DEICTIC: a Compositional and Declarative Gesture Description based on Hidden Markov Models

Alessandro Carcangiu[a,*], Lucio Davide Spano[b], Giorgio Fumera[a], Fabio Roli[a]

[a]*Department of Electric and Electronic Engineering, University of Cagliari via Marengo 2, 09123, Cagliari Italy*
[b]*Department of Mathematics and Computer Science, University of Cagliari, via Ospedale 72, 07124, Cagliari, Italy*

Abstract

The consumer-level devices that track the 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 focus mainly on accuracy and robustness. The resulting classifiers label precisely gestures after their performance, but they do not provide intermediate information during the 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 paper, we introduce DEICTIC, a compositional and declarative description for stroke gestures, which 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, evaluated on two datasets from the literature. Through a developer evaluation, we show that the implementation of a guidance system with DEICTIC requires an effort comparable to compositional approaches, while the definition procedure and the perceived recognition accuracy is comparable to machine learning.

Keywords: Gestures, Classification, Hidden Markov Models, Compositional gesture modelling, Declarative gesture modelling,

1. Introduction

The availability of consumer-level devices for tracking the user's movements fostered the design and deployment of interactive systems relying on gestural

*Corresponding author
Email address: `alessandro.carcangiu@diee.unica.it` (Alessandro Carcangiu)

input. Some examples are the Nintendo Wiimote, Microsoft Kinect (version 1 and 2) and, more recently, Leap Motion, Intel RealSense, and all the newest controllers shipped with virtual reality headsets (e.g., Oculus Rift and HTC Vive). Such hardware supports the adoption of gestural interaction in different scenarios e.g., exhibitions, museums and public spaces and, above all, in the entertainment field.

Similarly to other interaction modalities, the effectiveness of gestural interaction relies on how it supports the communication between the user and the application. On the one hand, the machine requires interpreting correctly the user's input, recognizing accurately her movements. On the other hand, the user must be aware of which movements are available for communicating with the system.

The 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) [1, 2, 3], as well as traditional supervised classification algorithms like support vector machines (although they are more suited to static gestures [1]). 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. It concluded that the interface should provide two pieces of information to the user during the gestural interaction: the *feedback* and the *feedforward* [4], which may be designed taking into account different options [5]. 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 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.

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 [6, 7, 8, 9, 10], and they allow receiving events for the recognition

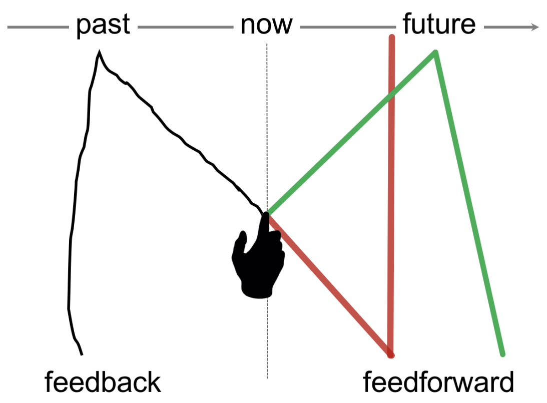


Figure 1: A simple guidance system for stroke gestures for a touch interface. It shows the previous touch positions with a black line (feedback) and the possible completions (feedforward). In this example, the system understands either an N stroke (in red) or M (in green).

of a whole gesture and its sub-parts. These approaches support effectively the development of guidance systems in a UI. However, the sub-part identification is currently achieved through geometric heuristics that 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.

In this paper, we fill the gap between these two fields of research. We define a declarative and compositional model for describing stroke gestures according to primitives (ground terms) and a set of compositional operators (iterative, sequence, parallel and choice) that leveraged on existing definition in the literature [6, 7], and it reaches an accuracy comparable with the state-of-the-art approaches in gesture recognition while supporting the sub-gesture identification.

For achieving such results, we created a set of algorithms taking as input the model of a complex gesture and returns a composite classifier, which connects basic HMMs recognizing the gesture primitives. The resulting composite HMM supports the recognition of the whole gesture and its sub-parts: the composition associates a specific sub-part to each HMM state, so the Viterbi algorithm [30] provides both the most likely state and sub-part sequence. In addition, we exploit the same property for predicting the most likely completion for a partial gesture through the forward algorithm [28]. This constitutes a novel approach to the problem, combining the advantages of classifiers providing an accurate and robust recognition together with the inspectability property of the declarative approaches.

The method, called DEICTIC (DEclaratIve and CompösiTional Input Classifier), maps the composite HMM states to the underlying primitives which constitute it. For each sample of the gesture sequence it identifies the most likely list of primitives that the user already performed (useful for providing *feedback* in Figure 1) and the most likely primitive that the user will perform in the future (useful for providing *feedforward* in Figure 1).

We validate the proposed methodology showing its accuracy on two datasets from the literature (the 1$ [11] and the N$ [12]), which is comparable with state-of-the-art approaches. In addition, we prove the advantages in developing a guidance system for a gestural UI through a developer evaluation.

The paper is organised as follows[1]: in Section 2 we summarise the existing approaches in the literature for recognising gestures and using them in the UI development; in Section 3 we introduce the gesture modelling syntax; in Section 4 we describe the algorithm for composing HMMs; in Section 5 we summarise the advantages and limitations of the approach for the UI development; in Section 6 we evaluate DEICTIC against the heuristic and machine learning classification approaches with developers; in Section 7 we measure the accuracy on two stroke gesture datasets; in Section 8 we discuss the conclusions and future work.

2. Background and Related work

In this section, we discuss a set of approaches for recognising gestures able to identify their sub-parts. The definition of gesture sub-part varies in different research fields. We will first introduce the classification approaches most relevant to our work, then we detail the main approaches based on declarative models.

2.1. Classification approaches using primitives

Classification methods that identify a set of sub-parts (or primitives) common to different gestures have already been proposed, either for increasing the recognition rate or to reduce the training set size in learning-based approaches. 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, 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 for each interpretation of the primitive concept.

In [14] 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 also for building

[1]The paper is an extended version of the work submitted in [13], which presents only the overall idea and it does not discuss i) the definition of the stroke gesture modelling language; ii) the composition algorithms for building composite HMMs; iii) the generation of a composite HMM from a gesture expression; iv) the accuracy evaluation against state of the art techniques with well-known datasets; v) how the gesture modelling and recognition technique may be applied for supporting feedback and feedforward in gesture interfaces and vi) the developer evaluation.

UIs. However, since primitives are identified automatically, they are difficult to understand for designers while creating feedback and feedforward systems. In [15] 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, they may have problems in identifying those easily distinguishable from each other, since usually they do not have a clear understanding of the underlying classification algorithms.

In [16] primitives are used 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.

A set of primitives that suits better the designer's understanding includes 3D properties of the movement trajectory. For instance, in [17] primitives identified in a 2D video are used for classifying 3D movements. Here the primitives are functions on the 2D features that represent the user's state. Holte et al. [18] propose 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.

To our knowledge, Kim et al. [19] proposed the most similar approach to the work discussed in this paper. 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. This technique is valid for describing stroke sequences, which corresponds to the *sequence* operator in our modelling language. Our approach 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 [19] requires a re-training step with samples of the complete gesture for avoiding degradation in the recognition performance. Our approach does not need this step, as we show in section 7.

2.2. Gesture description models

In this section, we summarise different compositional approaches based on heuristic gesture recognition. We point out that none of them includes a formal evaluation of the recognition accuracy.

Kammer et al. [20] introduced GeForMT, a multitouch gesture formalisation language that tries to fill the gap between the high-level complex-gestures (such as pinch to zoom) and the low-level device events, using an Extended Backus-Naur form grammar. Basic touches represent the basic movements (move, point, hold, line, circle and semicircle), composed using different operators (parallel and sequential). The model tracks the objects and the area manipulated in the interface.

Scholliers et al. [21] defined Midas, an architecture for recognising gestures according to a set of rules, which are matched against a set of input facts by a logical rule inference engine. The rules work on different features, such as the 2D positions and speed, and the tracking state (appear, move and disappear). Each rule consists of two components: a prerequisite part and an action part. The first defines the input fact pattern to be recognised while the second the UI behaviour. Mudra [22], a follow-up research from the same group, extends Midas for multimodal interfaces. It unifies the input stream coming from different devices, exploiting different modalities. Designers define both the low-level handling events and the high-levels rules, combining them into a single software architecture.

Khandkar et al. proposed GDL [23] (Gesture Description Language), a domain-specific language designed to streamline the process of defining gestures. GDL separates the gesture recognition code from the definition of the UI behaviour. This work defines three components: the gesture name, the code for the gesture validation and a return type. The last component represents the data notified with a callback to the application logic.

More structured and expressive declarative methods are Proton++ [8] and GestIT [7, 6], 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 multitouch framework allowing developers to declaratively describe custom gestures, separating the temporal sequencing of the events from the code related to the UI behaviour. Multitouch gestures are defined as regular expressions, where literals are identified by a triple composed of: i) the event type (e.g., touch down, move and up), ii) the touch identifier (e.g., 1 for the first finger, 2 for the second etc.), iii) the object hit by the touch (e.g., the background, a particular shape etc.). It allows developers to declaratively describe custom gestures through regular expressions, using the concatenation, alternation and Kleene's star operators. An improved version of the framework (presented in [9]) 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 accepted by the heuristic e.g., it moves north, north-west, south etc.

GestIT [7, 6] follows a similar approach, including operators for defining more advanced gestures. As discussed in [6], 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. A ground term is the smallest block for defining a gesture: it describes an atomic event which cannot be further decomposed. In general, 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. Composite terms are used for defining more complex gestures through a set of operators (iteration, sequence, parallel, choice and disabling). We will use them for the rest of this work since they are a superset of those included in Proton++ [6]. As a simple example, we define here a grab gesture for selecting an object in the user interface closing a hand (equation [1]). It begins with an iterative movement of the right hand (mH_r^*), which models the pointing before the object selection, which is interrupted (the first disabling operator) by a change of the hand state to *closed* (sH_r[closed]). After that (the sequence operator), the user can move the closed hand for e.g., moving the selected object (mH_r^*) and the gesture terminates (the second disabling operator) when the user reopens the hand (sH_r[open]).

$$(mH_r^* \; [> sH_r[\text{closed}]) \gg (mH_r^* \; [> sH_r[\text{open}]) \tag{1}$$

(Grab gesture)

In this work, we start from the modelling experience we acquired in defining GestIT and we try to fix its main drawback, which is 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 in this field 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.

3. Gesture Description

In this section, we introduce a simple modelling language for stroke gestures that, following an approach similar to other declarative approaches (e.g., GestIT [7, 6]). 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.

3.1. 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.

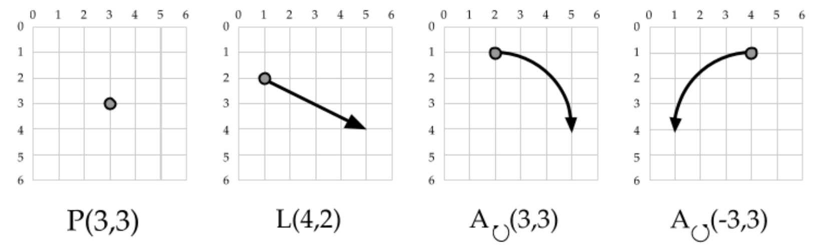


Figure 2: DEICTIC sample primitives. The first defines a stroke starting at (3, 3); the second defines a line that, assuming that the current position is (1,2), moves 4 units in the x and 2 units on the y axis; the third defines an arc that, starting from the position (2,1) moves 3 unit on both axes, 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.

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 2 shows a graphical representation of the four primitives. The current position is represented by a filled circle, the stroke path with an arrow.

3.2. 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 4.3.

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 $E_1|E_2|\dots|E_n$ 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 4.5.

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 2: 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.

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

Parallel Operator. The expression $E_1 \times E_2 \times \dots \times E_n$ represents a set of strokes that can be performed at the same time. It is useful for modelling e.g., multi-touch 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.

3.3. 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 [11] and the N$ gesture dataset [12]. 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 7.

The first example we discuss is depicted in Figure 3, 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 followed 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 3 shows graphically the resulting ideal definition and a set of real executions of the considered gesture.

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

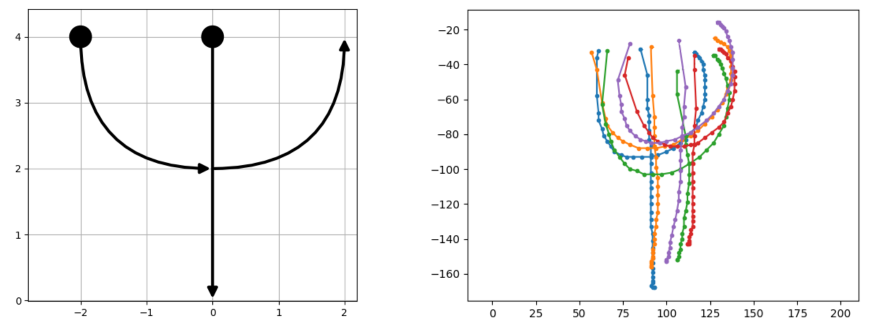


Figure 3: Modelling the pitchfork (ψ) gesture from the N$-dataset [12] with a DEICTIC expression. We show a graphical representation of the ideal model, together with 10 real gesture samples.

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 (◯), 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) in a video player, mimicking the interaction with a real handle. It is possible to iterate the sequence of arcs in ⏮, since it starts and it 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: she may start drawing the circle and then the pitchfork or vice-versa.

4. 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 tech-

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

$$\textbf{Rewind}: ⏮ = P(0,0) + (A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3) + A_{\circlearrowleft}(-3,3))^*$$

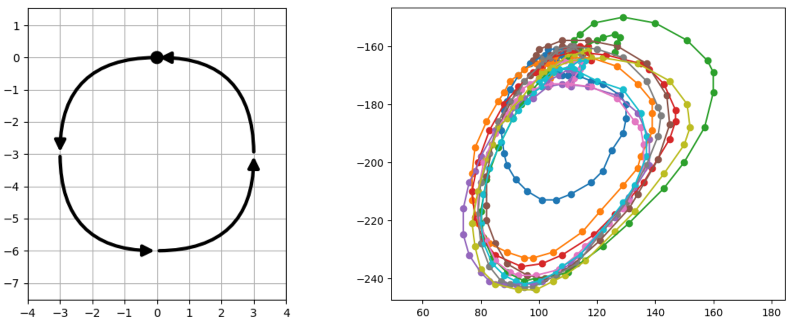


Figure 4: Modelling a circle gesture from the 1$-dataset [11] with a DEICTIC expression. We show a graphical representation of the ideal model, together with 10 real gesture samples.

niques. 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 first summarise the definition of a Hidden Markov Model, in order to point out the main properties we exploit in our work, and explaining the notation we use in this paper (for a complete description, please refer to [24]). Then, we discuss how we obtain an HMM 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.

4.1. Hidden Markov Models

A Hidden Markov Model (HMM) is a probabilistic model that maps a sequence of observations into a corresponding sequence of labels. They are special cases of weighted automata, defined through a finite set of states and a set of transitions with the associated weights. In the case of HMMs, such weights correspond to the probability of firing a transition. Therefore, considering a single state, the sum of the weights on all its transitions to any other state is 1. 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 $t-1$.

HMMs model events that are "hidden", i.e. not directly observable in the world, but influencing a set of observable events, which instead can be measured through sensors. In the gestural interaction domain, we do not observe directly the gesture performed by the user. Instead, we register a set of features tracked by an input device (e.g., a touchscreen or a Microsoft Kinect). 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.

Formally, an HMM $\lambda(S, V, A, B)$ can be defined by:

- A set of states S.
- A vocabulary of values for the observable events V
- 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.
- 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.

According to this definition, we can summarise the following properties:

$$\sum_{j=0}^{f} A_{ij} = 1 \qquad \forall i \tag{3}$$

(Outgoing probabilities)

$$P(s_i|s_1, s_2, \ldots, s_{i-1}) = P(s_i|s_{i-1}) \tag{4}$$

(Markov Assumption)

$$P(v_i|s_1, \ldots, s_i, \ldots, s_{i+k}, v_1, \ldots, v_i, \ldots, v_{i+k}) = P(v_i|s_i) \tag{5}$$

(Obvervation independence)

The equation 3 ensures that, in an HMM, the state transitions have well-defined probabilities; the equation 4 assumes that a state transition depends only on the current state; equation 5 expresses that an observed event value depends only on the current state.

4.2. *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 [19, 25], 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 paper, we use a Bakis topology for the ground terms, but the composition works also if different topologies are used (e.g., ergodic) [24].

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 supported by a well-known algorithm such as Viterbi learning [26] or Baum-Welch [27] (we use the latter in this work). The resulting HMM can be used for the recognition of a specific segment in different gesture definitions.

4.3. *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 5 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.

Algorithm 1 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).

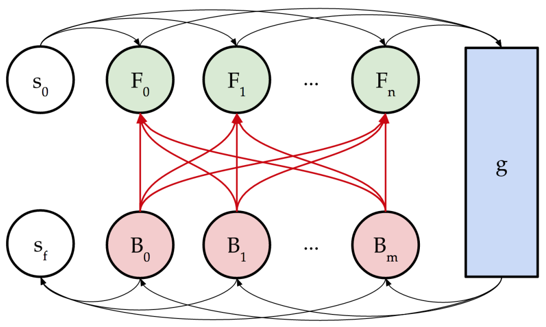


Figure 5: The topology of an HMM for the iterative operator. The s_0 is the starting state with its forward star $F_0 \dots F_n$; s_f is the final state with its backward star $B_0 \dots B_m$; the blue rectangle represents all the other states. The iterative operator is defined by the red arcs connecting each B_i with all F_j.

ALGORITHM 1: 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 $i$
```

4.4. 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 6: 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.

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 $g+h$ 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 2 shows how to define $g+h$ given an HMM recognising g and one recognising h.

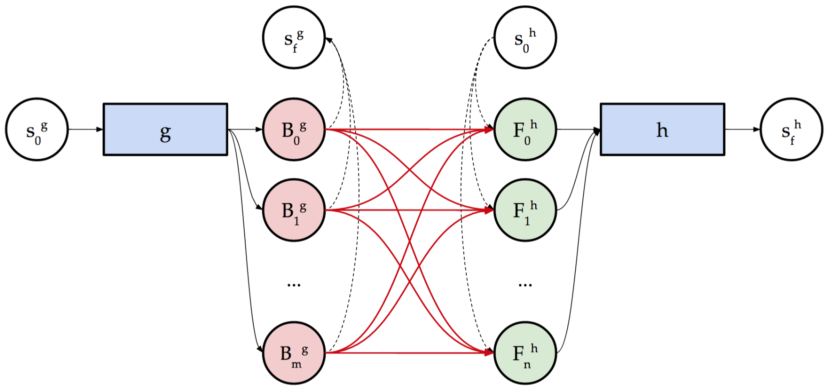


Figure 6: The topology of an HMM for the sequence operator. s_f^g is the final state of the HMM associated with g, while $B_0^g \dots B_m^g$ is its backward star; s_0^h is the starting state of the HMM associated with h, while $F_0^h \dots F_n^h$ is its forward star; the blue rectangles represent all the other states of both HMMs. The sequence operator is defined by the red arcs connecting each B_i with all F_j.

ALGORITHM 2: 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
$n \leftarrow |S^g| - 1$; $m \leftarrow |S^h| - 1$; ;
$s \leftarrow$ EmptyHMM$(n + m)$ ;
$S^s \leftarrow (S^g \setminus s^g_f) \cup (S^h \setminus s^h_0)$; $V^s \leftarrow V^g \cup V^h$; $A^s \leftarrow 0_{n+m,n+m}$ ;
$B^s \leftarrow$ CompleteDistributions ($B^g$, $B^h$, $V^s$, $V^g$, $V^h$, $n + m$);
/* copy the transitions from g                                        */
for $(u, v) \in A^g, v \neq s^g_f$ do $A^s_{u,v} \leftarrow A^g_{u,v}$ ;
/* copy the transitions from h                                        */
for $(u, v) \in A^h, v \neq s^h_0$ do $A^s_{u+n,v+n} \leftarrow A^h_{u,v}$ ;
/* connecting the ending states in g with the starting states in h   */
$\alpha \leftarrow$ count$((s^h_0, u) \in A^h)$ ;
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 ($B^g$, $B^h$, $V^s$, $V^g$, $V^h$,$n$)
    $B \leftarrow$ 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 $B_u[v] \leftarrow$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 $B_{u+n}[v] \leftarrow$unif$(v)$ ;
    end
    return $B$
```

4.5. 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 7: we connect the starting state with the forward state of both s^g_0 and s^h_0. 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^g_f and s^h_f are connected with the final state, with one-half of the original transition probability. We remove the arcs in the forward star of s^g_0 and s^h_0 together with those in the backward star of s^g_f and s^h_f.

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.

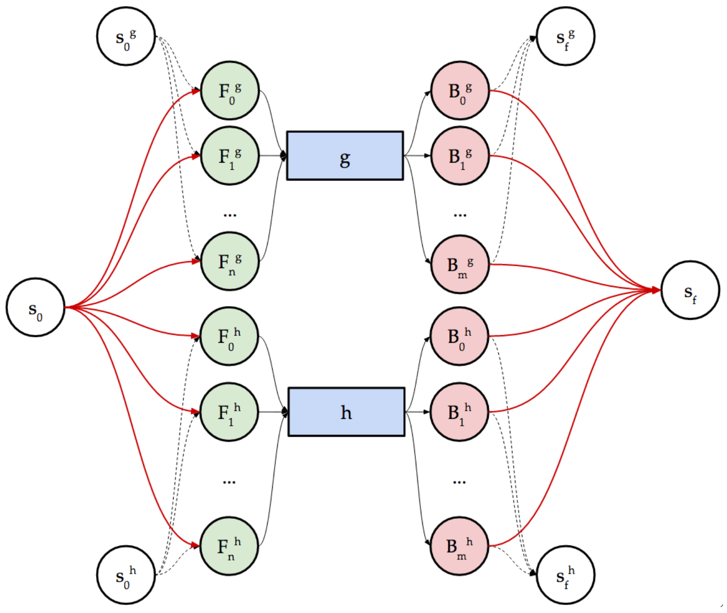


Figure 7: The topology of an HMM for a choice operator. The two original HMM are put in two separate recognition lines.

ALGORITHM 3: 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 $S^c$
```

4.6. 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_i^g, s_j^h) and (s_x^g, s_y^h), 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 4 summarizes the procedure for building the HMM. Figure 8 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.

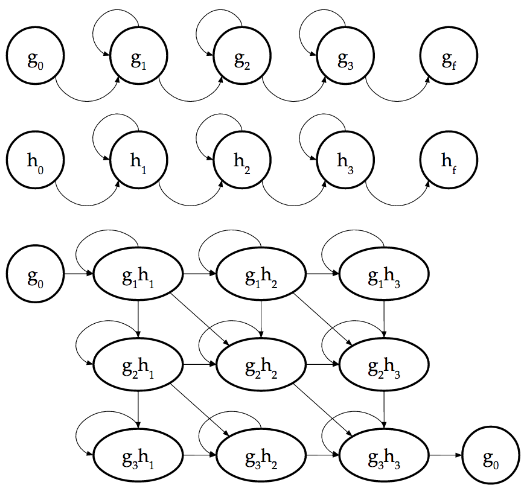


Figure 8: The topology of an HMM for a parallel operator, considering two Bakis terms

ALGORITHM 4: 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
$n \leftarrow |S^g| - 2$; $m \leftarrow |S^h| - 2$; ;
$c \leftarrow$ 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 $B^p_{(u,v)} \leftarrow$ 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 $S^p$
```

4.7. 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 [11]), 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 [27] 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 [28].

5. 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 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 [29] that allows writing gesture expressions similar to the ones reported in this paper for building classifiers through the operator overload. As we better detail in Section 7, 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 it. 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 [30].

From the mapping between states and ground terms derives another positive effect in developing gestural UIs. Given an HMM, we can predict the future distribution of its internal states through the forward algorithm [28]. 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: $\triangle = P(0,0) + L(-3,-4) + L(6,0) + L(-3,4)$

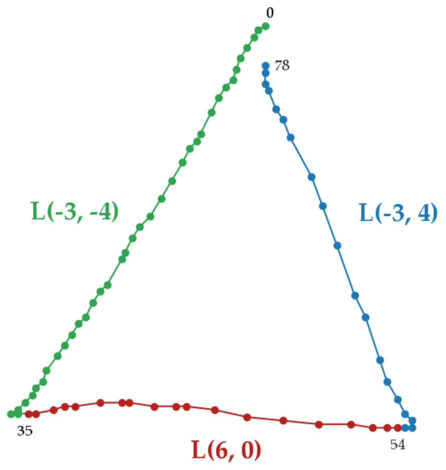


Figure 9: Ground term decomposition of a triangle gesture sample from the 1$-dataset. Each dot corresponds to a set of x, y coordinates of the sample point list. Points having a different colour corresponding to different ground terms.

Figure 9 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, where the user may execute gestures only inside the cells of a grid.

Our approach differs from other composition techniques on HMMs [19] 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 7 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 [28]. 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 HMM directly on complete gesture samples (which we call ad-hoc HMMs in this paper) 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.

6. Developer evaluation

In this section, we detail the results of a user study for assessing how the proposed method assists the UI developers in creating gestural interactions. We compare the DEICTIC recognition support against heuristic and machine-learning approaches. The small number of participants in this study does not allow us to provide conclusions on the robustness and accuracy of the approach, which is assessed through publicly available dataset in Section 7.

6.1. Participants

We recruited a group of 17 developers, 14 males and 3 females. Their education level ranged from the High School Degree (5), Bachelor Degree (3), Master Degree (5) and 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 10, left 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 10 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. They have to decide which one suits better the UI requirements.

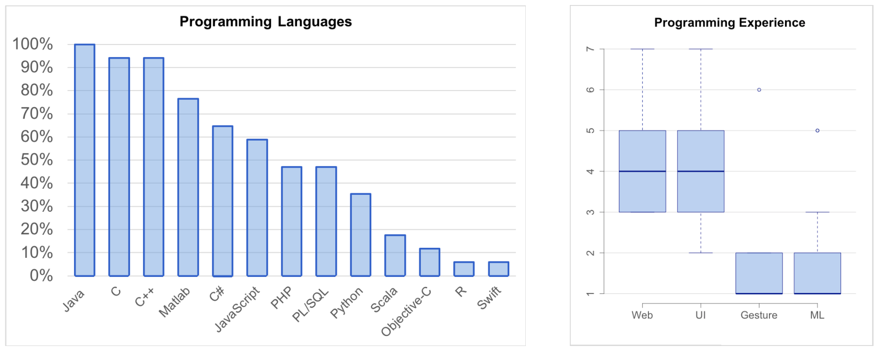


Figure 10: Participants' experience with programming languages and development task.

6.2. Procedure

The evaluation consisted in the development of a simple gestural interface for managing a 3×4 grid (see Figure 11, 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 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 supported 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 11), 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* [31], which shows a dynamic guidance while the user performs the stroke. Figure 11, 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 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 correspondent gesture). In Figure 11 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 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 work.

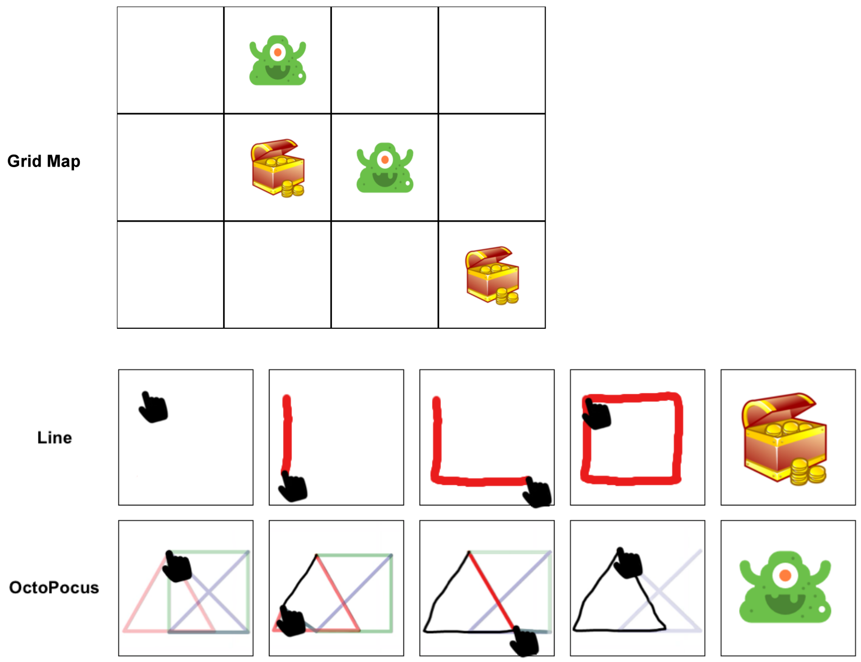


Figure 11: Gestural UI developed for the user test (top part). Line feedback for a square gesture (middle part) OctoPocus [31] feedback for a triangle gesture (bottom part).

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.[2] 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 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 [11], 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 paper, 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 5.

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

- *Grid*, which draws and manages the grid user interface, including functions for setting or resetting the cell content.

[2]For instance, considering the four segments of the square stroke in Figure 11, 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.

- *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 [29] [3].

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.

6.3. Evaluation metrics and success criteria

For each task, we collected the time spent for each task and the completion rate. After finishing a task, the participants filled a questionnaire, including a NASA-TLX [32], questions for evaluating a set of relevant criteria from those proposed for the evaluation of UI toolkits by Olsen [33], and an open-ended question for collecting their opinions and suggestions on the evaluated support.

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 [33] 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?

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

- *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 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 this test.

6.4. Results

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 being 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 12 (red boxes for the *Line* version and green boxes for the *OctoPocus* version).

Line Feedback task. We found a 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).

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

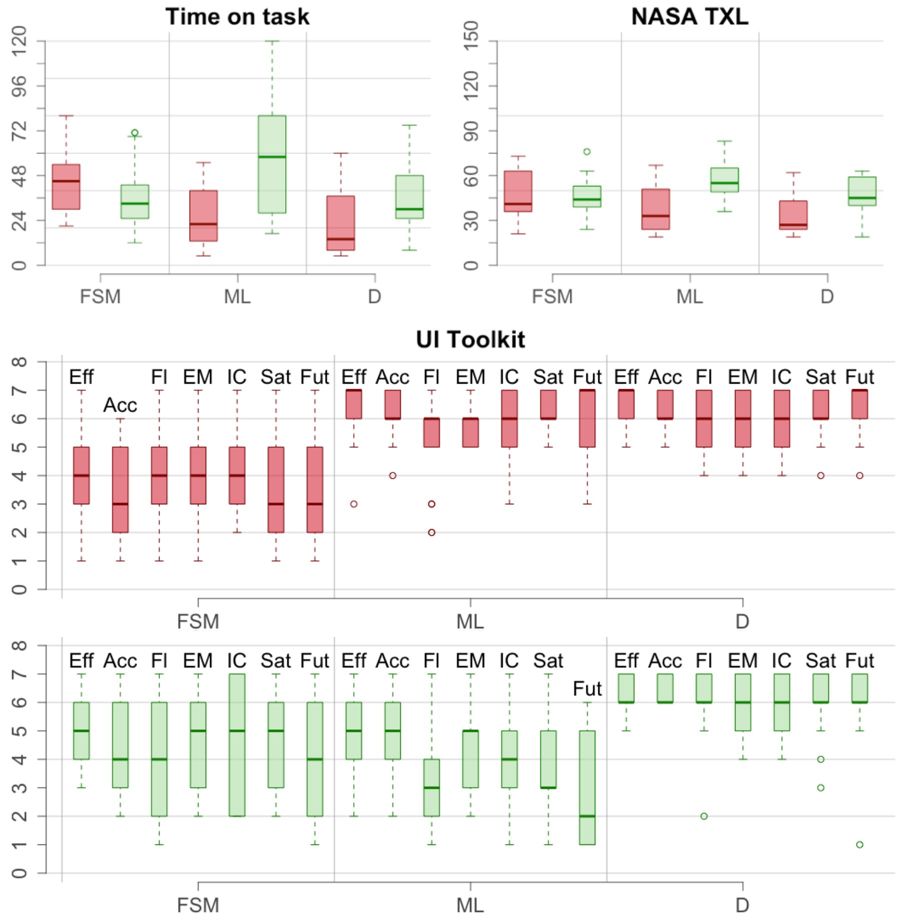


Figure 12: Post-task evaluation results. Red boxes correspond to the *line* feedback UI variant, while green boxes correspond to the *OctoPocus* variant. In the first row it shows the time on task and the task load (NASA TLX [32]), in the second row the selected UI toolkit criteria for the *line* feedback, in the third the same criteria for the *OctoPocus* task. The order for the criteria is the same as in section 6.3.

- Task load ($F(2,16) = 3.998$, $p < .03$, $\eta^2 = 0.20$, NASA TLX [32] in a 0 to 150 scale, lower is better). We registered a 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 a significant 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 practical significant difference between ML and FSM ($p < 10^{-3}$, $c.i. = [0.99, 3.83]$), and a significant one between D and FSM ($p < 10^{-4}$, $c.i. = [0.99, 3.83]$).

- 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 practical significant difference between ML and FSM ($p < 10^{-3}$, $c.i. = [0.85, 4.08]$) and a significant one 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.

OctoPocus feedback task. We found a 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 [32] in a 0 to 150 scale, lower is better). We registered a practical 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 a significant difference between D and ML ($p < .005$, $c.i. = [0.50, 2.55]$) and a practical 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 a practical 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 a practical significant one between D and FSM ($p < .09$, $c.i. = [0.13, 3.16]$).

The results confirm the hypothesis for this task as well: the 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.

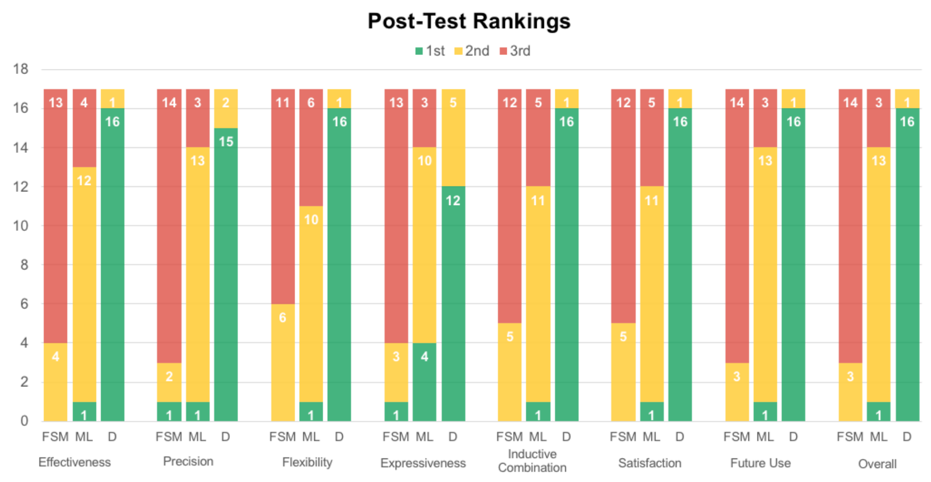


Figure 13: Post test evaluation results.

Post-test results. The post-test results in Figure 13 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, the Machine Learning was considered as the best approach by 4 participants, that explained their choice 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 the Machine Learning. Participants explained this 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 of them.

7. 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 in Python, relying on an existing HMM library called Pomegranate [10] for evaluating the proposed compositional approach. We published the DEICTIC software package for creating and composing gesture recognisers on GitHub [29].

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

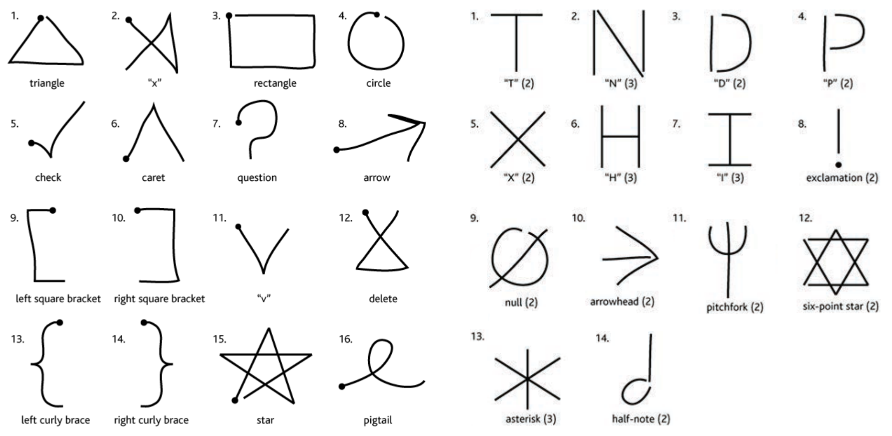


Figure 14: Gesture sets in the two evaluation datasets. The left part shows the single stroke gestures (figure taken from [11]), while the right part shows the multiple stroke gestures (figure taken from [12])

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 4.7.

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

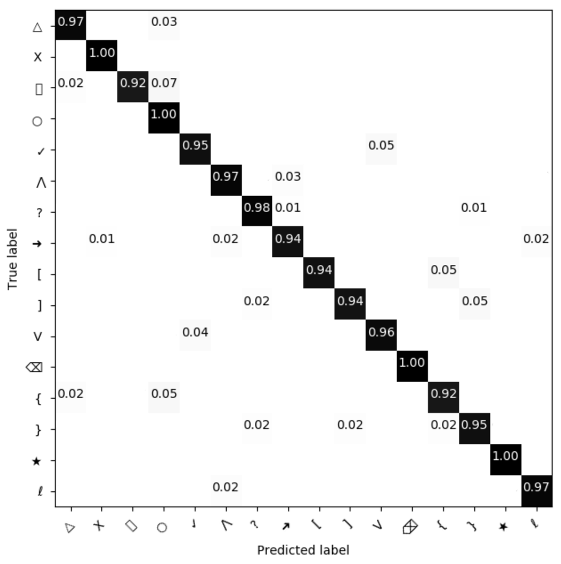
1. **Recognition of single-stroke gestures**. We considered the 1$-dataset introduced in [11]. It contains 330 repetitions of 16 single stroke gestures, shown in Figure 14, left part. We repeated the classification task using both DEICTIC and HMMs trained with samples from the dataset.

2. **Recognition of multiple-stroke gestures**. We considered the N$-dataset introduced in [12]. It contains 600 repetitions of 14 multi-stroke gestures, shown in Figure 14, 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 Appendix A and B. 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.

1$ Dataset **N$ Dataset**



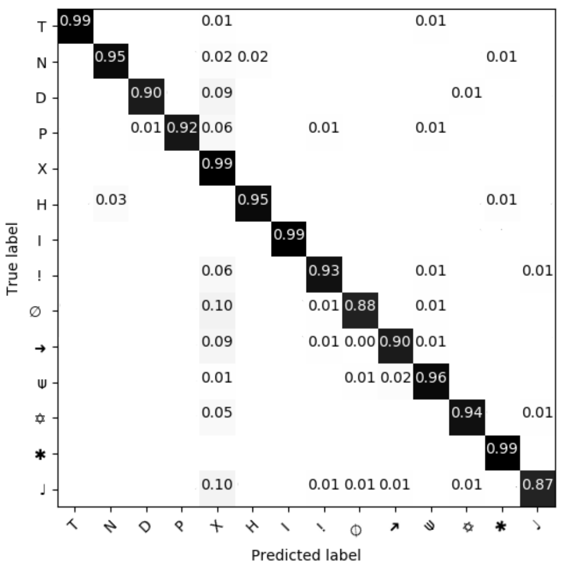


Table 1: Confusion matrix for the 1$-dollar [11] (left) and the the N$-dollar [12] (right) gesture dataset. Rows represent the ground-truth class, while the columns represent the class assigned by DEICTIC. Ground terms contained 6 states, working with gestures resampled to 20 samples per normalised unit. 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).

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 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 [34] for user-independent recognition, shown in Figure 15.

Considering the single-stroke gestures in the 1$-dataset, the 1$ algorithm [11] has the best performance (97.1%), followed by the P$ [34] (96.6%). DEICTIC (96.2%) is positioned immediately after, outperforming Protractor [35] (95.5%) and N$ [12] (95.2%). Table 1 (left 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.

Considering the multiple-stroke gestures in the N$-dataset, the best performance is reached by the P$ algorithm [34] (98.0%), followed by the N$ [12] (96.4%). DEICTIC (94.0%) outperforms Dynamic Time Warping [11, 34] (93.4%) and approaches based on angular cosine [35] (91.3%) and Euclidean [36] (91.5%) distances. Table 1, right 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.

We added to the approaches from the literature in Figure 15 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 about by 3%.

We did not register the same performance on the N$-dataset: the mean recognition rate was about 87%. They had particular difficulties in distinguishing D strokes from P and that many gestures are confused with the half-note and vice-versa. In contrast, DEICTIC had a lower recognition rate with respect to the single-stroke dataset (about 94%), but such decrease was not significant ($t(22) = 1.76$, $p = .09$). Our approach performed significantly better than ad-

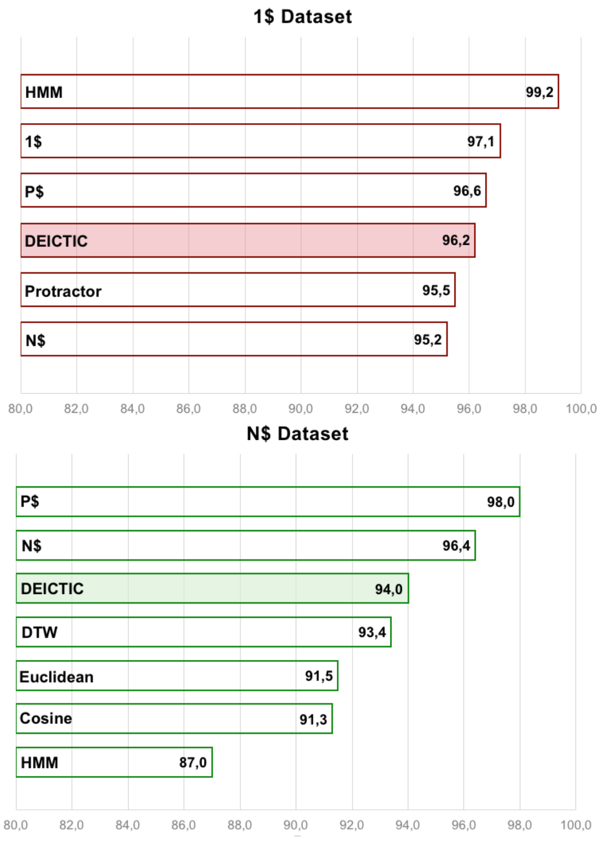


Figure 15: Accuracy of state-of-the-art approaches on the 1$-dataset [11] (top), and on the N$-dataset [12] (bottom).

hoc HMMs ($t(13) = 2.25$, $p = .04$), increasing their recognition level about by 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., 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 developer 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 [27] 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.

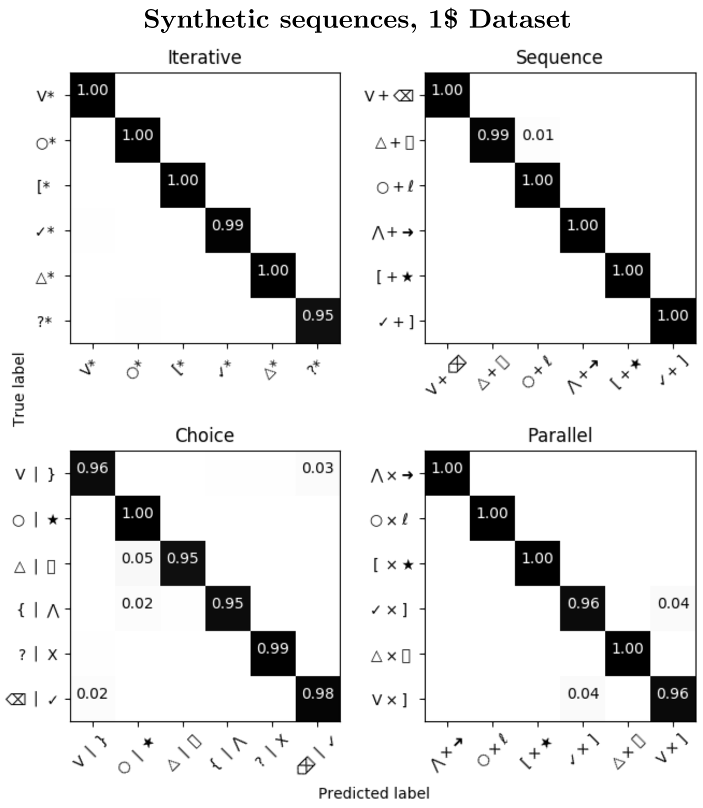


Table 2: Recognition of synthetic sequences

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 2 shows the recognition performance on single-stroke composite gestures, while Table 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 5, 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.

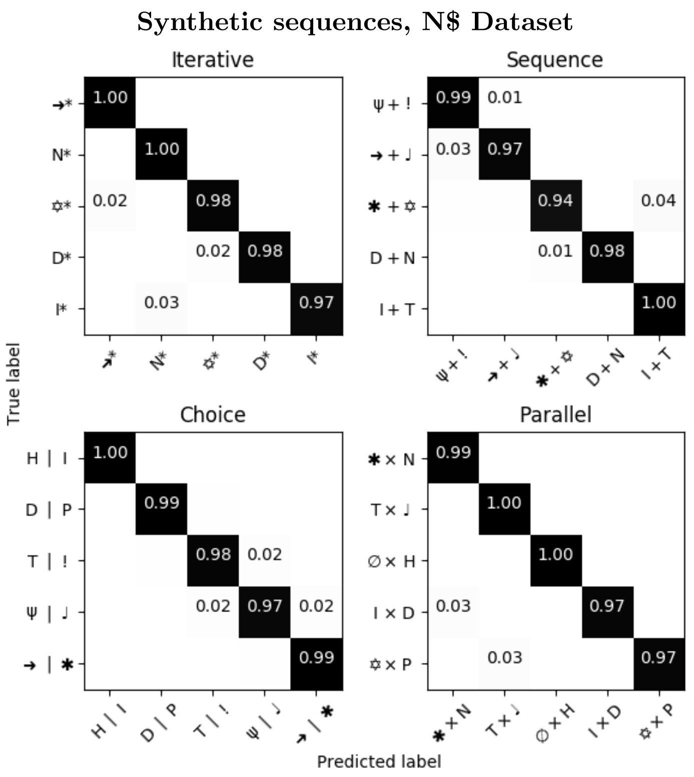


Table 3: Recognition of synthetic sequences

8. Conclusion and Future Work

In this paper, we filled the gap between the high accuracy offered by classification approaches and the inspection capabilities needed for providing feedback and feed-forward in user interfaces introducing 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 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. It requires only 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 such 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. We discussed two different experiments where we show that DEICTIC does not introduce sensible degradation of the recognition accuracy.

The approach has also set limitations. Even if our experiment showed an overall good performance, more precise recognition algorithms exist. However, the accuracy we obtained in our experiments is comparable with such approaches, which is an important improvement if compared with heuristic approaches. For that price, DEICTIC offers the same support for gesture sub-parts identification.

The second limitation is related to the number of states in the final HMM. 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 future work, we would like to extend the definition to 3D gestures and validate the composition approach in such setting. In addition, we will experiment different features for avoiding the 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.

References

[1] S. Mitra, T. Acharya, Gesture recognition: A survey, IEEE Trans. Systems, Man, and Cybernetics, Part C 37 (3) (2007) 311–324. doi:10.1109/TSMCC.2007.893280.

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

[3] H. Cheng, L. Yang, Z. Liu, Survey on 3D Hand Gesture Recognition, IEEE Trans. Circuits Syst. Video Techn. 26 (9) (2016) 1659–1673. doi:10.1109/TCSVT.2015.2469551.

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

[5] W. Delamare, C. Coutrix, L. Nigay, Designing guiding systems for gesture-based interaction, in: Proceedings of EICS 2015, ACM, 2015, pp. 44–53.

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

[7] L. D. Spano, A. Cisternino, F. Paternò, A Compositional Model for Gesture Definition, in: Proceedings of HCSE 2012, Springer, 2012, pp. 34–52.

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

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

[10] J. Schreiber, pomegranate, `https://github.com/jmschrei/pomegranate` (2016).

[11] J. O. Wobbrock, A. D. Wilson, Y. Li, Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes, in: Proceedings of UIST 2007, UIST '07, ACM, New York, NY, USA, 2007, pp. 159–168. `doi:10.1145/1294211.1294238`.

[12] L. Anthony, J. O. Wobbrock, $N-protractor: A Fast and Accurate Multistroke Recognizer, in: Proceedings of Graphics Interface 2012, 2012, pp. 117–120.

[13] A. Carcangiu, G. Fumera, F. Roli, L. D. Spano, Gesture modelling and recognition by integrating declarative models and pattern recognition algorithms, in: Proceedings of ICIAP 2017 (submitted paper), IAPR, Springer, 2017, pp. 1–11.

[14] Y. Yang, I. Saleemi, M. Shah, 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.

[15] Q. Chen, N. D. Georganas, E. M. Petriu, Real-time vision-based hand gesture recognition using haar-like features, in: Proceedings of IMTC 2007, IEEE, 2007, pp. 1–6.

[16] P. Natarajan, R. Nevatia, Online, real-time tracking and recognition of human actions, in: Proceedings of WMVC 2008, IEEE, 2008, pp. 1–8.

[17] P. Natarajan, V. K. Singh, R. Nevatia, Learning 3d action models from a few 2d videos for view invariant action recognition, in: Proceedings of CVPR 2010, IEEE, 2010, pp. 20006–2013.

[18] M. B. Holte, T. B. Moeslund, View invariant gesture recognition using 3d motion primitives, in: Proceedings of ICASSP 2008, IEEE, 2008, pp. 797–800.

[19] I. Kim, S. Chien, Analysis of 3D Hand Trajectory Gestures Using Stroke-Based Composite Hidden Markov Models, Appl. Intell. 15 (2) (2001) 131–143. doi:10.1023/A:1011231305559.

[20] D. Kammer, J. Wojdziak, M. Keck, R. Groh, S. Taranko, Towards a formalization of multi-touch gestures, in: Proceedings of ITS 2010, ACM, New York, NY, USA, 2010, pp. 49–58. doi:10.1145/1936652.1936662.

[21] C. Scholliers, L. Hoste, B. Signer, W. De Meuter, Midas: a declarative multi-touch interaction framework, in: Proceedings of TEI 2011, ACM, New York, NY, USA, 2011, pp. 49–56. doi:10.1145/1935701.1935712.

[22] L. Hoste, B. Dumas, B. Signer, Mudra: a unified multimodal interaction framework, in: Proceedings of ICMI 2011, ACM, New York, NY, USA, 2011, pp. 97–104. doi:10.1145/2070481.2070500.

[23] S. H. Khandkar, F. Maurer, A domain specific language to define gestures for multi-touch applications, in: Proceedings of the DSM 2010, ACM, New York, NY, USA, 2010, pp. 2:1—-2:6. doi:10.1145/2060329.2060339.

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

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

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

[27] L. E. Baum, T. Petrie, G. Soules, N. Weiss, A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains, The Annals of Mathematical Statistics 41 (1) (1970) 164–171. doi:10.1214/aoms/1177697196.

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

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

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

[31] O. Bau, W. E. Mackay, Octopocus: A dynamic guide for learning gesture-based command sets, in: Proceedings of UIST 2008, UIST '08, ACM, New York, NY, USA, 2008, pp. 37–46. doi:10.1145/1449715.1449724.

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

[33] D. R. Olsen, Jr., Evaluating user interface systems research, in: Proceedings of UIST 2007, UIST '07, ACM, New York, NY, USA, 2007, pp. 251–258. doi:10.1145/1294211.1294256.

[34] R.-D. Vatavu, L. Anthony, J. O. Wobbrock, Gestures as point clouds: a $P recognizer for user interface prototypes, in: Proceedings of ICMI-2012, ACM, 2012, pp. 273–280.

[35] Y. Li, Protractor: a fast and accurate gesture recognizer, in: Proceedings of CHI 2012, ACM, 2010, pp. 2169–2172.

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

Appendix A: 1$-dataset Gesture Modelling

	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	$P(0,0) + A_{\circlearrowright}(-3,-3) + A_{\circlearrowright}(3,-3) + A_{\circlearrowright}(3,3) + A_{\circlearrowright}(-3,3)$
✓	Check	$P(0,0) + L(2,-2) + L(4,6)$
⋀	Caret	$P(0,0) + L(2,3) + L(2,-3)$
?	Question m.	$P(0,0) + A_{\circlearrowleft}(4,4) + A_{\circlearrowleft}(4,-4) + A_{\circlearrowleft}(-4,-4) + A_{\circlearrowright}(-2,-2) + A_{\circlearrowright}(2,-2)$
→	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	$P(0,0) + A_{\circlearrowleft}(-5,-5) + A_{\circlearrowright}(-3,-3) + A_{\circlearrowright}(3,-3) + A_{\circlearrowleft}(5,-5)$
}	R. brace	$P(0,0) + A_{\circlearrowright}(5,-5) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowright}(-5,-5)$
⋆	Star	$P(0,0) + L(2,5) + L(2,-5) + L(-5,3) + L(6,0) + L(-5,-3)$
ℓ	Pigtail	$P(0,0) + A_{\circlearrowleft}(3,3) + A_{\circlearrowleft}(-1,1) + A_{\circlearrowleft}(-1,-1) + A_{\circlearrowleft}(3,-3)$

Appendix B: N$-dataset modelling

	Gesture	Expression
T	T	$P(-2,0) + L(4,0) + P(0,0) + L(0,-4) \quad \mid$
		$P(0,0) + L(0,-4) + P(-2,0) + L(4,0)$
N	N	$P(0,4) + L(0,-4) + P(0,4) + L(4,-4) + P(4,4) + L(0,-4)$
D	D	$P(0,6) + L(0,-6) + P(0,6) + L(2,0) + A_{\circlearrowright}(3,-3) + A_{\circlearrowright}(-3,-3) + L(-2,0)$
P	P	$P(0,8) + L(0,-8) + P(0,8) + L(2,0) + A_{\circlearrowright}(2,-2) + A_{\circlearrowright}(-2,-2) + L(-2,0)$
X	X	$P(0,0) + L(4,4) + P(0,4) + L(4,-4) \quad \mid$
		$P(4,4) + L(-4,-4) + P(0,4) + L(4,-4) \quad \mid$
		$P(0,4) + L(4,-4) + P(0,0) + L(4,4 \quad \mid)$
		$P(0,4) + L(4,-4) + L(4,4) + L(-4,-4)$
H	H	$P(0,4) + L(0,-4) + P(0,2) + L(4,0) + P(4,4) + L(0,-4) \quad \mid$
		$P(0,4) + L(0,-4) + P(4,4) + L(0,-4) + P(0,2) + L(4,0)$
I	I	$P(0,4) + L(4,0) + P(2,4) + L(0,-4) + P(0,0) + L(4,0) \quad \mid$
		$P(2,4) + L(0,-4) + P(0,0) + L(4,0) + P(0,4) + L(4,0) \quad \mid$
		$P(0,4) + L(4,0) + P(0,0) + L(4,0) + P(2,0) + L(0,4) \quad \mid$
		$P(2,4) + L(0,-4) + P(0,4) + L(4,0) + P(0,0) + L(4,0)$
!	Exc. point	$P(0,4) + L(0,-3) + P(0,1) + L(0,-1)$
$\varnothing$	Null	$P(0,0) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3) + A_{\circlearrowleft}(-3,3) + P(4,1) + L(-8,-8) \quad \mid$
		$P(0,0) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3) + A_{\circlearrowleft}(-3,3) + P(-4,-7) + L(8,8)$
$\rightarrow$	Arrow	$P(0,0) + L(6,0) + P(4,2) + L(2,-2) + L(-2,-2) \quad \mid$
		$P(4,2) + L(2,-2) + L(-2,-2) + P(0,0) + L(6,0)$
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	$P(0,0)+L(-2,-4)+L(4,0)+L(-2,4)+P(-2,-1)+L(4,0)+L(-2,-4)+L(-2,4) \quad \mid$
	star	$P(-2,-1)+L(4,0)+L(-2,-4)+L(-2,4)+P(0,0)+L(-2,-4)+L(4,0)+L(-2,4) \quad \mid$
		$P(-2,-2)+L(2,4)+L(2,-4)+L(-4,0)+P(-2,1)+L(4,0)+L(-2,-4)+L(-2,4) \quad \mid$
		$P(-2,1)+L(4,0)+L(-2,-4)+L(-2,4)+P(-2,-2)+L(2,4)+L(2,-4)+L(-4,0) \quad \mid$
		$P(-2,-2)+L(2,4)+L(2,-4)+L(-4,0)+P(-2,1)+L(2,-4)+L(2,4)+L(-4,0) \quad \mid$
		$P(-2,1)+L(2,-4)+L(2,4)+L(-4,0)+L(-2,-2)+L(2,4)+L(2,-4)+L(-4,0) \quad \mid$
		$P(0,0)+L(-2,-4)+L(4,0)+L(-2,4)+P(-2,-1)+L(2,-4)+L(2,4)+L(-4,0) \quad \mid$
		$P(-2,-1) + L(2,-4) + L(2,4) + L(-4,0) + P(0,0) + L(-2,-4) + L(4,0) + L(-2,4)$
$\ast$	Asterisk	$P(4,4) + L(-4,-4)) + P(0,4) + L(4,-4) + P(2,4) + L(0,-4)$
𝅗𝅥	Half note	$P(0,0) + A_{\circlearrowleft}(-3,3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3) + P(2,16) + L(0,-20)$
		$P(2,16) + L(0,-20) + P(0,0) + A_{\circlearrowleft}(-3,3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3)$
		$P(0,0) + A_{\circlearrowleft}(-3,3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3) + P(2,-4) + L(0,20)$
		$P(2,-4) + L(0,20) + P(0,0) + A_{\circlearrowleft}(-3,3) + A_{\circlearrowleft}(-3,-3) + A_{\circlearrowleft}(3,-3) + A_{\circlearrowleft}(3,3)$