2,0)$
P	P	$P(0,8)+L(0,-8)+P(0,8)+L(2,0)+A_{\circlearrowleft}(2,-2)+A_{\circlearrowleft}(-2,-2)+L(-2,0)$
X	X	$P(0,0)+L(4,4)+P(0,4)+L(4,-4)\|$
		$P(4,4)+L(-4,-4)+P(0,4)+L(4,-4)\|$
		$P(0,4)+L(4,-4)+P(0,0)+L(4,4\|)$
		$P(0,4)+L(4,-4)+L(4,4)+L(-4,-4)$
H	H	$P(0,4)+L(0,-4)+P(0,2)+L(4,0)+P(4,4)+L(0,-4)\|$
		$P(0,4)+L(0,-4)+P(4,4)+L(0,-4)+P(0,2)+L(4,0)$
I	I	$P(0,4)+L(4,0)+P(2,4)+L(0,-4)+P(0,0)+L(4,0)\|$
		$P(2,4)+L(0,-4)+P(0,0)+L(4,0)+P(0,4)+L(4,0)\|$
		$P(0,4)+L(4,0)+P(0,0)+L(4,0)+P(2,0)+L(0,4)\|$
		$P(2,4)+L(0,-4)+P(0,4)+L(4,0)+P(0,0)+L(4,0)$
!	Exc. point	$P(0,4)+L(0,-3)+P(0,1)+L(0,-1)$
∅	Null	$P(0,0)+A_{\circlearrowleft}(-3,-3)+A_{\circlearrowleft}(3,-3)+A_{\circlearrowleft}(3,3)+A_{\circlearrowleft}(-3,3)+P(4,1)+L(-8,-8)\|$
		$P(0,0)+A_{\circlearrowleft}(-3,-3)+A_{\circlearrowleft}(3,-3)+A_{\circlearrowleft}(3,3)+A_{\circlearrowleft}(-3,3)+P(-4,-7)+L(8,8)$

Table 4.5: N$-dataset Gesture Modelling DEICTIC

	Gesture	Expression
→	Arrow	$P(0,0) + L(6,0) + P(4,2) + L(2,-2) + L(-2,-2)\|$
		$P(4,2) + L(2,-2) + L(-2,-2) + P(0,0) + L(6,0)$
ψ	Pitchfork	$P(-2,4) + A_{\circlearrowleft}(2,-2) + A_{\circlearrowleft}(2,2) + P(0,4) + L(0,-4)$
		$P(0,4) + L(0,-4) + P(-2,4) + A_{\circlearrowleft}(2,-2) + A_{\circlearrowleft}(2,2)$
✡	Six point star	$P(0,0) + L(-2,-4) + L(4,0) + L(-2,4) + P(-2,-1) + L(4,0) + L(-2,-4) + L(-2,4)\|$
		$P(-2,-1) + L(4,0) + L(-2,-4) + L(-2,4) + P(0,0) + L(-2,-4) + L(4,0) + L(-2,4)\|$
		$P(-2,-2) + L(2,4) + L(2,-4) + L(-4,0) + P(-2,1) + L(4,0) + L(-2,-4) + L(-2,4)\|$
		$P(-2,1) + L(4,0) + L(-2,-4) + L(-2,4) + P(-2,-2) + L(2,4) + L(2,-4) + L(-4,0)\|$
		$P(-2,-2) + L(2,4) + L(2,-4) + L(-4,0) + P(-2,1) + L(2,-4) + L(2,4) + L(-4,0)\|$
		$P(-2,1) + L(2,-4) + L(2,4) + L(-4,0) + L(-2,-2) + L(2,4) + L(2,-4) + L(-4,0)\|$
		$P(0,0) + L(-2,-4) + L(4,0) + L(-2,4) + P(-2,-1) + L(2,-4) + L(2,4) + L(-4,0)\|$
		$P(-2,-1) + L(2,-4) + L(2,4) + L(-4,0) + P(0,0) + L(-2,-4) + L(4,0) + L(-2,4)$
✳	Asterisk	$P(4,4) + L(-4,-4)) + P(0,4) + L(4,-4) + P(2,4) + L(0,-4)$
𝅗𝅥	Half note	$P(0,0) + A_{\circlearrowleft}(-3,3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3) + P(2,16) + L(0,-20)$
		$P(2,16) + L(0,-20) + P(0,0) + A_{\circlearrowleft}(-3,3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3)$
		$P(0,0) + A_{\circlearrowleft}(-3,3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3) + P(2,-4) + L(0,20)$
		$P(2,-4) + L(0,20) + P(0,0) + A_{\circlearrowleft}(-3,3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3)$

Table 4.6: N$-dataset Gesture Modelling DEICTIC

Chapter 5

G-Gene

As illustrated in 4.3, DEICTIC requires a normalization step for dealing with different scalings of the same shape. Therefore, in the general case, DEICTIC needs to preprocess the whole input for classifying gestures. In this chapter, we introduce G-Gene, an online stroke gesture recognizer scale independent. The proposed method combines a simple gesture modelling technique, designed for supporting feedback and feedforward mechanisms, and an HMM classifier. The generated HMM supports online recognition, without requiring both the whole input and a training phase. It relieves the developer from the need to collect any gesture sample. It is worth pointing out that the contribution of this work is not on i) the recognition accuracy, ii) the gesture modelling itself or iii) the robustness in continuous gesture recognition, but on establishing a compromise between accuracy and providing the information needed for feedback and feedforward systems through the recognition support.

We start with the section 5.1, where we introduce the definition of G-Gene, illustrating the procedure adopted to create a new profile HMM, highlighting both benefits and limitations in our work. After that, in section 5.2, we analyse the accuracy rate obtained in experimental tests. This chapter is an extended version of the work in [39].

5.1 Method

In this section, we discuss the methodology we applied for creating a profile HMM, using as input the declarative gesture modelling expression defined in Section 4.1. The resulting composite HMM has three main properties: i) it does not require any training sample for recognizing gestures, ii) it does

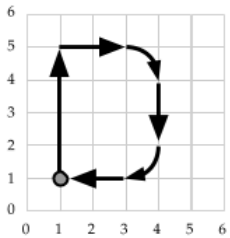
not use any global feature in the gesture classification and iii) it allows recognizing gesture sub-parts, supporting the development of *feedback* and *feedforward* systems.

5.1.1 Building the G-Gene profiles

Starting from an ideal gesture expression model defined through the modelling language discussed in section 4.1, we derive a set of character sequences representing the G-Gene target profiles. The characters define a label for each stroke gesture sub-part, which distinguishes them from each other according to local features. In particular, we consider:

- The direction of a linear movement transformed to the closest one in the compass (north, north-west, west etc.). We use the following characters ↑↗→↘↓↙←↖ for representing these directions.
- The rotation direction of a curve movement, either clockwise (denoted by the ↻ character) or counterclockwise (denoted by ↺).
- The boundary point between two sub-parts of a stroke, denoted by the · character.

In Figure 5.1, we show an example profile for a D gesture. The stroke starts at the bottom left corner of the shape, then it goes up (↑), it changes the subpart (·, which is repeated between each part), it moves right (→), it makes a clockwise arc movement (↻), it moves down (↓), it makes clockwise arc (↻), and finally it moves left (←). We add a boundary character at the beginning and at the end of the string for obtaining the final profile.



· ↑ · → · ↻ · ↓ · ↻ · ← ·

Figure 5.1: Sample G-Gene target profile for a D gesture. The stroke starts from the filled dot. We add a boundary character between each gesture sub-part (·).

We derive the target profiles with a semi-automatic procedure, consisting of two phases. In the first phase, we associate to each ground term in the modelling language its corresponding profile character as follows:

- For each term of point type $P(x, y)$, we associate a boundary point $(\cdot)$;
- For each term of type line $L(\Delta x, \Delta y)$, we associate the closest direction in the compass, minimising the difference between the vector angle $v(\Delta x, \Delta y)$ and the eight considered directions;
- For each term of type arc $A_{\circlearrowleft}(\Delta x, \Delta y)$ or $A_{\circlearrowright}(\Delta x, \Delta y)$, we associate the rotation direction (either $\circlearrowleft$ or $\circlearrowright$);
- We add a boundary character $(\cdot)$ for each sequence operator (the $+$) in the expression.
- We add a boundary character $(\cdot)$ at the end of the expression.

The result is a single profile string that represents the ideal stroke performance in G-Gene. In the second phase, we generate a set of mutated profile strings from the ideal profile, replacing each line segment with the two "siblings" compass directions, rotating it $\pm 45^o$ around its end point. The designer selects the changes that preserve the gesture shape, according to her own needs. It is worth pointing out that the profiles are scale independent, since their definition relies only on the ideal movement direction and not on its size, but not rotation independent.

The process is depicted in Figure 5.2 for an N gesture. It contains three line segments, highlighted in the second level of the tree. For each of them, the G-Gene profile generation procedure creates the mutation selecting the compass direction by rotating the line around the end point. For instance, considering the first line segment in Figure 5.2 that points north, the tool generates a north-east and a north-west line segment. The designer may accept or reject the resulting shapes. We highlight the former case in green and the latter in red in Figure 5.2. The resulting strokes that are acceptable as G-Gene profiles are depicted at the bottom. They are obtained by combining the selected options for all the line segments (in blue the segments coming from the ideal profile, in green the segments accepted during the selection process). The process may generate an exponential number of profiles, but in practice, many intermediate changes are always rejected (we used no more than 4 mutations for all gestures discussed in Section 5.2.1).

After completing the two phases, each gesture expression is associated with a set of profile strings, which we use for gesture recognition.

5.1.2 Building the G-Gene representation of the stroke input

At runtime, we track the 2D position of the stroke and we incrementally build its G-Gene representation. We decompose a trajectory in subparts, corresponding to the characters we introduced in Section 5.1.1. We exploit

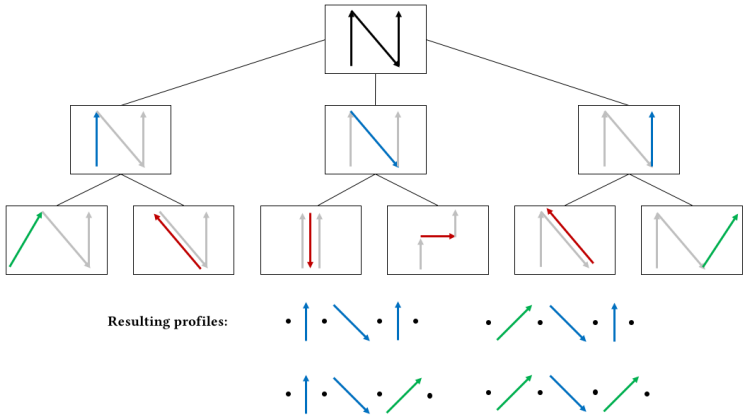


Figure 5.2: Generation of the G-Gene profiles from an N stroke expression. For each line primitive, we show the shape mutation considering the direction at $\pm 45^o$ around the line end point. The designer selects the ones that maintain a reasonable similarity with the desired shape (in green) and she rejects those that are too different. The bottom part of the figure shows the generated profiles.

local information for supporting online recognition.

The first and the last character in the input string representation is a boundary. We generate them respectively when the stroke starts and when it ends. While the user is performing the gesture, we basically distinguish between lines and arcs through a collinearity test. We generate a character each time the collinearity test changes its result.

More formally, we receive from the input device a stream of points, we denote as x_t and the y_t the coordinates of the stroke position at the current time t. In the collinearity test, we start from the first position in the stream that was aligned with the last input sample, or from the beginning of the stream. We need at least three points for the test, so if t is the current stream and s the starting position we have that $0 \leq s \leq t-2$. The test is defined in Equation 5.1, where $\langle \overrightarrow{v}, \overrightarrow{w} \rangle$ is the inner product of $\overrightarrow{v}$ and $\overrightarrow{w}$ and $\| \overrightarrow{v} \|$ is the l_2 norm. We use an error threshold (ϵ) that guarantees robustness to nearly-collinear points, as described in [198]. We apply a Kalman iterative smoother to the input stream for reducing noise while preserving the stroke shape and the movement properties.

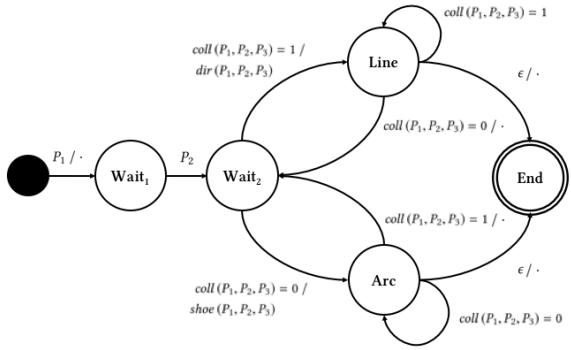


Figure 5.3: The G-Gene stroke string generation FSM. It receives the stroke position sequence as input (P), together with the start and end event. The transition labels define the firing condition and the generated character (if any), separated by a slash.

$$\begin{aligned} &coll\colon (\langle \overrightarrow{v}, \overrightarrow{w} \rangle^2 \leq \epsilon \cdot \| \overrightarrow{v} \|^2 \cdot \| \overrightarrow{w} \|^2) \rightarrow \{0, 1\} \\ &\overrightarrow{v} = (x_b - x_a, y_b - y_a), \quad \overrightarrow{w} = (x_c - x_b, y_c, -y_b), \qquad \forall \quad a, b, c \in [s, t] \end{aligned} \tag{5.1}$$

Such collinearity test drives the stroke segmentation and labelling algorithm, represented in Figure 5.3 as a Finite State Machine (FSM). It starts buffering the first three samples, which is the minimum number for starting the labelling process ($Wait_1$ and $Wait_2$). After that, it checks if the samples are aligned or not. The process forks according to the value obtained by applying the Equation 5.1.

- If *true*, the FSM moves to the *Line* state. The transition generates a character minimising the difference between the vector angle defined by the current line segment and the eight compass directions. In the FSM this corresponds to the function $dir\colon (P_1, P_2, P_3) \rightarrow \lambda \in \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow\}$, which assigns one of the direction characters to the three aligned points. The FSM stays in that state until the collinearity of the segment holds.
- If *false*, the FSM moves to the *Arc* state. We compute the area of the polygon defined by three buffered points (P_1, P_2, P_3) using the Shoelace formula[1], If the area is greater than zero, then the segment describes a clockwise trajectory, or counter-clockwise movement otherwise. We generate respectively a ↻ or a ↺ character in the former

[1] `https://en.wikipedia.org/wiki/Shoelace_formula`

and in the latter case. In the FSM, we represent such generation with the function $shoe$: $(P_1, P_2, P_3) \rightarrow k \in \{\circlearrowleft, \circlearrowright\}$. The FSM stays in *Arc* state until a collinear segment is spotted.

From both states the FSM goes back to the $Wait_2$ state when the collinearity test changes its value for the current segment, generating a boundary $(\cdot)$ character. Finally, it goes to the *End* state when the stroke ends. The input representation is scale-independent (it relies on the stroke direction and not on its size), but not rotation independent.

As we illustrated in Section 3.2 Hidden Markov Models (HMMs) have been used in the literature for solving and modelling the correlations between adjacent symbols. In G-Gene, we rely on profile-HMMs for determining the similarity between two G-Gene profiles; therefore, our approach solves the gesture recognition problem by finding the most similar gesture string in the reference G-Gene profiles for a given input.

5.1.3 Limitations

G-Gene uses local features and groups segment samples for splitting a trajectory in real-time, representing it as a sequence of characters, corresponding to gesture sub-parts. On the one hand, this allows us to create profile-HMMs for the recognition, which provide information on the gesture sub-components. On the other hand, this set some limitations.

First of all, using local features makes the recognition process less robust to the input noise, since characters are generated on the fly according to a limited number of samples. Many classification techniques we reported in the related work section use global features for achieving a higher accuracy.

In addition, the local features we use do not maintain information about the length of a segment. This makes gestures which differ from the length of their primitives indistinguishable with our method. An example may be trying to distinguish a V from a checkmark (✓) stroke. We can describe both of them with the sequence $\cdot \searrow \cdot \nearrow$, but we would need to encode the length of the segment in $\searrow$ for telling them apart.

Finally, the robustness to noise may become a limit when we have similar profiles: it may be hard for designers to control them given the generation procedure. As a consequence, we may need to balance the number of profiles used for each gesture and the accuracy of the recognition.

5.2 Evaluation

In this section we illustrate the experimental tests conducted to evaluate both accuracy and performance of our methods. In the section 5.2.1, we

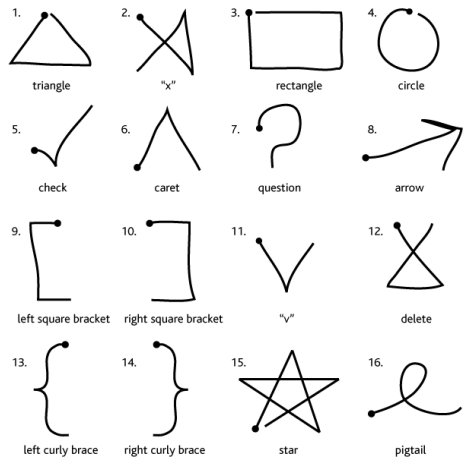


Figure 5.4: The stroke gestures in the 1$-dataset (figure taken from [241])

evaluated the G-Gene recognition rate with the dataset in [241].

5.2.1 Accuracy Evaluation

We implemented the proposed method in Python, on top of the Pomegranate HMM library [200]. In this section we evaluate the method accuracy, testing it on the 1$-dataset [241]. That dataset contains 330 repetitions of 16 single stroke gestures, performed by 11 different users on a touch surface. Figure 5.4 shows the set of gestures in the dataset.

We modelled the stroke gestures using the profile sequences described in section 5.1 experiment. Considering the limitation discussed in Section 5.1.3 we excluded the checkmark (✓) that is indistinguishable from the stroke V.

We simulated an online dispatching, feeding the profile HMM with a single frame at a time (each stroke point in the dataset has an associated timestamp). We evaluated the accuracy comparing the ground-truth label with the profile having the highest probability. Figure 5.5 shows the resulting confusion matrix. The overall accuracy of the proposed approach is 88.2%, which is good enough for building interface prototypes: as discussed in [224], the accuracy of a good recogniser is expected to fall between 88% and 98%. However, it is below the results reported in the literature for the same dataset: the 1$ algorithm [241] has the best performance (97.1%), followed by P$ [231] (96.6%), Protractor [131] (95.5%) and N$ [8] (95.2%). It is worth pointing out that all these approaches use an offline recognition technique that, as we will discuss in

Gesture	Ideal Sequences
△	· ↙ · → · ↖ ·
X	· ↘ · ↑ · ↙ ·
□	· ↓ · → · ↑ · ← ·
◯	· ↺ · ↺ · ↺ · ↺ ·
∧	· ↗ · ↘ ·, · ↑ · ↘ ·, · ↗ · ↓ ·, · ↑ · ↓ ·
?	· ↺ · ↺ ·, · ↺ · ↺ · ↺ ·
↗	· ↗ · ↙ · → · ↓ ·, · ↗ · ← · → · ↓ ·, · ↗ · ← · → · ↙ ·
[	· ← · ↓ · → ·
]	· → · ↓ · ← ·
V	· ↘ · ↑ ·, · ↘ · ↗ ·, · ↓ · ↗ ·, · ↓ · ↑ ·
⌫	· ↘ · ← · ↗ ·
}	· ↺ · ↺ · ↺ · ↺ · ↺ ·, · ↺ · ↺ · ↺ · ↺ · ↺ · ↺ ·
{	· ↺ · ↺ · ↺ · ↺ · ↺ ·, · ↺ · ↺ · ↺ · ↺ · ↺ · ↺ ·
✡	· ↗ · ↘ · ↖ · → · ↙ ·
ℓ	· ↗ · ↺ · ↙ ·, · ↗ · ↺ ·

Table 5.1: The profile sequences used to describe each gesture in the 1$-dataset.

Section 6.3, creates more difficulties for developers in creating feedback and feedforward systems.

We identified two groups of gestures in the confusion matrix. The first includes those gestures having a very high recognition rate, ranging from the maximum 99% of X, the rectangle and delete, to the 90% for the V stroke (whose instances were sometimes confused with the delete). The second group consists of gestures having profiles similar to other ones in the set, which are much more affected by the lack of precision induced by local features. The recognition rate, in this case, ranges from 85% for the left square bracket, to 70%, which is the minimum reached for the question mark. Some examples of gestures in this group are the pigtail, whose instances are confused with the arrow, and the circle that is confused with the left square or curly bracket. Both of them have a 13% of samples wrongly labelled.

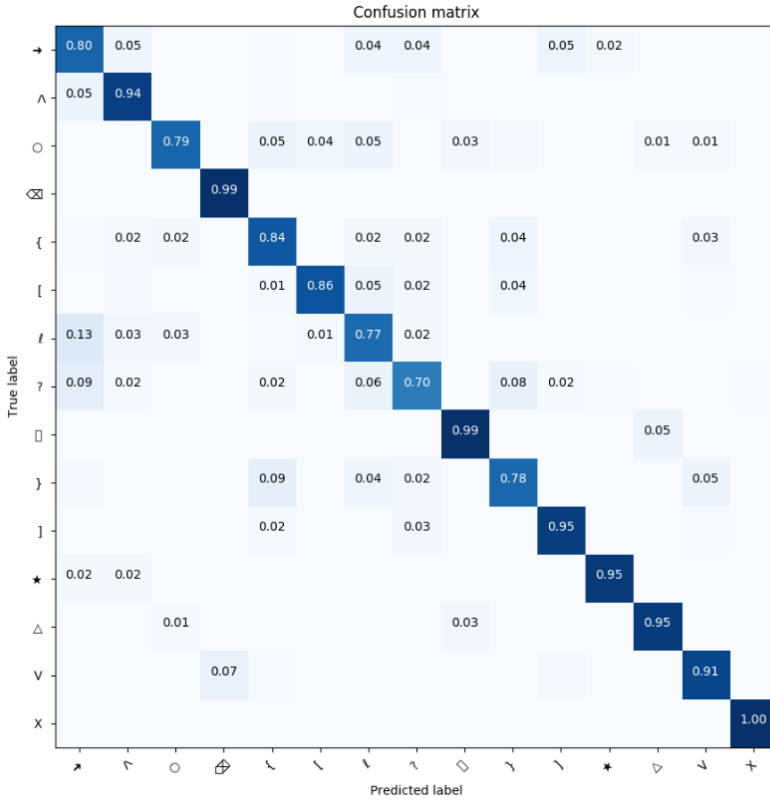


Figure 5.5: G-Gene confusion matrix on the 1$-dollar [241] gesture dataset. Rows represent the ground-truth class, while the columns represent the class assigned by G-Gene. The dataset contains 330 samples for each one of the 15 gestures (4905 samples in total).

Chapter 6

Developer Evaluation

In this chapter, we detail the results of a user study for assessing how the two proposed methods assist the UI developers in creating gestural interactions. Firstly, we compared DEICTIC recognition support against heuristic and machine-learning approaches. After that, we assessed the G-Gene support in creating a well-known interface that requires feedback and feedforward information against one of the available toolkit discussed in Section 2.1, which provides information on the final gesture label. In both cases, our hypothesis is that the modelling effort introduced by our gesture recognition libraries is balanced by the advantages of receiving intermediate information.

We start describing the tasks and the partecipants recruited for the test in section 6.1. Then, we discuss the developer evaluation of DEICTIC in section 6.2 and finally we illustrate the results obtained with G-Gene in section 6.3.

6.1 Environment

In this section we detail the environment implemented in order to evaluate the performance of DEICTIC and G-Gene in creating gesture guiding systems. Then we describe the participant sample recruited for the tests (section 6.1.1), we illustrate the development tasks (section 6.1.2) and we discuss the results (section 6.1.3).

6.1.1 Partecipants

We recruited a group of 17 developers, 14 males and 3 females for both experiments. Their education level ranged from the High School Degree (5), Bachelor Degree (3), Master Degree (5) to PhD (3). Most of them have a development experience between 1 and 5 years (9), 5 between 6 and 10 years and 3 more than 10 years. All participants are fluent with Object-Oriented languages such as Java, C++ or C#, about half of them know Javascript, PHP and PL/SQL, while they are less familiar with the other ones reported in Figure 6.1, upper part. We asked them to self-evaluate their experience with a group of development tasks relevant for the evaluation. They have a mid-level experience in web and UI development, while they have a low experience with gesture interface development and in using machine learning techniques. Figure 6.1 shows the details on the participants' programming skills.

The recruited participants are a relevant sample considering the development of a specific UI whose design was already established by another team during the development process (e.g., by graphical or user experience experts). We simulate the existence of different development libraries, which offer gesture recognition capabilities. In both tests, participants have to decide which one better suits the UI requirements.

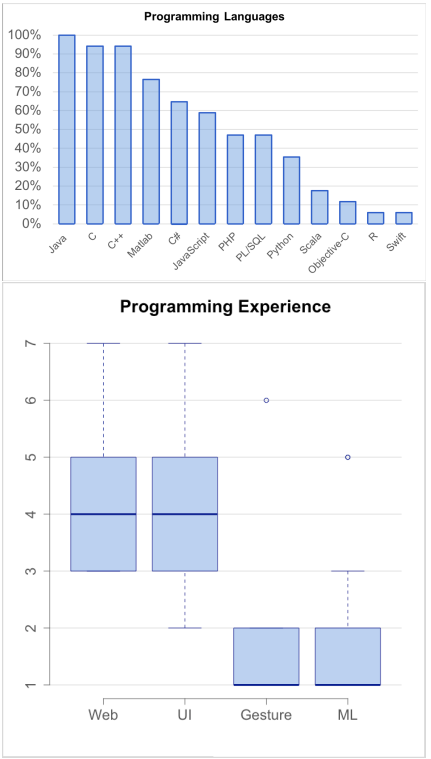


Figure 6.1: Participants' experience with programming languages and development task.

6.1.2 Procedure

We conducted two different experiments, each one to measure the support provided, respectively, by DEICTIC and G-Gene. We start describing the development tasks required in DEICTIC; after that, we characterize the tasks which make up the evaluation of G-Gene.

DEICTIC Development Tasks The evaluation of DEICTIC consisted in the development of a simple gestural interface for managing a 3×4 grid (see Figure 6.2, top part). In each cell, the user can put a monster, a treasure cell, or she can leave it empty. Three simple stroke gestures support the of control the content of each cell: a monster appears by drawing a triangle, a treasure box by drawing a square, while an X stroke clears the cell content. The participants were asked to create two variants of the same application, which differ with respect to the guidance to be

provided as feedback while performing these gestures. This corresponds to having two hypothetical alternative versions requested by the UI design team.

In the first variant, the application uses a *Line Feedback* (see Figure 6.2), which simply draws a red line as the user moves the finger on the screen. In this configuration, the information required for drawing the line is simply the sequence of the stroke positions over time. Since there is no need for intermediate gesture recognition, the classification support is required only at the end of the stroke. We inserted this task in the test for evaluating the difficulties in modelling and recognising gestures through a given technique.

The alternative feedback design is *OctoPocus* [14], which shows a dynamic guidance while the user performs the stroke. Figure 6.2, bottom part, shows the feedback for a triangle gesture. When the stroke starts, it displays the possible gesture completions using different colours. While the user performs the stroke, it updates the representation by encoding in the colour opacity the predicted likelihood (the more opaque the colour, the more the system is confident that the user is performing the corresponding gesture). In Figure 6.2 the triangle opacity increases since the user is following its shape. Finally, when the user lifts the finger from the screen, the application executes the command associated to the most likely gesture. OctoPocus is a good benchmark for evaluating the support offered by different recognition techniques for the UI development. Indeed, OctoPocus requires information on partially executed gestures, so the participant needs to invoke the recognition support *while* the user is performing the stroke. This task evaluates the ability of the underlying support to provide information for building feedback and feedforward representations, which motivated our works.

The participants implemented the two variants of the grid UI using three different recognition libraries:

1. The first library uses **Finite State Machine**s (FSMs) for recognizing polyline gestures, according to the movement orientation angle. A stroke is represented as an FSM having a state for each segment, defined by the direction in its ideal trajectory. A direction is represented as through an angle range in the goniometric circle.[1] When the user changes the stroke direction, the machine fires a transition. It goes to the next state if its corresponding range contains the new direction angle, or to an error state otherwise. The gesture is

[1] For instance, considering the four segments of the square stroke in Figure 6.2, we need four directions: down (270^o), right (0^o), up (90^o) and left (180^o). Obviously, the user's movements are not perfectly aligned to the ideal direction, so the developer specifies an interval around the ideal angle (e.g., $270^o \pm 20^o$) for the down direction.

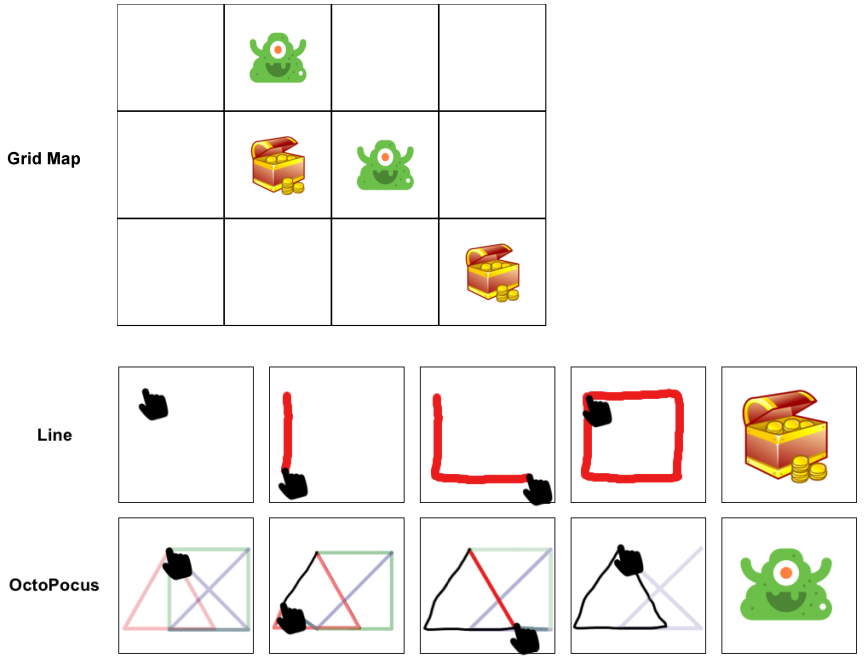


Figure 6.2: Gestural UI developed for the user test (top part). Line feedback for a square gesture (middle part) OctoPocus [14] feedback for a triangle gesture (bottom part).

recognised when the stroke ends and the FSM is in its final state. Such library represents a simplified declarative approach which uses geometric heuristics for the recognition. It summarises this category in the evaluation development task: gestures are easy to define (it is sufficient to declare the parts and their direction angle), it recognises gestures without training, and it provides information on partial gesture recognition (in our case, each FSM state corresponds to one of its sub-parts), but they are not robust to the user's input variability (the range definition is critical for a correct recognition).

2. The second library represents the **Machine Learning** approaches and it recognizes gestures according to a given set of labelled examples, which has to be provided by the developer. Such approaches are robust to the input noise, but they are a black box for developers. They provide a label for the whole stroke and they do not provide information on partial gesture recognition. In order to speed-up the

evaluation procedure while enabling the participants to grasp the importance of the training phase, we used the library presented in [241], since it requires only a few samples for each gesture, it provides a JavaScript implementation and the overall development interface is a good representative for the machine learning approaches.

3. The third library is **DEICTIC**, the approach described in this work, which defines gestures through expressions. The recognition does not require gesture-specific training samples (the dataset for the ground terms was included in the library) and provides information on partial gesture recognition as discussed in section 4.3.

Besides the gesture recognition support, we provided the participants with three other libraries for facilitating the interface development:

- *Grid*, which draws and manages the grid user interface, including functions for setting or resetting the cell content.
- *Feedback*, which provides the implementation of the drawing procedures for the *Line* and the *OctoPocus* feedback.
- *Input*, which masks the differences between mouse and touch events in the browser for easing the development tasks.

We documented the API of all pieces of software and we provided a tutorial on each component, including both explanations and sample code as usual in open-source libraries. We asked the participants to read the tutorials before starting the implementation tasks. The documentation material was available for them throughout the test, and they were free to read it whenever they wanted to. The test description and the documentation material for the evaluation are available in the DEICTIC source code distribution [38] [2].

All participants developed the grid interface using all recognition supports. The six possible orders for executing the development tasks with all supports were randomly assigned to each participant for counter-balancing the carry-over effect. The first task is the most critical for each participant since s/he has to learn both the gesture recognition support and the interface management components (grid, feedback and input). In the other two conditions, s/he can leverage on the grid, feedback and input components knowledge acquired in the previous tasks.

G-Gene Development Tasks The development task, performed to evaluated G-Gene, consists of coding the gesture interaction support for a simplified drawing application. The canvas responds to three simple stroke gestures: two of them have the effect of drawing the corresponding

[2] `https://github.com/davidespano/deictic/blob/master/user-test/gesturemap/basic/static/basic/js/lib/out/index.html`

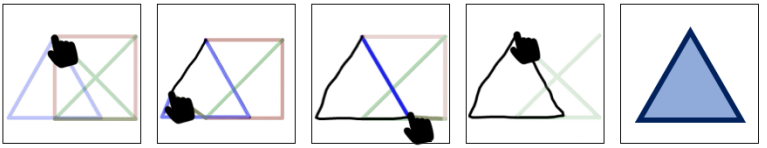


Figure 6.3: The simplified *Octopocus* [14] feedback for a triangle gesture.

geometric figure (triangle and rectangle) at the stroke position, and one has the effect of erasing the underlying object (the delete gesture). The user has to execute all gestures as depicted in Figure 6.3. Similarly to the development tasks required in the evaluation of DEICTIC, the interface guides the user during the performance by using a simplified *OctoPocus* [14] feedback, represented in Figure 6.3.

We assessed the G-Gene support in creating a well-known interface that requires feedback and feedforward information against one of the available toolkit discussed in Section 2 that provide information on the final gesture label. Our hypothesis is that the modelling effort introduced by the library is balanced by the advantages of receiving intermediate information.

The participants were requested to develop the same interface using two different recognition supports. The first one is the 1$ recognizer [241], which requires the developer to enter only a few examples for each considered gesture. It represents the techniques able to provide the label only at the end of the stroke as we discussed in Section 2.1. The second one is the G-Gene recognizer. We alternated the starting condition among the participants for balancing the carry-over effect.

In order to speed-up the development process, we created four JavaScript libraries that were provided to the test participants together with development tutorials and documentation as described in the following list[3].

- A *Canvas* management library. It contains a class representing a 3x4 matrix. Each cell may be empty, or it may contain a triangle or a square. The developer can sets cell content using the `setTriangle`, `setSquare` and `clearCell` methods.
- A *OctoPocus* library, containing a class for managing the stroke input and displaying the guidance for the considered gestures. The class has a `start` and a `clear` method respectively for showing and hiding the guidance underneath the current stroke position. At the beginning of the interaction all gesture guidance paths are considered as equally

[3]The documentation provided to the test participants is available at the URL: `hhttps://goo.gl/GL7xsT`

likely (see Figure 6.3, first box) The `update` method changes the current state of the visual guidance. It receives as parameter an array that, for each guiding shape, specifies the index of its current side and its opacity level. For instance, in the third box of Figure 6.3 the array element corresponding to the triangle guidance has 1 as side index (the second) and nearly 1 as opacity.

- The 1$-recognizer as described in [241]. It requires a training step, where the developer provides a small (3) number of labelled samples for each gesture through a dedicated interface. The samples are stored in a global variable. In the recognition phase, the library provides a buffer object for storing the stroke points. The object implementing the recognition algorithm has an `eval` method that receives the buffer as input and returns the likelihood of each gesture in the set, represented as an array of key-value pairs.
- A G-Gene JavaScript wrapper. It requires a modelling phase, where the developer writes the gesture definition expression and passes it to the library for initializing the underlying HMMs. This is accomplished by invoking the `init` method and passing a key-value pair array, containing for each gesture its name and a string representing the modelling expression as described in Section 5.1.2. Once the library is initialized, the developer uses the same buffer object described at the previous point for collecting the user's input. The `eval` method takes the buffer as a parameter and returns an array containing the likelihood of each gesture and its sub-parts.

6.1.3 Evaluation metrics and success criteria

In both tests, for each task, we collected the time spent and the completion rate. After finishing a task, the participants filled in questionnaire, including questions from a NASA-TLX [86], for evaluating a set of relevant criteria from those proposed for the evaluation of UI toolkits by Olsen [181], and an open-ended question for collecting their opinions and suggestions on the evaluated support. Finally, we analysed the developed code for finding common patterns in their solutions.

At the end of the test, we asked the participants to rank the approaches on their overall effectiveness, satisfaction and willingness to reuse, together with the criteria proposed by Olsen [181] for finding possible differences in the assessment after testing all approaches.

The questions for the selected UI toolkit criteria are the following (1 to 7 Likert scale):

- *Effectiveness.* Please rate how effective the gesture recognition support is in your opinion.

- *Perceived accuracy*: Please rate how accurate the gesture recognition support is in your opinion.
- *Flexibility*. How easily do you think that the current configuration supports rapid changes for e.g., evaluating them with the end-users?
- *Expressive Match*. How close the means for expressing the gesture design are to the problem being solved?
- *Inductive Combination*. How reusable is the solution you built with this gesture recognition support? How easily you can take out components that may be combined for creating other interfaces?
- *Overall satisfaction*. Please rate your level of satisfaction with respect to the gesture recognition support.
- *Willingness to reuse the approach in the future*. If you would need to implement gestural interfaces in the future, are you willing to reuse this recognition support?

In our hypothesis, DEICTIC and G-Gene should be able to provide intermediate feedback information with an effort comparable to heuristic approaches, with a definition procedure and perceived recognition accuracy comparable to machine learning approaches. Achieving such results constitutes the success criteria for these tests.

6.2 DEICTIC Evaluation

In this section we report the results that DEICTIC achieved in the development tasks described in Section 6.1. All tasks were successfully completed by all the users in all conditions but the *OctoPocus* feedback with the *Machine Learning* approach. In that task, two participants gave-up since not they were not able to retrieve the information on partially executed gestures. Other participants found difficulties in identifying a possible solution (5) and the moderator suggested them that partially executed gestures may be considered as gestures as well. With such advise, they added more labels to the set, representing each phase of a stroke execution (e.g., specifying one label for the first triangle side, another one for the first plus the second, and finally the complete triangle) and they were able to complete the task.

We use a one-way ANOVA for repeated measures for comparing the results across the three conditions: i) Finite State Machines (FSM), ii) Machine Learning (ML) and iii) DEICTIC (D). The resulting dataset had homogeneous variance and satisfy the sphericity assumption for all metrics, therefore no transformation nor correction was needed for running the one-way ANOVA analysis.

The post-task metrics collected for each task are summarised in Figure 6.4 (red boxes for the *Line* version and green boxes for the *OctoPocus*

version).

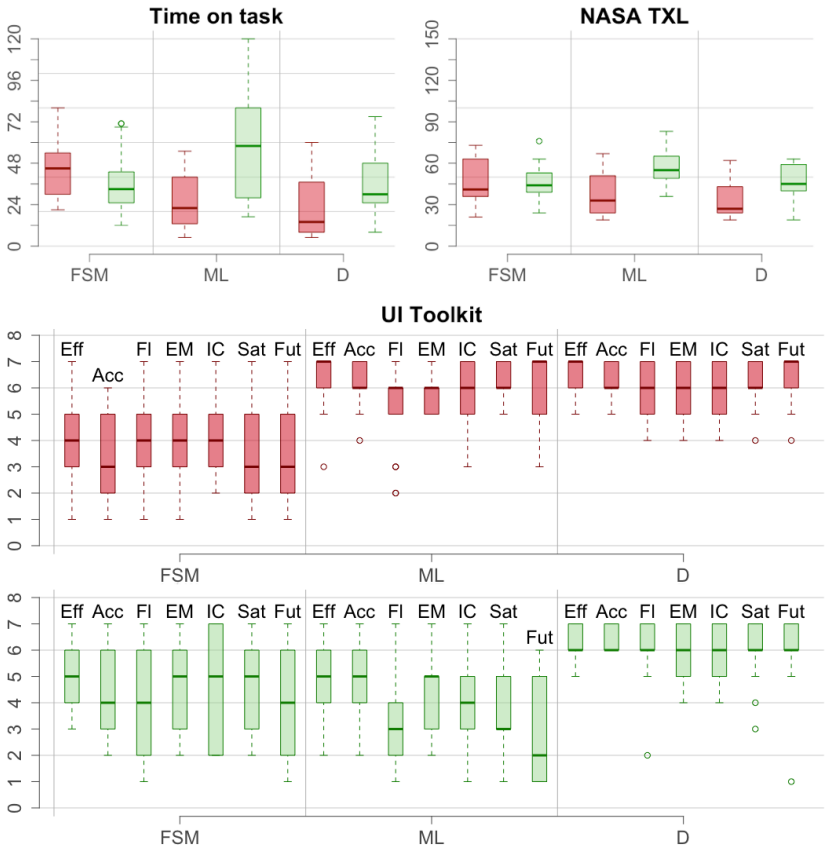


Figure 6.4: Post-task evaluation results. Red boxes correspond to the *line* feedback UI variant, while green boxes correspond to the *OctoPocus* variant. The first row shows the time on task and the task load (NASA TLX [86]), the second row shows the selected UI toolkit criteria for the *line* feedback, in the third row is depicted the same criteria for the *OctoPocus* task. The order for the criteria is the same as in section 6.1.3.

6.2.1 Line Feedback Task

We found the following significant effect of the recognition library on the following metrics[4]:

- Time on task ($F(2,16) = 6.673$, $p < .004$, $\eta^2 = 0.29$, in minutes, lower is better). We registered a significant difference between FSM and D ($p < .02$, *c.i.* $= [2.9, 31.6]$ min) and between FSM and ML ($p < .05$, *c.i.* $= [4.3, 34.7]$ min).

- Task load ($F(2,16) = 3.998$, $p < .03$, $\eta^2 = 0.20$, NASA TLX [86] in a 0 to 150 scale, lower is better). We registered an almost significant difference between FSM and D ($p < .004$, *c.i.* $= [0.7, 27.4]$).

- Effectiveness ($F(2,16) = 22.75$, $p < 10^{-3}$, $\eta^2 = 0.59$, 1 to 7 Likert scale, higher is better). We registered an almost difference between ML and FSM ($p < .002$, *c.i.* $= [0.90, 3.58]$) and between D and FSM ($p = .003$, *c.i.* $= [1.14, 3.68]$).

- Perceived accuracy ($F(2,16) = 40.74$, $p < 10^{-3}$, $\eta^2 = 0.71$, 1 to 7 Likert scale, higher is better). We registered a significant difference between ML and FSM ($p < 10^{-4}$, *c.i.* $= [1.56, 4.00]$) and between D and FSM ($p < 10^{-4}$, *c.i.* $= [1.75, 4.02]$).

- Flexibility ($F(2,16) = 7.944$, $p < .001$, $\eta^2 = 0.33$, 1 to 7 Likert scale, higher is better). We registered a significant difference between D and FSM ($p < .007$, *c.i.* $= [0.48, 3.16]$).

- Expressive Match ($F(2,16) = 7.37$, $p < .002$, $\eta^2 = 0.31$, 1 to 7 Likert scale, higher is better). We registered a significant difference between ML and FSM ($p < .01$, *c.i.* $= [0.29, 2.65]$) and between D and FSM ($p < .02$, *c.i.* $= [0.08, 2.62]$).

- Inductive Combination ($F(2,16) = 7.29$, $p < .002$, $\eta^2 = 0.31$, 1 to 7 Likert scale, higher is better). We registered a practical significant difference between ML and FSM ($p < .07$, *c.i.* $= [0.15, 2.67]$), and a significant one between D and FSM ($p < .01$, *c.i.* $= [0.28, 2.77]$).

- Overall Satisfaction ($F(2,16) = 21.06$, $p < 10^{-6}$, $\eta^2 = 0.57$, 1 to 7 Likert scale, higher is better). We registered a significant difference between ML and FSM ($p < 10^{-3}$, *c.i.* $= [0.99, 3.83]$), and between D and FSM ($p < 10^{-4}$, *c.i.* $= [0.99, 3.83]$).

[4] We used a Bonferroni-corrected pairwise comparison for establishing the differences between the gesture recognition library pairs.

- Willingness to reuse the approach in the future ($F(2,16) = 22.02$, $p < 10^{-5}$, $\eta^2 = 0.58$, 1 to 7 Likert scale, higher is better). We registered a significant difference between ML and FSM ($p < 10^{-3}$, *c.i.* = $[0.85, 4.08]$) and between D and FSM ($p < 10^{-4}$, *c.i.* = $[1.37, 4.39]$).

In summary, the collected data confirm our hypothesis for this task: developers preferred and performed better in defining the gesture set using the Machine Learning and DEICTIC, which achieved comparable results. They had more difficulties using FSMs that, according to the comments we collected, where tedious to tune-up for achieving an acceptable recognition performance.

6.2.2 OctoPocus feedback task

We found the following significant effect of the recognition library on the following metrics:

- Time on task ($F(2,16) = 4.838$, $p < .02$, $\eta^2 = 0.23$, in minutes, lower is better). We registered a significant difference between ML and D ($p = .05$, *c.i.* = $[1.4, 45.0]$ min).

- Task load ($F(2,16) = 4.011$, $p < .03$, $\eta^2 = 0.20$, NASA TLX [86] in a 0 to 150 scale, lower is better). We registered an almost significant difference between ML and D ($p < .08$, *c.i.* = $[0.8, 25.8]$) and between ML and FSM ($p < .07$, *c.i.* = $[0.6, 25.8]$).

- Effectiveness ($F(2,16) = 5.738$, $p < .008$, $\eta^2 = 0.26$, 1 to 7 Likert scale, higher is better). We registered an almost significant difference between D and ML ($p < .005$, *c.i.* = $[0.50, 2.55]$) and a significant difference between D and FSM ($p < .07$, *c.i.* = $[0.05, 2.18]$).

- Perceived accuracy ($F(2,16) = 9.372$, $p < 10^{-3}$, $\eta^2 = 0.37$, 1 to 7 Likert scale, higher is better). We registered a significant difference between D and FSM ($p < .003$, *c.i.* = $[0.72, 2.93]$).

- Flexibility ($F(2,16) = 12.07$, $p < .10^{-3}$, $\eta^2 = 0.43$, 1 to 7 Likert scale, higher is better). We registered a significant difference between D and FSM ($p < .05$, *c.i.* = $[0.37, 3.39]$) and between D and ML ($p < 10^{-3}$, *c.i.* = $[1.6, 4.18]$).

- Expressive Match ($F(2,16) = 4.781$, $p < .02$, $\eta^2 = 0.23$, 1 to 7 Likert scale, higher is better). We registered a significant difference between D and ML ($p < .005$, *c.i.* = $[0.39, 2.66]$).

- Inductive Combination ($F(2,16) = 7.755$, $p < .002$, $\eta^2 = 0.33$, 1 to 7 Likert scale, higher is better). We registered a significant difference between D and ML ($p < .002$, $c.i. = [1.09, 3.50]$).

- Overall Satisfaction($F(2,16) = 10.08$, $p = 10^{-3}$, $\eta^2 = 0.38$, 1 to 7 Likert scale, higher is better). We registered a significant difference between D and ML ($p < 10^{-3}$, $c.i. = [1.05, 3.65]$) and an almost significant difference between D and FSM ($p < .08$, $c.i. = [0.09, 2.61]$).

- Willing to reuse the approach in the future ($F(2,16) = 10.93$, $p = 10^{-4}$, $\eta^2 = 0.41$, 1 to 7 Likert scale, higher is better). We registered a significant difference between D and ML ($p = 10^{-3}$, $c.i. = [1.49, 4.39]$) and an almost significant one between D and FSM ($p < .09$, $c.i. = [0.13, 3.16]$).

The results confirm the hypothesis for this task as well: DEICTIC required a significantly lower time and effort for completing the task with respect to the Machine Learning approach, and it was consistently preferred by developers for all the considered criteria. DEICTIC performed better than the FSM approach considering the task load, showing that the modelling expressions fit the stroke gestures description better, as confirmed by the results for the overall satisfaction and willingness to reuse. The perceived accuracy was higher with respect to FSM as expected.

6.2.3 Post-test results

The post-test results in Figure 6.5 shows an overall preference for the DEICTIC approach in the development of stroke-based interactions. DEICTIC was considered the best option for all evaluated criteria by a large majority of the participants (min 12, max 16) and the second one by the others. In the expressiveness aspect, Machine Learning was considered as the best approach by 4 participants, that explained their choice by saying that drawing examples was easier than modelling the gestures. However, they acknowledged that samples should be collected by more than one user for reaching a reliable recognition rate.

The second approach in the ranking was Machine Learning. Participants explained this by saying that they preferred to work with a more difficult approach that reaches a higher accuracy with the user's input. The FSM was ranked third, even if participants acknowledged that it provided an easy way for accessing to the gesture sub-parts.

In summary, the DEICTIC approach was considered successful in bridging the two approaches and preserving the strong points from both

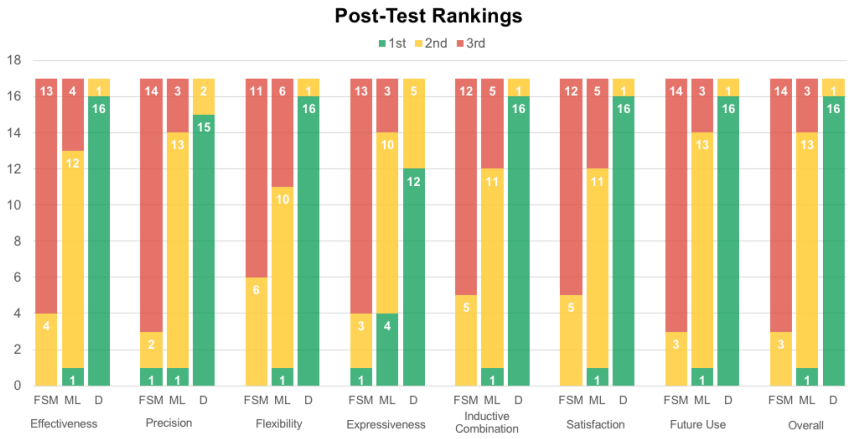


Figure 6.5: Post test evaluation results.

of them.

6.3 G-Gene Evaluation

In this section we report the results obtained by G-Gene in the simplified development task illustrated in the Section 6.3. All participants concluded the task with G-Gene, apart for two participants who were not able to conclude the development task with the 1$-recognizer. The registered time on task was significantly higher in the 1$ condition ($\bar{x}_1 = 58.06$, $s_1 = 30.00$ VS $\bar{x}_G = 34.88$, $s_G = 18.40$, $t(16) = 2.65$, $p < .02$, time in minutes), consistently with the user self-reported effort ($\bar{x}_1 = 58.24$, $s_1 = 14.72$ VS $\bar{x}_G = 34.88$, $s_G = 18.40$, $t(16) = 2.65$, $p < .02$, in a 1 to 100 scale). This demonstrates that the G-Gene development does not introduce additional effort for the modelling phase. In addition, participants spent less time in developing the interface with G-Gene and it required less effort.

We analysed the code produced during the test, in order to find the development patterns applied by developers for adapting the information received from the recognition support to the developed UI. The support offered by G-Gene was effective for them: they were consistently able to map the information on the current gesture sub-part and the continuation likelihood to the parameters requested by the *OctoPocus* library. None of them created code for tracking additional information directly from the input stream.

We registered two main patterns in the UI development with the 1$-recognizer. First of all, two participants gave up without being able to

figure out a strategy in a reasonable time (they spent more than one hour in the task). The majority of them reduced the granularity of the gesture definition, downgrading the gesture concept from the complete figure to its parts. In practice, they created a larger gesture set, which included the complete figures and each one of their subparts. For instance, they included three gestures for representing the triangle dynamic gesture: the first side, the first and the second side and then the complete gesture. In this way, the recognizer would classify the gesture *parts* considering them all as complete gestures.

Such approach has two disadvantages. First, it requires much more data for training the classifier, requiring examples for each gesture part. This may be solved by either collecting new samples or segmenting the existing ones. Neither of the solutions is easily practicable for a real dataset like the one we discussed in Section 5.2.1. The second problem is the representation of the gesture: the connection among the parts is implicit in the program, and so is the prediction of the gesture completion, which is completely hard-coded in the UI. This may result in poorly maintainable code when the number of gestures increases.

The second approach we registered consisted of using the 1$-recognizer for the classification at the end of the stroke and applying a developer-defined algorithm for identifying the parts. Basically, the participants defined a Finite State Machine (FSM) tracking the different parts of the gesture according to the movement direction. Such solution requires more developer effort in defining the heuristic for segmenting the gesture, which is obviously less accurate than the classification algorithm. In addition, the recognition at the end of the stroke and the tracking during the performance are not related, and this may surprise the user if the two parts diverge in labelling the stroke. Indeed, it may happen that e.g., the FSM-based guidance system helps the user in completing a square that will be recognized as a triangle by the 1$-recogniser.

In the open-ended questions, we asked the participants to provide feedback on the strong and weak aspects for both approaches. They acknowledged that providing (a few) examples for the 1$-recognizer is a very easy modelling technique. However, they considered more difficult tracking the gesture part when needed by the guidance system. Instead, they all considered reasonable the effort in modelling gestures with G-Gene expressions for getting the intermediate guidance information from the underlying support. They considered also the G-Gene expression a gesture representation that closely represent the information required for building a UI. However, 4 of them clearly stated that they would go for the 1$-

recognizer if the recognition without guidance was sufficient for building the interface. Therefore, we can conclude that G-Gene supports effectively the development of gestural UI, but the modelling technique is worth when the interface requires intermediate feedback and feedforward.

Chapter 7

Conclusions

7.1 Summary of Contributions

Over the years, different machine learning methods, like Hidden Markov Models, Dynamic Time Warping, Time-Delay Neural Networks, Finite-State Machines, Support Vector Machines, have been employed for recognizing gestures online: a large part of them supports developers by providing a high accuracy for different types of gestures. Generally, in such methods, the recognition process requires the complete gesture sample before recognizing it. Therefore, if on the one hand including classifiers techniques solve the accuracy problem, on the other hand, they do not support the generation of feedback and feedforward mechanisms in a guiding system. In gestural interfaces, such mechanism is useful to make the user aware of which gestures are available for communicating with the application.

In the literature there are several compositional methods that address both problems (the second in particular), allowing developers to display hints which help the user to perform gestures correctly. If on the one hand, these methods support the development of effective guidance systems, on the other hand, they rely on heuristics that reduce the recognition accuracy. In this thesis, we implemented two novel methods for recognizing stroke gestures, which fill the gap between the high accuracy offered by classification approaches and the inspection capabilities needed for providing feedback and feedforward in user interfaces.

In chapter 4 we discussed DEICTIC, a declarative and compositional description for stroke gestures, based on the composition of a set of basic

movements (points for starting a stroke, arcs and lines for its continuation) through a set of operators (iterative, sequence, choice, parallel). We defined a syntax for describing strokes through simple expressions, allowing to describe both single and multi-stroke gestures. We described a set of algorithms that, according to the expression definition, create an HMM that recognises gestures following the temporal composition semantics, without any additional training. On the one hand, the composed HMMs have a set of properties that make them suitable for defining user interfaces, breaking the single-event notification at the complete recognition of a gesture, without the need of a specific training set for each composite gesture. Such HMMs only require a dataset for training the expression ground terms, reusable across different gesture sets. DEICTIC is able to provide the most likely sequence of ground terms recognised, together with information on the most likely completion of the current sequence. The developer evaluation shows that the information provided by DEICTIC supports the implementation of feedback and feedforward with an effort comparable to heuristic approaches, together with a definition procedure and perceived recognition accuracy comparable to machine learning approaches. On the other hand, the recognition accuracy of the HMM built through the composition mechanism is comparable with respect to other approaches in the literature which is an important improvement if compared with heuristic approaches. We discussed two different experiments where we show that DEICTIC does not introduce sensible degradation of the recognition accuracy.

However, DEICTIC exploits a normalisation step for dealing with shape scaling, that does not allow to use for the online recognition in a general case. In order to address such limitation, we defined G-Gene, discussed in chapter 5. G-Gene is a method for representing gestures as character strings that can be recognized by Hidden Markov Models, by applying a gene-sequencing technique. More in detail, we employed profile-HMMs, a specialized type of Markov sources, which are designed for alignment of sequences. The proposed method preserves the composition information on gesture parts, in order to support the development of gesture guidance systems through feedback and feedforward. In addition, it exploits only local features for enabling the recognition while the user is performing the gesture. Experimental results on the 1$-dataset [241] show that our approach preserves a good accuracy, together with the information on gesture sub-parts. In addition, we evaluated the approach on a gestural UI development task, which required to include the OctoPocus [14] guidance system in a simple drawing application, comparing the effort and the strategies against a recognition support not providing partial gesture

recognition. The development with G-Gene required both less time and effort, supporting effectively the creation of the UI prototype without adding too much complexity in the gesture description.

7.2 Limitations and Future Work

The approaches presented in this dissertation include different limitations which can be improved. The HMMs generated by our methods do not support 3D gestures. Therefore, they are unusable in those interfaces which are designed for 3D manipulation and interaction, for example, applications based on Hololens or Oculus devices. In the future, we would like to extend both methods by supporting 3D stroke gestures. After that, another research direction is to validate the composition approach for 3D interfaces and investigating the performance of such solutions in developing 3D guiding systems.

Another common limitation is related to the recognition accuracy of our approaches, in particular for G-Gene. Despite our experiments demonstrate an overall good performance, more accurate recognition algorithms exist. It is worth pointing out that these algorithms rely on other features, not only touch or joint coordinate spaces. In the future, we would like to experiment with different sets of features, in order to improve the performance of our methods. In addition, a new set of features can be useful to overcome other limitations. On the one hand, the use of other sets of features may avoid the need of a preprocessing step in the general case, which limits the DEICTIC scope to applications that know the position and the scaling of the user's strokes. On the other hand, experimental tests showed that the use of local features reduces the robustness of the HMMs generated by G-Gene, and consequently its recognition accuracy. A new set of features can reduce the impact of noise or small user errors in the accuracy of G-Gene.

It is worth pointing out that DEICTIC can work online, namely without preprocessing the whole user movements, if scale and position of gesture input are known, as happens for instance in the grid used for the developer test. A possible method to address this limitation consists of defining a set of HMMs for each gesture, describing its temporal evolution through an HMM for each primitive. In more detail, this solution aims to move up the preprocessing step employing an HMM for recognizing each sequence of primitives which compose the gesture. In future we would like to extend DEICTIC, by implementing and evaluating the accuracy of this solution in online recognition.

In addition, DEICTIC presents a second limitation which can be improved. This limitation is related to the number of states in the final HMMs. The composition approach forces the growth of the number of states linearly (for sequence and choice) or quadratically (parallel). Instead, ad-hoc HMMs may be optimised to balance the trade-off between recognition rate and the number of states. In the implementation phase, we verified that not all states are always reachable after the training phase. Therefore, it might be possible to remove these states preserving the original connection and the recognition rate. In the future, we plan to address this problem by applying some algorithms which allow improving efficiency.

Considering G-Gene, it is worth pointing out that such method is not able to distinguish gestures which differ from the length of their primitives. Even in this case, that problem is connected to the local features used to split trajectories in real time. In the future, we would like to extend G-Gene by characterizing each primitive with its length thus allowing to distinguish gestures with the same profile but different lengths, such a V from a checkmark (✓) stroke.

In our work, we relied on HMMs, a well-known supervised algorithm suitable to handle temporal sequences, for recognizing basic movements (DEICTIC) or aligning string sequences (G-Gene). However, their performances in terms of recognition rate and time are conditioned from both used features and number of states. In general, the accuracy of such models can be improved by increasing both the number of states and/or the involved features. However, these solutions would reduce the efficiency in our methods, overfitting the models and increasing the time spent in both training the generated models and evaluating the sequences extracted from the user input. In the future, we plan to evaluate these components in order to find a better configuration. In addition, we also plan to study the properties of other machine learning-based approaches, such as time delay neural networks and recurrent neural networks. We aim to determine whether approaches are suitable to be combined with declarative approaches, as we have already done with HMMs.

Bibliography

[1] Aleotti, J., Cionini, A., Fontanili, L., and Caselli, S. Arm gesture recognition and humanoid imitation using functional principal component analysis. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on* (2013), IEEE, pp. 3752–3758.

[2] Alharbi, N., Liang, Y., and Wu, D. A data preprocessing technique for gesture recognition based on extended-kalman-filter. In *Proceedings of the Second IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies* (2017), IEEE Press, pp. 77–83.

[3] Alt, F., Geiger, S., and Höhl, W. Shapelineguide: Teaching mid-air gestures for large interactive displays. In *Proceedings of the 7th ACM International Symposium on Pervasive Displays* (2018), ACM, p. 3.

[4] Amin, M. A., and Yan, H. Sign language finger alphabet recognition from gabor-pca representation of hand gestures. In *Machine learning and cybernetics, 2007 international conference on* (2007), vol. 4, IEEE, pp. 2218–2223.

[5] Amir, A., Taba, B., Berg, D. J., Melano, T., McKinstry, J. L., Di Nolfo, C., Nayak, T. K., Andreopoulos, A., Garreau, G., Mendoza, M., et al. A low power, fully event-based gesture recognition system. In *CVPR* (2017), pp. 7388–7397.

[6] Anderson, F., Grossman, T., Matejka, J., and Fitzmaurice, G. Youmove: enhancing movement training with an augmented reality mirror. In *Proceedings of the 26th annual ACM symposium on User interface software and technology* (2013), ACM, pp. 311–320.

[7] Anthony, L., and Wobbrock, J. O. A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics*

Interface 2010 (2010), Canadian Information Processing Society, pp. 245–252.

[8] Anthony, L., and Wobbrock, J. O. $ n-protractor: A fast and accurate multistroke recognizer. In *Proceedings of Graphics Interface 2012* (2012), Canadian Information Processing Society, pp. 117–120.

[9] Aoki, R., Chan, B., Ihara, M., Kobayashi, T., Kobayashi, M., and Kagami, S. A gesture recognition algorithm for vision-based unicursal gesture interfaces. In *Proceedings of the 10th European conference on Interactive tv and video* (2012), ACM, pp. 53–56.

[10] Appert, C., and Bau, O. Scale detection for a priori gesture recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), ACM, pp. 879–882.

[11] Araga, Y., Shirabayashi, M., Kaida, K., and Hikawa, H. Real time gesture recognition system using posture classifier and jordan recurrent neural network. In *Neural Networks (IJCNN), The 2012 International Joint Conference on* (2012), IEEE, pp. 1–8.

[12] Ball, A., Rye, D., Ramos, F., and Velonaki, M. A comparison of unsupervised learning algorithms for gesture clustering. In *Human-Robot Interaction (HRI), 2011 6th ACM/IEEE International Conference on* (2011), IEEE, pp. 111–112.

[13] Balli Altuglu, T., and Altun, K. Recognizing touch gestures for social human-robot interaction. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction* (2015), ACM, pp. 407–413.

[14] Bau, O., and Mackay, W. E. Octopocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st annual ACM symposium on User interface software and technology* (2008), ACM, pp. 37–46.

[15] Baum, L. E., Petrie, T., Soules, G., and Weiss, N. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics 41*, 1 (2 1970), 164–171.

[16] Bautista, M. A., Hernández-Vela, A., Escalera, S., Igual, L., Pujol, O., Moya, J., Violant, V., and Anguera, M. T. A gesture recognition system for detecting behavioral patterns of adhd. *IEEE transactions on cybernetics 46*, 1 (2016), 136–147.

[17] Ben, K., and Van der Smagt, P. An introduction to neural networks. *University of Amsterdam* (1996).

[18] Benatti, S., Casamassima, F., Milosevic, B., Farella, E., Schönle, P., Fateh, S., Burger, T., Huang, Q., and Benini, L. A versatile embedded platform for emg acquisition and gesture recognition. *IEEE transactions on biomedical circuits and systems 9*, 5 (2015), 620–630.

[19] Besacier, L., Barnard, E., Karpov, A., and Schultz, T. Automatic speech recognition for under-resourced languages: A survey. *Speech Communication 56* (2014), 85–100.

[20] Bhowmick, S., Kumar, S., and Kumar, A. Hand gesture recognition of english alphabets using artificial neural network. In *Recent Trends in Information Systems (ReTIS), 2015 IEEE 2nd International Conference on* (2015), IEEE, pp. 405–410.

[21] Billinghurst, M., and Buxton, B. Gesture based interaction. *Haptic input 24* (2011).

[22] Birk, H., Moeslund, T. B., and Madsen, C. B. Real-time recognition of hand alphabet gestures using principal component analysis. In *Proceedings of the Scandinavian conference on image analysis* (1997), vol. 1, PROCEEDINGS PUBLISHED BY VARIOUS PUBLISHERS, pp. 261–268.

[23] Bishop, C., Bishop, C. M., et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[24] Bishop, G., Welch, G., et al. An introduction to the kalman filter. *Proc of SIGGRAPH, Course 8*, 27599-3175 (2001), 59.

[25] Bohari, U., Chen, T.-J., et al. To draw or not to draw: Recognizing stroke-hover intent in non-instrumented gesture-free mid-air sketching. In *23rd International Conference on Intelligent User Interfaces* (2018), ACM, pp. 177–188.

[26] Bora, R., Bisht, A., Saini, A., Gupta, T., and Mittal, A. Isl gesture recognition using multiple feature fusion. In *Wireless Communications, Signal Processing and Networking (WiSPNET), 2017 International Conference on* (2017), IEEE, pp. 196–199.

[27] Borman, S. The expectation maximization algorithm-a short tutorial. *Submitted for publication* (2004), 1–9.

[28] Bragdon, A., Zeleznik, R., Williamson, B., Miller, T., and LaViola Jr, J. J. Gesturebar: improving the approachability of gesture-based interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2009), ACM, pp. 2269–2278.

[29] Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[30] Cadoz, C. Le geste canal de communication homme/machine: la communication" instrumentale". *Technique et science informatiques 13*, 1 (1994), 31–61.

[31] Camurri, A., Canepa, C., Ghisio, S., and Volpe, G. Automatic classification of expressive hand gestures on tangible acoustic interfaces according to laban s theory of effort. In *International Gesture Workshop* (2007), Springer, pp. 151–162.

[32] Camurri, A., Mazzarino, B., Ricchetti, M., Timmers, R., and Volpe, G. Multimodal analysis of expressive gesture in music and dance performances. In *International gesture workshop* (2003), Springer, pp. 20–39.

[33] Camurri, A., Mazzarino, B., and Volpe, G. Analysis of expressive gesture: The eyesweb expressive gesture processing library. In *International Gesture Workshop* (2003), Springer, pp. 460–467.

[34] Camurri, A., Volpe, G., Piana, S., Mancini, M., Niewiadomski, R., Ferrari, N., and Canepa, C. The dancer in the eye: towards a multi-layered computational framework of qualities in movement. In *Proceedings of the 3rd International Symposium on Movement and Computing* (2016), ACM, p. 6.

[35] Canavan, S., Keyes, W., Mccormick, R., Kunnumpurath, J., Hoelzel, T., and Yin, L. Hand gesture recognition using a skeleton-based feature representation with a random regression forest. In *Image Processing (ICIP), 2017 IEEE International Conference on* (2017), IEEE, pp. 2364–2368.

[36] Caputo, F. M., Prebianca, P., Carcangiu, A., Spano, L. D., and Giachetti, A. A 3 cent recognizer: Simple and effective retrieval and classification of mid-air gestures from single 3d traces. *Smart Tools and Apps for Graphics. Eurographics Association* (2017).

[37] Caputo, F. M., Prebianca, P., Carcangiu, A., Spano, L. D., and Giachetti, A. Comparing 3d trajectories for simple mid-air gesture recognition. *Computers & Graphics 73* (2018), 17–25.

[38] Carcangiu, A., and Spano, L. D. DEICTIC Python prototype implementation. `https://github.com/davidespano/deictic`. Accessed: 2017-03-13.

[39] Carcangiu, A., and Spano, L. D. G-gene: A gene alignment method for online partial stroke gestures recognition. *Proceedings of the ACM on Human-Computer Interaction 2*, EICS (2018), 13.

[40] Carcangiu, A., Spano, L. D., Fumera, G., and Roli, F. Gesture modelling and recognition by integrating declarative models and pattern recognition algorithms. In *International Conference on Image Analysis and Processing* (2017), Springer, pp. 84–95.

[41] Carcangiu, A., Spano, L. D., Fumera, G., and Roli, F. Deictic: A compositional and declarative gesture description based on hidden markov models. *International Journal of Human-Computer Studies 122* (2019), 113–132.

[42] Chai, X., Liu, Z., Yin, F., Liu, Z., and Chen, X. Two streams recurrent neural networks for large-scale continuous gesture recognition. In *Pattern Recognition (ICPR), 2016 23rd International Conference on* (2016), IEEE, pp. 31–36.

[43] Cheema, S., Hoffman, M., and LaViola Jr, J. J. 3d gesture classification with linear acceleration and angular velocity sensing devices for video games. *Entertainment Computing 4*, 1 (2013), 11–24.

[44] Chen, Q., Georganas, N. D., and Petriu, E. M. Real-time vision-based hand gesture recognition using haar-like features. In *Proceedings of IMTC 2007* (2007), IEEE, pp. 1–6.

[45] Chen, X., Guo, H., Wang, G., and Zhang, L. Motion feature augmented recurrent neural network for skeleton-based dynamic hand gesture recognition. In *Image Processing (ICIP), 2017 IEEE International Conference on* (2017), IEEE, pp. 2881–2885.

[46] Cheng, H., Yang, L., and Liu, Z. Survey on 3D Hand Gesture Recognition. *IEEE Trans. Circuits Syst. Video Techn. 26*, 9 (2016), 1659–1673.

[47] Choi, H.-R., Kim, E.-J., and Kim, T.-Y. A dtw gesture recognition system based on gesture orientation histogram. In *Consumer Electronics (ISCE 2014), The 18th IEEE International Symposium on* (2014), IEEE, pp. 1–2.

[48] Cirelli, M., and Nakamura, R. A survey on multi-touch gesture recognition and multi-touch frameworks. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces* (2014), ACM, pp. 35–44.

[49] Cracknell, J., Cairns, A., Gregor, P., Ramsay, C., and Ricketts, I. Gesture recognition: an assessment of the performance of recurrent neural networks versus competing techniques. In *Applications of Neural Networks to Signal Processing (Digest No. 1994/248), IEE Colloquium on* (1994), IET, pp. 8–1.

[50] Cuenca, F., Van den Bergh, J., Luyten, K., and Coninx, K. Hasselt: Rapid prototyping of multimodal interactions with composite event-driven programming. *International Journal of People-Oriented Programming (IJPOP) 5*, 1 (2016), 19–38.

[51] Czuszynski, K., Kwasniewska, A., Szankin, M., and Ruminski, J. Optical sensor based gestures inference using recurrent neural network in mobile conditions. In *2018 11th International Conference on Human System Interaction (HSI)* (2018), IEEE, pp. 101–106.

[52] Czuszynski, K., Ruminski, J., and Kwasniewska, A. Gesture recognition with the linear optical sensor and recurrent neural networks. *IEEE Sensors Journal* (2018).

[53] Daliri, F., and Girouard, A. Visual feedforward guides for performing bend gestures on deformable prototypes. In *Graphics Interface* (2016), pp. 209–216.

[54] Dekker, B., Jacobs, S., Kossen, A., Kruithof, M., Huizing, A., and Geurts, M. Gesture recognition with a low power fmcw radar and a deep convolutional neural network. In *Radar Conference (EURAD), 2017 European* (2017), IEEE, pp. 163–166.

[55] Dekker, B., Jacobs, S., Kossen, A., Kruithof, M., Huizing, A., and Geurts, M. Gesture recognition with a low power fmcw radar and a deep convolutional neural network. In *Radar Conference (EURAD), 2017 European* (2017), IEEE, pp. 163–166.

[56] Delamare, W., Coutrix, C., and Nigay, L. Designing guiding systems for gesture-based interaction. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (2015), ACM, pp. 44–53.

[57] Delamare, W., Coutrix, C., and Nigay, L. A tool for optimizing the use of a large design space for gesture guiding systems.

In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (2015), ACM, pp. 238–241.

[58] Delamare, W., Janssoone, T., Coutrix, C., and Nigay, L. Designing 3d gesture guidance: visual feedback and feedforward design options. In *Proceedings of the International Working Conference on Advanced Visual Interfaces* (2016), ACM, pp. 152–159.

[59] Deng, J.-W., and Tsui, H.-T. An hmm-based approach for gesture segmentation and recognition. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on* (2000), vol. 3, IEEE, pp. 679–682.

[60] Deo, N., Rangesh, A., and Trivedi, M. In-vehicle hand gesture recognition using hidden markov models. In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on* (2016), IEEE, pp. 2179–2184.

[61] Devineau, G., Moutarde, F., Xi, W., and Yang, J. Deep learning for hand gesture recognition on skeletal data. In *Automatic Face & Gesture Recognition (FG 2018), 2018 13th IEEE International Conference on* (2018), IEEE, pp. 106–113.

[62] Dietterich, T. G., and Bakiri, G. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research 2* (1994), 263–286.

[63] Doliotis, P., Stefan, A., McMurrough, C., Eckhard, D., and Athitsos, V. Comparing gesture recognition accuracy using color and depth information. In *Proceedings of the 4th international conference on PErvasive technologies related to assistive environments* (2011), ACM, p. 20.

[64] Dominio, F., Donadeo, M., Marin, G., Zanuttigh, P., and Cortelazzo, G. M. Hand gesture recognition with depth data. In *Proceedings of the 4th ACM/IEEE international workshop on Analysis and retrieval of tracked events and motion in imagery stream* (2013), ACM, pp. 9–16.

[65] Dominio, F., Donadeo, M., and Zanuttigh, P. Combining multiple depth-based descriptors for hand gesture recognition. *Pattern Recognition Letters 50* (2014), 101–111.

[66] Echtler, F., and Butz, A. Gispl: gestures made easy. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction* (2012), ACM, pp. 233–240.

[67] Elliott, R. J., Aggoun, L., and Moore, J. B. *Hidden Markov models: estimation and control*, vol. 29. Springer Science & Business Media, 2008.

[68] Ellis, C., Masood, S. Z., Tappen, M. F., LaViola, J. J., and Sukthankar, R. Exploring the trade-off between accuracy and observational latency in action recognition. *International Journal of Computer Vision 101*, 3 (2013), 420–436.

[69] Endres, C., Schwartz, T., and Müller, C. A. Geremin: 2d microgestures for drivers based on electric field sensing. In *Proceedings of the 16th international conference on Intelligent user interfaces* (2011), ACM, pp. 327–330.

[70] Escalera, S., Baró, X., Gonzalez, J., Bautista, M. A., Madadi, M., Reyes, M., Ponce-López, V., Escalante, H. J., Shotton, J., and Guyon, I. Chalearn looking at people challenge 2014: Dataset and results. In *Workshop at the European Conference on Computer Vision* (2014), Springer, pp. 459–473.

[71] for Intelligent, L. L., and Automobiles, S. Viva, 2016. `http://cvrr.ucsd.edu/vivachallenge/`, Last accessed on 2018-09-23.

[72] Forney, G. D. The viterbi algorithm. *Proceedings of the IEEE 61*, 3 (1973), 268–278.

[73] Fradkin, D., and Muchnik, I. Support vector machines for classification. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 70* (2006), 13–20.

[74] Gani, E., Kika, A., and Goxhi, B. A real-time vision based system for recognition of static dactyls of albanian alphabet. In *RTA-CSIT* (2016), pp. 17–22.

[75] Gaus, Y. F. A., Olugbade, T., Jan, A., Qin, R., Liu, J., Zhang, F., Meng, H., and Bianchi-Berthouze, N. Social touch gesture recognition using random forest and boosting on distinct feature sets. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction* (2015), ACM, pp. 399–406.

[76] Ge, S. S., Yang, Y., and Lee, T. H. Hand gesture recognition and tracking based on distributed locally linear embedding. *Image and Vision Computing 26*, 12 (2008), 1607–1620.

[77] Ghosh, D. K., and Ari, S. Static hand gesture recognition using mixture of features and svm classifier. In *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on* (2015), IEEE, pp. 1094–1099.

[78] Glowinski, D., Dael, N., Camurri, A., Volpe, G., Mortillaro, M., and Scherer, K. Toward a minimal representation of affective gestures. *IEEE Transactions on Affective Computing 2*, 2 (2011), 106–118.

[79] Glowinski, D., Mortillaro, M., Scherer, K., Dael, N., Volpe, G., and Camurri, A. Towards a minimal representation of affective gestures. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on* (2015), IEEE, pp. 498–504.

[80] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[81] Guillon, M., Leitner, F., and Nigay, L. Target expansion lens: It is not the more visual feedback the better! In *Proceedings of the International Working Conference on Advanced Visual Interfaces* (2016), ACM, pp. 52–59.

[82] Gupta, S., Jaafar, J., and Ahmad, W. F. W. Static hand gesture recognition using local gabor filter. *Procedia Engineering 41* (2012), 827–832.

[83] Gurney, K. *An introduction to neural networks*. CRC press, 2014.

[84] Hachaj, T., Ogiela, M. R., and Koptyra, K. Application of hidden markov models and gesture description language classifiers to oyama karate techniques recognition. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2015 9th International Conference on* (2015), IEEE, pp. 160–165.

[85] Hang, C., Zhang, R., Chen, Z., Li, C., and Li, Z. Dynamic gesture recognition method based on improved dtw algorithm. In *2017 International Conference on Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII)* (2017), IEEE, pp. 71–74.

[86] Hart, S. G., and Staveland, L. E. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, vol. 52. Elsevier, 1988, pp. 139–183.

[87] Hastie, T., Tibshirani, R., and Friedman, J. Unsupervised learning. In *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[88] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[89] Holte, M. B., and Moeslund, T. B. View invariant gesture recognition using 3d motion primitives. In *Proceedings of ICASSP 2008* (2008), IEEE, pp. 797–800.

[90] Hopcroft, J. E. *Introduction to Automata Theory, Languages and Computation: For VTU, 3/e*. Pearson Education India, 2013.

[91] Hoste, L., Dumas, B., and Signer, B. Mudra: a unified multimodal interaction framework. In *Proceedings of the 13th international conference on multimodal interfaces* (2011), ACM, pp. 97–104.

[92] Hoste, L., and Signer, B. Criteria, challenges and opportunities for gesture programming languages. *Proc. of EGMI* (2014), 22–29.

[93] Hsu, Y.-L., Chu, C.-L., Tsai, Y.-J., and Wang, J.-S. An inertial pen with dynamic time warping recognizer for handwriting and gesture recognition. *IEEE Sensors Journal 15*, 1 (2015), 154–163.

[94] Huang, D.-Y., Hu, W.-C., and Chang, S.-H. Vision-based hand gesture recognition using pca+ gabor filters and svm. In *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP'09. Fifth International Conference on* (2009), IEEE, pp. 1–4.

[95] Jangyodsuk, P., Conly, C., and Athitsos, V. Sign language recognition using dynamic time warping and hand shape distance based on histogram of oriented gradient features. In *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments* (2014), ACM, p. 50.

[96] Ji, S., Xu, W., Yang, M., and Yu, K. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence 35*, 1 (2013), 221–231.

[97] Ji, Y., Ko, Y., Shimada, A., Nagahara, H., and Taniguchi, R.-i. Cooking gesture recognition using local feature and depth image. In *Proceedings of the ACM multimedia 2012 workshop on Multimedia for cooking and eating activities* (2012), ACM, pp. 37–42.

[98] John, V., Umetsu, M., Boyali, A., Mita, S., Imanishi, M., Sanma, N., and Shibata, S. Real-time hand posture and gesture-based touchless automotive user interface using deep learning. In *2017 IEEE Intelligent Vehicles Symposium (IV)* (2017), IEEE, pp. 869–874.

[99] Jolliffe, I. Principal component analysis. In *International encyclopedia of statistical science*. Springer, 2011, pp. 1094–1096.

[100] Joshi, A., Monnier, C., Betke, M., and Sclaroff, S. A random forest approach to segmenting and classifying gestures. In *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on* (2015), vol. 1, IEEE, pp. 1–7.

[101] Kalman, R. E. A new approach to linear filtering and prediction problems. *Journal of basic Engineering 82*, 1 (1960), 35–45.

[102] Kammer, D., Wojdziak, J., Keck, M., Groh, R., and Taranko, S. Towards a formalization of multi-touch gestures. In *ACM International Conference on Interactive Tabletops and Surfaces* (2010), ACM, pp. 49–58.

[103] Kao, C.-Y., and Fahn, C.-S. A human-machine interaction technique: hand gesture recognition based on hidden markov models with trajectory of hand motion. *Procedia Engineering 15* (2011), 3739–3743.

[104] Karaboga, D. An idea based on honey bee swarm for numerical optimization. Tech. rep., Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.

[105] Karam, M., et al. A taxonomy of gestures in human computer interactions.

[106] Kazllarof, V., Karlos, S., Panagopoulou, A.-P., and Kotsiantis, S. Automated hand gesture recognition for educational applications. In *Proceedings of the 20th Pan-Hellenic Conference on Informatics* (2016), ACM, p. 20.

[107] Keogh, E., and Ratanamahatana, C. A. Exact indexing of dynamic time warping. *Knowledge and information systems 7*, 3 (2005), 358–386.

[108] Keskin, C., Erkan, A., and Akarun, L. Real time hand tracking and 3d gesture recognition for interactive interfaces using hmm. *ICANN/ICONIPP 2003* (2003), 26–29.

[109] Keskin, C., Kiraç, F., Kara, Y. E., and Akarun, L. Real time hand pose estimation using depth sensors. In *Consumer depth cameras for computer vision*. Springer, 2013, pp. 119–137.

[110] Khandkar, S. H., and Maurer, F. A domain specific language to define gestures for multi-touch applications. In *Proceedings of the 10th Workshop on Domain-Specific Modeling* (2010), ACM, p. 2.

[111] Kim, I., and Chien, S. Analysis of 3D Hand Trajectory Gestures Using Stroke-Based Composite Hidden Markov Models. *Appl. Intell. 15*, 2 (2001), 131–143.

[112] Kim, S. Y., Han, H. G., Kim, J. W., Lee, S., and Kim, T. W. A hand gesture recognition sensor using reflected impulses. *IEEE Sensors Journal 17*, 10 (2017), 2975–2976.

[113] Kin, K., Hartmann, B., DeRose, T., and Agrawala, M. Proton++ : A Customizable Declarative Multitouch Framework. In *Proceedings of UIST 2012* (Berkeley, California, USA, 2012), ACM Press, pp. 477–486.

[114] Kin, K., Hartmann, B., DeRose, T., and Agrawala, M. Proton: multitouch gestures as regular expressions. In *Proceedings of CHI 2012* (Austin, Texas, USA, 2012), ACM Press, pp. 2885–2894.

[115] Koller, O., Ney, H., and Bowden, R. Deep hand: How to train a cnn on 1 million hand images when your data is continuous and weakly labelled. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 3793–3802.

[116] Kopinski, T., Sachara, F., and Handmann, U. A deep learning approach to mid-air gesture interaction for mobile devices from time-of-flight data. In *Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services* (2016), ACM, pp. 1–9.

[117] Kramer, J., and Leifer, L. The talking glove. *ACM SIGCAPH Computers and the Physically Handicapped*, 39 (1988), 12–16.

[118] Kratz, L., Smith, M., and Lee, F. J. Wiizards: 3d gesture recognition for game play input. In *Proceedings of the 2007 conference on Future Play* (2007), ACM, pp. 209–212.

[119] Kratz, S., Rohs, M., and Essl, G. Combining acceleration and gyroscope data for motion gesture recognition using classifiers with dimensionality constraints. In *Proceedings of the 2013 international conference on Intelligent user interfaces* (2013), ACM, pp. 173–178.

[120] Kratz, S., and Wiese, J. 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* (2016), ACM, pp. 61–72.

[121] Kristensson, P.-O., and Zhai, S. SHARK 2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of UIST 2004* (2004), ACM, pp. 43–52.

[122] Krupka, E., Karmon, K., Bloom, N., Freedman, D., Gurvich, I., Hurvitz, A., Leichter, I., Smolin, Y., Tzairi, Y., Vinnikov, A., et al. Toward realistic hands gesture interface: Keeping it simple for developers and machines. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017), ACM, pp. 1887–1898.

[123] Kumar, P., Saini, R., Behera, S. K., Dogra, D. P., and Roy, P. P. Real-time recognition of sign language gestures and air-writing using leap motion. In *Machine Vision Applications (MVA), 2017 Fifteenth IAPR International Conference on* (2017), IEEE, pp. 157–160.

[124] Kurtenbach, G. Gestures in human-computer communication. *The art of human computer interface design 309* (1990).

[125] Kurtenbach, G., Moran, T. P., and Buxton, W. Contextual animation of gestural commands. In *Computer Graphics Forum* (1994), vol. 13, Wiley Online Library, pp. 305–314.

[126] Laban, R., and Lawrence, F. Effort london: Macdonald & evans. *Google Scholar* (1947).

[127] Lamar, M. V., Bhuiyan, M. S., and Iwata, A. Hand gesture recognition using morphological principal component analysis and an improved combnet-ii. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on* (1999), vol. 4, IEEE, pp. 57–62.

[128] Lee, H.-K., and Kim, J.-H. An hmm-based threshold model approach for gesture recognition. *IEEE Transactions on pattern analysis and machine intelligence 21*, 10 (1999), 961–973.

[129] Li, S.-Z., Yu, B., Wu, W., Su, S.-Z., and Ji, R.-R. Feature learning based on sae–pca network for human gesture recognition in rgbd images. *Neurocomputing 151* (2015), 565–573.

[130] Li, X., Dai, H., Cui, L., and Wang, Y. Sonicoperator: Ultrasonic gesture recognition with deep neural network on mobiles. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)* (2017), IEEE, pp. 1–7.

[131] Li, Y. Protractor: a fast and accurate gesture recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), ACM, pp. 2169–2172.

[132] Liang, H., Yuan, J., Thalmann, D., and Thalmann, N. M. Ar in hand: Egocentric palm pose tracking and gesture recognition for augmented reality applications. In *Proceedings of the 23rd ACM international conference on Multimedia* (2015), ACM, pp. 743–744.

[133] Lin, H.-I., Hsu, M.-H., and Chen, W.-K. Human hand gesture recognition using a convolution neural network. In *Automation Science and Engineering (CASE), 2014 IEEE International Conference on* (2014), IEEE, pp. 1038–1043.

[134] Liu, J., Yu, K., Zhang, Y., and Huang, Y. Training conditional random fields using transfer learning for gesture recognition. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on* (2010), IEEE, pp. 314–323.

[135] Liu, K., Chen, C., Jafari, R., and Kehtarnavaz, N. Multi-hmm classification for hand gesture recognition using two differing modality sensors. In *Circuits and Systems Conference (DCAS), 2014 IEEE Dallas* (2014), IEEE, pp. 1–4.

[136] Liu, W., Fan, Y., Lei, T., and Zhang, Z. Human gesture recognition using orientation segmentation feature on random rorest. In *Signal and Information Processing (ChinaSIP), 2014 IEEE China Summit & International Conference on* (2014), IEEE, pp. 480–484.

[137] Liu, X., Sacks, J., Zhang, M., Richardson, A. G., Lucas, T. H., and Van der Spiegel, J. The virtual trackpad: An electromyography-based, wireless, real-time, low-power, embedded hand-gesture-recognition system using an event-driven artificial neural network. *IEEE Trans. Circuits Syst. II Express Briefs 64* (2017), 1257–1261.

[138] Louppe, G. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502* (2014).

[139] Lu, W.-L., and Little, J. J. Simultaneous tracking and action recognition using the pca-hog descriptor. In *null* (2006), IEEE, p. 6.

[140] Lucchese, G., Field, M., Ho, J., Gutierrez-Osuna, R., and Hammond, T. Gesturecommander: continuous touch-based gesture prediction. In *CHI'12 Extended Abstracts on Human Factors in Computing Systems* (2012), ACM, pp. 1925–1930.

[141] Luong, D. D., Lee, S., and Kim, T.-S. Human computer interface using the recognized finger parts of hand depth silhouette via random forests. In *Control, Automation and Systems (ICCAS), 2013 13th International Conference on* (2013), IEEE, pp. 905–909.

[142] Ma, L., and Huang, W. A static hand gesture recognition method based on the depth information. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2016 8th International Conference on* (2016), vol. 2, IEEE, pp. 136–139.

[143] Ma, Y., Zhou, G., Wang, S., Zhao, H., and Jung, W. Signfi: Sign language recognition using wifi. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2*, 1 (2018), 23.

[144] Maes, P., Darrell, T., Blumberg, B., and Pentland, A. The alive system: Wireless, full-body interaction with autonomous agents. *Multimedia systems 5*, 2 (1997), 105–112.

[145] Mandic, D. P., and Chambers, J. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. John Wiley & Sons, Inc., 2001.

[146] Maraqa, M., and Abu-Zaiter, R. Recognition of arabic sign language (arsl) using recurrent neural networks. In *Applications of Digital Information and Web Technologies, 2008. ICADIWT 2008. First International Conference on the* (2008), IEEE, pp. 478–481.

[147] Marcel, S., Bernier, O., Viallet, J.-E., and Collobert, D. Hand gesture recognition using input-output hidden markov models. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on* (2000), IEEE, pp. 456–461.

[148] Marin, G., Dominio, F., and Zanuttigh, P. Hand gesture recognition with leap motion and kinect devices. In *Image Processing (ICIP), 2014 IEEE International Conference on* (2014), IEEE, pp. 1565–1569.

[149] Martin, J., and Crowley, J. L. An appearance-based approach to gesture-recognition. In *International Conference on Image Analysis and Processing* (1997), Springer, pp. 340–347.

[150] McCulloch, W. S., and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*, 4 (1943), 115–133.

[151] Medsker, L., and Jain, L. C. *Recurrent neural networks: design and applications*. CRC press, 1999.

[152] Mezari, A., and Maglogiannis, I. Gesture recognition using symbolic aggregate approximation and dynamic time warping on motion data. In *Proceedings of the 11th EAI International Conference on Pervasive Computing Technologies for Healthcare* (2017), ACM, pp. 342–347.

[153] MIIRE research group. Shrec 2017, 2017. `http://www-rech.telecom-lille.fr/shrec2017-hand/`, Last accessed on 2018-09-23.

[154] Millar, R. B. *Maximum likelihood estimation and inference: with examples in R, SAS and ADMB*, vol. 111. John Wiley & Sons, 2011.

[155] Min, B.-W., Yoon, H.-S., Soh, J., Yang, Y.-M., and Ejima, T. Hand gesture recognition using hidden markov models. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on* (1997), vol. 5, IEEE, pp. 4232–4235.

[156] Mitra, S., and Acharya, T. Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 37*, 3 (2007), 311–324.

[157] Mo, S., Cheng, S., and Xing, X. Hand gesture segmentation based on improved kalman filter and tsl skin color model. In *Multimedia Technology (ICMT), 2011 International Conference on* (2011), IEEE, pp. 3543–3546.

[158] Modler, P., and Myatt, T. Recognition of separate hand gestures by time-delay neural networks based on multi-state spectral image patterns from cyclic hand movements. In *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on* (2008), IEEE, pp. 1539–1544.

[159] Mohamed, A. S. A., Soter, E. B., Singh, A., and Ruhaiyem, N. I. R. Gesture based help identification for hospital & elderlycare using dynamic time warping: A systematic study. In *Proceedings of the International Conference on Video and Image Processing* (2017), ACM, pp. 94–98.

[160] Molchanov, P., Gupta, S., Kim, K., and Kautz, J. Hand gesture recognition with 3d convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (2015), pp. 1–7.

[161] Molchanov, P., Yang, X., Gupta, S., Kim, K., Tyree, S., and Kautz, J. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4207–4215.

[162] Moschetti, A., Fiorini, L., Esposito, D., Dario, P., and Cavallo, F. Toward an unsupervised approach for daily gesture recognition in assisted living applications. *IEEE Sensors Journal 17*, 24 (2017), 8395–8403.

[163] Mulder, A. Hand gestures for hci. *Hand Centered Studies of Human Movement Project, Technical Report* (1996), 96–1.

[164] Müller, M. *Information retrieval for music and motion*, vol. 2. Springer, 2007.

[165] Muñoz-Salinas, R., Medina-Carnicer, R., Madrid-Cuevas, F. J., and Carmona-Poyato, A. Depth silhouettes for gesture recognition. *Pattern Recognition Letters 29*, 3 (2008), 319–329.

[166] Murakami, K., and Taguchi, H. Gesture recognition using recurrent neural networks. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (1991), ACM, pp. 237–242.

[167] Myers, C. S., and Rabiner, L. R. A comparative study of several dynamic time-warping algorithms for connected-word recognition. *Bell System Technical Journal 60*, 7 (1981), 1389–1409.

[168] Naguri, C. R., and Bunescu, R. C. Recognition of dynamic hand gestures from 3d motion data using lstm and cnn architectures. In *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on* (2017), IEEE, pp. 1130–1133.

[169] Natarajan, P., and Nevatia, R. Online, real-time tracking and recognition of human actions. In *Proceedings of WMVC 2008* (2008), IEEE, pp. 1–8.

[170] Natarajan, P., Singh, V. K., and Nevatia, R. Learning 3d action models from a few 2d videos for view invariant action recognition. In *Proceedings of CVPR 2010* (2010), IEEE, pp. 20006–2013.

[171] Neverova, N., Wolf, C., Paci, G., Sommavilla, G., Taylor, G., and Nebout, F. A multi-scale approach to gesture detection and recognition. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (2013), pp. 484–491.

[172] Neverova, N., Wolf, C., Taylor, G. W., and Nebout, F. Multi-scale deep learning for gesture detection and localization. In *Workshop at the European conference on computer vision* (2014), Springer, pp. 474–490.

[173] Ng, C. W., and Ranganath, S. Gesture recognition via pose classification. In *icpr* (2000), IEEE, p. 3703.

[174] Norman, D. A. Natural user interfaces are not natural. *interactions 17*, 3 (May 2010), 6–10.

[175] Normani, N., Urru, A., Abraham, L., Walsh, M., Tedesco, S., Cenedese, A., Susto, G. A., and O Flynn, B. A machine learning approach for gesture recognition with a lensless smart sensor system.

[176] Nowlan, S. J., and Platt, J. C. A convolutional neural network hand tracker. *Advances in neural information processing systems* (1995), 901–908.

[177] of Padova, U. Hand Gesture Dataset, university of padova microsoft kinect and leap motion, 2015. `http://lttm.dei.unipd.it/downloads/gesture/kinect_leap`, Last accessed on 2018-09-27.

[178] of Twente, U. CoST, corpus of social touch dataset, 2015. `https://data.4tu.nl/repository/uuid:5ef62345-3b3e-479c-8e1d-c922748c9b29`, Last accessed on 2018-09-27.

[179] of Twente, U. HAART, human-animal affective robot touch, 2017. `http://www.cs.ubc.ca/labs/spin/data/HAART20DataSet.zip`, Last accessed on 2018-09-27.

[180] Okada, S., and Otsuka, K. Recognizing words from gestures: Discovering gesture descriptors associated with spoken utterances. In *Automatic Face & Gesture Recognition (FG 2017), 2017 12th IEEE International Conference on* (2017), IEEE, pp. 430–437.

[181] Olsen, Jr., D. R. Evaluating user interface systems research. In *Proceedings of UIST 2007* (New York, NY, USA, 2007), UIST '07, ACM, pp. 251–258.

[182] Park, J., and Cho, S. H. Ir-uwb radar sensor for human gesture recognition by using machine learning. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on* (2016), IEEE, pp. 1246–1249.

[183] Park, T., Lee, J., Hwang, I., Yoo, C., Nachman, L., and Song, J. E-gesture: a collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems* (2011), ACM, pp. 260–273.

[184] Pittman, C. R., and LaViola Jr, J. J. Multiwave: Complex hand gesture recognition using the doppler effect. In *Proceedings of the 43rd Graphics Interface Conference* (2017), Canadian Human-Computer Communications Society, pp. 97–106.

[185] Platt, J. C., Cristianini, N., and Shawe-Taylor, J. Large margin dags for multiclass classification. In *Advances in neural information processing systems* (2000), pp. 547–553.

[186] Pugeault, N., and Bowden, R. Spelling it out: Real-time asl fingerspelling recognition. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on* (2011), IEEE, pp. 1114–1119.

[187] Rabiner, L. R. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE 77*, 2 (1989), 257–286.

[188] Rabiner, L. R., Wilpon, J. G., and Juang, B.-H. A segmental k-means training procedure for connected word recognition. *AT&T technical journal 65*, 3 (1986), 21–31.

[189] Rakowski, A., and Wandzik, L. Hand shape recognition using very deep convolutional neural networks. In *Proceedings of the 2018*

International Conference on Control and Computer Vision (2018), ACM, pp. 8–12.

[190] Rautaray, S. S., and Agrawal, A. Vision based hand gesture recognition for human computer interaction: a survey. *Artif. Intell. Rev. 43*, 1 (2015), 1–54.

[191] Reyes-Ortiz, J.-L., Oneto, L., Samà, A., Parra, X., and Anguita, D. Transition-aware human activity recognition using smartphones. *Neurocomputing 171* (2016), 754–767.

[192] Ripley, B. D. *Pattern recognition and neural networks*. Cambridge university press, 2007.

[193] Rovelo, G., Degraen, D., Vanacken, D., Luyten, K., and Coninx, K. Gestu-wan-an intelligible mid-air gesture guidance system for walk-up-and-use displays. In *Human-Computer Interaction* (2015), Springer, pp. 368–386.

[194] Ruan, X., and Tian, C. Dynamic gesture recognition based on improved dtw algorithm. In *Mechatronics and Automation (ICMA), 2015 IEEE International Conference on* (2015), IEEE, pp. 2134–2138.

[195] Rubine, D. *The automatic recognition of gestures*. PhD thesis, Citeseer, 1991.

[196] Russell, S. J., and Norvig, P. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[197] Sagayam, K. M., and Hemanth, D. J. Abc algorithm based optimization of 1-d hidden markov model for hand gesture recognition applications. *Computers in Industry 99* (2018), 313–323.

[198] Schneider, P., and Eberly, D. H. *Geometric tools for computer graphics*. Elsevier, 2002.

[199] Scholliers, C., Hoste, L., Signer, B., and De Meuter, W. Midas: a declarative multi-touch interaction framework. In *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction* (2011), ACM, pp. 49–56.

[200] Schreiber, J. pomegranate. `https://github.com/jmschrei/pomegranate`, 2016.

[201] Schuldt, C., Laptev, I., and Caputo, B. Recognizing human actions: a local svm approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on* (2004), vol. 3, IEEE, pp. 32–36.

[202] Schwarz, J., Mankoff, J., and Hudson, S. Monte carlo methods for managing interactive state, action and feedback under uncertainty. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (2011), ACM, pp. 235–244.

[203] Schwarz, J., Mankoff, J., and Hudson, S. E. An architecture for generating interactive feedback in probabilistic user interfaces. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (2015), ACM, pp. 2545–2554.

[204] Serra, G., Camurri, M., Baraldi, L., Benedetti, M., and Cucchiara, R. Hand segmentation for gesture recognition in ego-vision. In *Proceedings of the 3rd ACM international workshop on Interactive multimedia on mobile & portable devices* (2013), ACM, pp. 31–36.

[205] Sezgin, T. M., and Davis, R. Hmm-based efficient sketch recognition. In *Proceedings of the 10th international conference on Intelligent user interfaces* (2005), ACM, pp. 281–283.

[206] Shalev-Shwartz, S., and Ben-David, S. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[207] Shin, S., and Sung, W. Dynamic hand gesture recognition for wearable devices with low complexity recurrent neural networks. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on* (2016), IEEE, pp. 2274–2277.

[208] Simão, M. A., Neto, P., and Gibaru, O. Unsupervised gesture segmentation of a real-time data stream in matlab. In *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE* (2016), IEEE, pp. 809–814.

[209] Simão, M. A., Neto, P., and Gibaru, O. Unsupervised gesture segmentation by motion detection of a real-time data stream. *IEEE Transactions on Industrial Informatics 13*, 2 (2017), 473–481.

[210] Smith, K. A., Csech, C., Murdoch, D., and Shaker, G. Gesture recognition using mm-wave sensor for human-car interface. *IEEE Sensors Letters 2*, 2 (2018), 1–4.

[211] Smola, A., and Vishwanathan, S. Introduction to machine learning. *Cambridge University, UK 32* (2008), 34.

[212] Sodhi, R., Benko, H., and Wilson, A. Lightguide: projected visualizations for hand movement guidance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2012), ACM, pp. 179–188.

[213] Song, J., Sörös, G., Pece, F., Fanello, S. R., Izadi, S., Keskin, C., and Hilliges, O. In-air gestures around unmodified mobile devices. In *Proceedings of the 27th annual ACM symposium on User interface software and technology* (2014), ACM, pp. 319–329.

[214] Song, Y., Demirdjian, D., and Davis, R. Tracking body and hands for gesture recognition: Natops aircraft handling signals database. In *Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on* (2011), IEEE, pp. 500–506.

[215] Spano, L. D., Cisternino, A., and Paternò, F. A Compositional Model for Gesture Definition. In *Proceedings of HCSE 2012* (2012), Springer, pp. 34–52.

[216] Spano, L. D., Cisternino, A., Paternò, F., and Fenu, G. GestIT: a Declarative and Compositional Framework for Multiplatform Gesture Definition. In *Proceedings of EICS 2013* (2013), ACM, pp. 187–196.

[217] Starner, T., and Pentland, A. Real-time american sign language recognition from video using hidden markov models. In *Motion-Based Recognition*. Springer, 1997, pp. 227–243.

[218] Steinwart, I., and Christmann, A. *Support vector machines*. Springer Science & Business Media, 2008.

[219] Strezoski, G., Stojanovski, D., Dimitrovski, I., and Madjarov, G. Hand gesture recognition using deep convolutional neural networks. In *International Conference on ICT Innovations* (2016), Springer, pp. 49–58.

[220] Suk, H.-I., Sin, B.-K., and Lee, S.-W. Hand gesture recognition based on dynamic bayesian network framework. *Pattern recognition 43*, 9 (2010), 3059–3072.

[221] Tao, M., and Ma, L. A hand gesture recognition model based on semi-supervised learning. In *Intelligent Human-Machine Systems*

and Cybernetics (IHMSC), 2015 7th International Conference on (2015), vol. 2, IEEE, pp. 43–46.

[222] Tappert, C. C. Cursive script recognition by elastic matching. *IBM Journal of Research and development 26*, 6 (1982), 765–771.

[223] Tappert, C. C., Suen, C. Y., and Wakahara, T. The state of the art in online handwriting recognition. *IEEE Transactions on pattern analysis and machine intelligence 12*, 8 (1990), 787–808.

[224] Taranta II, E. M., Samiei, A., Maghoumi, M., Khaloo, P., Pittman, C. R., and LaViola Jr, J. J. Jackknife: A reliable recognizer with few samples and many modalities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017), ACM, pp. 5850–5861.

[225] ten Holt, G. A., Reinders, M. J., and Hendriks, E. Multi-dimensional dynamic time warping for gesture recognition. In *Thirteenth annual conference of the Advanced School for Computing and Imaging* (2007), vol. 300, p. 1.

[226] Theodoridis, S., Koutroumbas, K., et al. Pattern recognition. *IEEE Transactions on Neural Networks 19*, 2 (2008), 376.

[227] Tsai, W.-J., Chen, J.-C., and Lin, K. W. Depth-based hand pose segmentation with hough random forest. In *Green Technology and Sustainable Development (GTSD), International Conference on* (2016), IEEE, pp. 166–167.

[228] Vamsikrishna, K., Dogra, D. P., and Desarkar, M. S. Computer-vision-assisted palm rehabilitation with supervised learning. *IEEE Transactions on Biomedical Engineering 63*, 5 (2016), 991–1001.

[229] Vapnik, V. *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

[230] Vatavu, R.-D. 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* (2017), ACM, pp. 4667–4679.

[231] Vatavu, R.-D., Anthony, L., and Wobbrock, J. O. Gestures as point clouds: a $ p recognizer for user interface prototypes. In *Proceedings of the 14th ACM international conference on Multimodal interaction* (2012), ACM, pp. 273–280.

[232] Vatavu, R.-D., Anthony, L., and Wobbrock, J. O. $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* (New York, NY, USA, 2018), MobileHCI 2018, ACM, pp. 23:1–23:12.

[233] Vermeulen, J., Luyten, K., van den Hoven, E., and Coninx, K. Crossing the bridge over norman's gulf of execution: revealing feedforward's true identity. In *Proceedings of CHI 2013* (2013), ACM, pp. 1931–1940.

[234] von Laban, R. *Modern educational dance*. Princeton Book Co Pub, 1975.

[235] Vu, T. H., Dang, A., Dung, L., and Wang, J.-C. Self-gated recurrent neural networks for human activity recognition on wearable devices. In *Proceedings of the on Thematic Workshops of ACM Multimedia 2017* (2017), ACM, pp. 179–185.

[236] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. Phoneme recognition using time-delay neural networks. In *Readings in speech recognition*. Elsevier, 1990, pp. 393–404.

[237] Walter, M., Psarrou, A., and Gong, S. Auto clustering for unsupervised learning of atomic gesture components using minimum description length. In *Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 2001. Proceedings. IEEE ICCV Workshop on* (2001), IEEE, pp. 157–162.

[238] Ware, C. *Information visualization: perception for design*. Elsevier, 2012.

[239] Willems, D., Niels, R., van Gerven, M., and Vuurpijl, L. Iconic and multi-stroke gesture recognition. *Pattern Recognition 42*, 12 (2009), 3303–3312.

[240] Wilson, A. D. Robust computer vision-based detection of pinching for one and two-handed gesture input. In *Proceedings of the 19th annual ACM symposium on User interface software and technology* (2006), ACM, pp. 255–258.

[241] Wobbrock, J. O., Wilson, A. D., and Li, Y. 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* (New York, NY, USA, 2007), UIST '07, ACM, pp. 159–168.

[242] Won, K.-J., Prugel-Bennett, A., and Krogh, A. Evolving the structure of hidden markov models. *IEEE Transactions on Evolutionary Computation 10*, 1 (2006), 39–49.

[243] Xochicale, M., Baber, C., and Oussalah, M. Understanding movement variability of simplistic gestures using an inertial sensor. In *Proceedings of the 5th ACM International Symposium on Pervasive Displays* (2016), ACM, pp. 239–240.

[244] Xu, S., and Xue, Y. A long term memory recognition framework on multi-complexity motion gestures. In *Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on* (2017), vol. 1, IEEE, pp. 201–205.

[245] Yamato, J., Ohya, J., and Ishii, K. Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on* (1992), IEEE, pp. 379–385.

[246] Yang, J., Yuan, J., and Li, Y. Parsing 3d motion trajectory for gesture recognition. *Journal of Visual Communication and Image Representation 38* (2016), 627–640.

[247] Yang, M.-H., and Ahuja, N. Recognizing hand gestures using motion trajectories. In *Face Detection and Gesture Recognition for Human-Computer Interaction*. Springer, 2001, pp. 53–81.

[248] Yang, M.-H., Ahuja, N., and Tabb, M. Extraction of 2d motion trajectories and its application to hand gesture recognition. *IEEE Transactions on pattern analysis and machine intelligence 24*, 8 (2002), 1061–1074.

[249] Yang, Y., Saleemi, I., and Shah, M. Discovering motion primitives for unsupervised grouping and one-shot learning of human actions, gestures, and expressions. *IEEE transactions on pattern analysis and machine intelligence 35*, 7 (2013), 1635–1648.

[250] Yingxin, X., Jinghua, L., Lichun, W., and Dehui, K. A robust hand gesture recognition method via convolutional neural network. In *Digital Home (ICDH), 2016 6th International Conference on* (2016), IEEE, pp. 64–67.

[251] Yoon, B.-J. Hidden markov models and their applications in biological sequence analysis. *Current genomics 10*, 6 (2009), 402–415.

[252] Yu, M.-C., Yu, T., Wang, S.-C., Lin, C.-J., and Chang, E. Y. Big data small footprint: the design of a low-power classifier for detecting transportation modes. *Proceedings of the VLDB Endowment 7*, 13 (2014), 1429–1440.

[253] Yuan, Y., and Barner, K. Hybrid feature selection for gesture recognition using support vector machines. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on* (2008), IEEE, pp. 1941–1944.

[254] Yuan, Y., Liu, Y., and Barner, K. Tactile gesture recognition for people with disabilities. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on* (2005), vol. 5, IEEE, pp. v–461.

[255] Zhai, S., and Kristensson, P.-O. Shorthand writing on stylus keyboard. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (2003), ACM, pp. 97–104.

[256] Zhang, Y., Chen, Y., Yu, H., Yang, X., Lu, W., and Liu, H. Wearing-independent hand gesture recognition method based on emg armband. *Personal and Ubiquitous Computing 22*, 3 (2018), 511–524.

[257] Zhang, Z., Tian, Z., and Zhou, M. Latern: Dynamic continuous hand gesture recognition using fmcw radar sensor. *IEEE Sensors Journal 18*, 8 (2018), 3278–3289.

[258] Zhao, Y., Liu, Y., Dong, M., and Bi, S. Multi-feature gesture recognition based on kinect. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2016 IEEE International Conference on* (2016), IEEE, pp. 392–396.

[259] Zimmerman, T. G., Lanier, J., Blanchard, C., Bryson, S., and Harvill, Y. A hand gesture interface device. In *ACM SIGCHI Bulletin* (1987), vol. 18, ACM, pp. 189–192.

[260] Zucchini, W., MacDonald, I. L., and Langrock, R. *Hidden Markov models for time series: an introduction using R*. Chapman and Hall/CRC, 2016.