Journal of Computational Science 10 (2015) 66–76

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

A new approach for touch gesture recognition: Conversive Hidden non-Markovian Models

Tim Dittmar*, Claudia Krull, Graham Horton

Otto-von-Guericke-University Magdeburg, P.O. Box 4120, 39016 Magdeburg, Germany

ARTICLE INFO

Article history:
Received 15 March 2014
Received in revised form 3 November 2014
Accepted 5 March 2015
Available online 16 March 2015

Keywords:
Touch gesture recognition
Hidden Markov models
Conversive Hidden non-Markovian models
Proxel

ABSTRACT

With the current boom of touch devices the recognition of touch gestures is becoming an important field of research. Performing such gestures can be seen as a stochastic process, as there can be many little differences between different executions. Therefore stochastic models like Hidden Markov Models have already been applied to gesture recognition. Although the modelling possibilities of Hidden Markov Models are limited, they achieve an acceptable recognition quality. But they have never been tested with gestures that only differ in execution speed.

We propose the use of Conversive Hidden non-Markovian Models for touch gesture recognition. This extension of Hidden Markov Models enhances the modelling possibilities and adds timing features. In this paper, two touch gesture recognition systems were developed and implemented based on these two model types. Experiments with a set of similar gestures show that the proposed model class is a good and competitive alternative to Hidden Markov Models.

1. Introduction

1.1. Background

Due to the big success of smartphones and tablets, a ubiquitous presence of multi-touch devices is establishing itself around the world. While the multi-touch input method offers manifold possibilities for controlling devices, almost all of them are usually controlled by using a fixed set of simple gestures like tap, drag and pinch. Such systems can be realized quite easily using heuristics, but they are not very flexible.

To create a gesture recognition system using heuristics requires that the system has all of the gestures implemented *a priori*. The subsequent addition of a new gesture could make code adaptations of the previously implemented gestures necessary, especially when the gestures are very similar. In order to create a more flexible gesture recognition system, other methods need to be used that enable the definition of a gesture by performing several examples. This way the user of a touch device could define personalized gestures for each action.

A flexible multi-touch gesture recognition system was presented by Damaraju and Kerne [1]. This system is based on Hidden Markov Models (HMM) and creates one HMM for each gesture using sample inputs. While there are other pattern recognition methods that are deployed for touch gesture recognition, the current work focuses on hidden models, namely HMM and a quite new model class: Conversive Hidden non-Markovian models (CHnMM).

This new model class is an extension of HMMs and therefore should also be able to be the basis of a flexible gesture recognition system with an open gesture set. But in contrast to HMMs, CHnMMs are able to incorporate explicit timing information. As a result, it should be possible to create more precise models of gestures, so that even very similar gestures can be distinguished. However, their computation is more complex and could render real-time applications infeasible. With this work, we want to evaluate whether CHnMMs are applicable for touch gesture recognition and how they perform in comparison to HMMs.

1.2. Motivation and goals

The idea to use non-Markovian Models for gesture recognition was first brought into consideration by Bosse et al. [2]. The goal was to show that a system based on Hidden non-Markovian models (HnMM) could distinguish gestures that are similar in shape

* Corresponding author. Tel.: +49 15114441120.
E-mail addresses: tim.dittmar@ovgu.de (T. Dittmar), claudia.krull@ovgu.de (C. Krull), graham.horton@ovgu.de (G. Horton).

http://dx.doi.org/10.1016/j.jocs.2015.03.002
1877-7503/

but differ in execution speed. This feature had not been considered for HMM gesture recognition systems before. For that reason an HMM- and an HnMM-system were developed to recognize gestures performed with a Nintendo Wiimote and both systems were compared in their recognition quality.

Inspired by this, we attempt to apply a similar approach to touch devices. The general goal of this work is to find out whether CHnMMs are applicable for touch gesture recognition. Therefore the CHnMM-system needs to reach similar or better recognition rates than the HMM-system. In addition, the time needed for recognizing the gesture has to be competitive, so that the system could be used in real-time scenarios.

In order to evaluate the recognition quality, the metrics Precision and Recall are used. These metrics give a measure of the quality of the recognition results and a measure for the completeness. These metrics are appropriate, because it is possible that a performed gesture is not classified by the recognition system. A high level of Precision will be needed to support the claim that a CHnMM-system can distinguish between similar gestures. We therefore aim for a Precision that is at least 85% and that exceeds the HMM-system Precision value. Furthermore, a minimum Recall value of 70% should be reached.

Since CHnMMs have never been used in practical applications before, our goal is to merely to establish the plausibility of our approach, not to develop a ready-to-use touch recognition system.

2. Background information

2.1. Previous work

In this section, we summarise the methods on which our algorithm is based and introduce the terms and models that will be used.

In the context of this work, the term hidden models stands for all model classes that are able to infer conclusions from so-called partially observable discrete stochastic (PODS) systems [3, p. 1]. This is a special class of real-world problems where a stochastic description of the system behaviour is known, but the discrete states of the system cannot be directly observed. Instead, observable signals are created by the system that give clues about its inner state.

2.1.1. Hidden Markov models

One of the first types of hidden model that was formalized and applied to recognition problems is the Hidden Markovian model (HMM). A very good overview of its elements and how it can be applied to speech recognition is given by Rabiner in [4]. Today, HMMs are used in many fields of pattern recognition such as speech recognition, handwriting and gesture recognition and DNA analysis [5, pp. 9–29]. An HMM can be described mathematically and the notation is quite consistent across the literature (e.g. [3–6]), with the exception of some subtleties. The following paragraphs define the notations that are used in this article.

The two basic elements of an HMM are:

- a finite set of N possible states $S = \{s_1, s_2, \ldots, s_N\}, N \in \mathbb{N}$
- a finite set of discrete outputs (also called symbols) $V = \{v_1, v_2, \ldots, v_M\}$

For this work, a *trace* is defined as a simple sequence of T observations $O = o_1, o_2 \ldots o_T$ with $o_t \in V$. Furthermore, $q_t \in S$ denotes the state of the system after the tth symbol emission and a *path* $Q = q_1, q_2 \ldots q_T$ is a possible sequence of traversed internal states of an HMM. With these definitions the additional elements of an HMM can be defined:

- a quadratic matrix of state-transition probabilities

$$A = \{a_{ij} | P(q_{t+1} = s_j | q_t = s_i)\}, A \in \mathbb{R}^{N \times N}$$

- a *initial probability vector* Π of start probabilities

$$\Pi = (\pi_0, \pi_1, \ldots, \pi_N), \pi_i = P(q_0 = s_i), \Pi \in \mathbb{R}^N$$

- a N-by-M matrix of state specific output probability distributions

$$B = \{b_i(v_k) | b_i(v_k) = P(o_t = v_k | q_t = s_i)\}$$

As a result, an HMM – usually denoted by λ – can be represented as a tuple $\lambda = (S, V, A, B, \Pi)$, or $\lambda = (A, B, \Pi)$ for short, since the sets S and V are merely names for states and symbols that do not affect the model behaviour.

The sets S and A define a Markov chain that fulfils the Markov property, i.e. the behaviour of a system or process at a given time t depends only on the previous state. This can be expressed as follows:

$$P(q_t | q_1, q_2, \ldots, q_{t-1}) = P(q_t | q_{t-1}), q_t \in S$$

The internal states are assumed to be unobservable, thus the Markov chain represents the *Hidden* part of a *Hidden* Markov model. The only part that is observable are the outputs or symbols of a system or process o_t. The HMM gives a stochastic description of how a PODS system or process creates these symbols by assuming that in every state a certain probability for creating a certain symbol is present, which is expressed in matrix B. Since an HMM is processed in a step-wise manner, it is also assumed that the system or process creates a symbol in each step. In most cases, the step size is a discrete interval δt that describes the time between two outputs/symbols emitted by the process or system. When such a system is observed, these signals are collected in a trace O that could also be interpreted as a log of the system.

There are three basic problems that are of interest when using HMMs: Evaluation, Decoding and Training [3–5], which all involve an observed trace O of a PODS system or process and an HMM λ that represents that system or process.

The evaluation problem is the determination of the probability that a given trace O is the result of a system or process described by the HMM λ or formally $P(O|\lambda)$. It "is the most widely used measure for assessing the quality with which an HMM describes the statistical properties of certain data" [5, p. 78].

The calculation of this measure could be naively done with a "brute force" approach where every possible *path* is determined and the probability of emitting the *trace* O along each *path* is calculated and summarized. Since this approach has an exponential complexity of $O(TN^T)$, it is impractical for most use cases. Instead, a more efficient method is used: the *forward algorithm*. This is based on the *forward variable* $\alpha_t(s_i)$ which represents the probability that a given model λ generated the given trace up to o_t and that state i has been reached:

$$\alpha_t(s_i) = P(o_1 o_2 \ldots o_t \cap q_t = s_i | \lambda)$$

These *forward variables* are trivial to calculate for the first step by multiplying the initial probability for being in the state i with the probability to generate the first symbol of the trace:

$$\alpha_1(i) = \pi_i b_i(o_1).$$

For further time steps the formula

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right) b_j(o_{t+1})$$

is used to finally get the forward variables $\alpha_T(i)$ for each state i with which the evaluation value is determined as follows:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

The second problem or task, the decoding, determines $^*\arg\max_Q P(Q|O, \lambda)$, the *path* that is the most probable for a given trace O and model λ. It can be efficiently solved using the Viterbi algorithm [7], but does not play a role in this article.

The training task however is needed to create a model that fits or best describes a given *trace* of a system or process that is to be modelled. For this task an initial model λ is used to find a new model λ' that fulfils the condition $P(O|\lambda') \geq P(O|\lambda)$. In other words, training of an HMM uses traces of a system, i.e. example data, to determine model parameters that better describe the system. An efficient algorithm that solves the training task for HMMs is the Baum–Welch algorithm that iteratively improves a model according to a given trace or a set of traces. The iterations are terminated if the probability $P(O|\lambda)$ stagnates, which means a local optimum has been found. The quality of the local optimum, i.e. the trained model, depends strongly on the chosen initial model.

The existence of efficient algorithms to solve these basic problems has made this model class widely used, although they also have some disadvantages. HMMs do not incorporate explicit timing or duration information, for example. That means that the transition from one state to another cannot be expressed as a duration like "2–3 s after state one has been reached", because HMMs "implicitly model state durations by a geometric distribution, which is usually inappropriate [8]". Additionally, the information when a symbol has been created is not considered by an HMM, although it seems to be an important clue from the observed system. This latter limitation can be worked around by using an implicit timing which is caused by a fixed discrete time step δt. In this case, systems that do not emit a symbol periodically cannot be modelled correctly. A further limitation of HMMs is that only Markovian systems can be modelled exactly, although these systems rarely occur in practice. In almost all cases therefore, an HMM is an oversimplification of the real system.

2.1.2. Conversive Hidden non-Markovian Models

Conversive Hidden non-Markovian Models have been thoroughly formalised and analysed by Buchholz in his dissertation [3] from which most definitions in this section are taken. However, for consistency, some variables have been adapted. This model class originates from the research of Hidden non-Markovian Models (HnMM) [9], since they are a subclass of HnMMs. HnMMs are very expressive in their modelling capabilities but this advantage comes with computational complexity. CHnMMs only slightly limit these modelling capabilities but reduce the computational complexity by an amount that makes efficient solution algorithms feasible. Their formal definition is presented in the following paragraphs.

Certain elements and definitions of a CHnMM are similar to those in an HMM, namely

- the state set S of length N,
- the symbol set V of length M,
- the initial probability vector $\Pi \in \mathbb{R}^N$
- and the $N \times N$ matrix A containing the state change behaviour, however, the elements a_{ij} are more complex in a CHnMM and they will be exactly defined in the next section.

In addition to these shared elements, a CHnMM also contains the following variables:

- a set $TR = \{tr_1, tr_2, \ldots, tr_K\}$ of K transitions (also called activities) that define the model behaviour. Each transition tr_i is a tuple consisting of the following three elements:
  - *dist* represents the continuous probability distribution that specifies the duration of the transition which causes a discrete state change on completion
  - $b(v)$ is a function that returns the output probability of symbol v when the transition causes a state change. It is the semantic equivalent to the output probabilities in B for HMMs, but they are associated with transitions for CHnMMs instead of states as in HMMs.
  - *aging* is a boolean value that determines if the time that the transition has been active is saved (*aging* = *true*) or reset to 0 (*aging* = *false*) if there is a state change caused by another transition, i.e. if the current activity is interrupted by another one.

  All elements a_{ij} in A are elements of TR, thus the following holds $\forall i \forall j : a_{ij} \in TR, 1 \leq i \leq N, 1 \leq j \leq N$

A CHnMM model λ is fully defined as a tuple $\lambda = (S, V, A, TR, \Pi)$ that contains all previously described elements. As was the case for the HMM, an abbreviated form is possible that implicitly contains all needed information: $\lambda = (A, \Pi)$.

In the context of CHnMMs the definition of an observation o needs to be extended to also contain the time of the symbol emission. Therefore o becomes a tuple (*symbol*, *time*) with *symbol* $\in V$ being the observed symbol and *time* the point in time when the symbol was observed. As a result, a *trace* O becomes a sequence of these extended observations in the context of CHnMMs.

In his dissertation Buchholz [3, pp. 89–115] developed efficient algorithms for solving the evaluation and decoding task for CHnMMs based on the algorithms used for HMMs. The algorithm for the training of CHnMMs had the disadvantage of not being guaranteed to improve the model with each iteration. All of these algorithms make use of the proxel method, which is explained in the next section.

2.1.3. The Proxel method

The Proxel method is a technique for simulating concurrent non-Markovian behaviour described by stochastic models with discrete states. It was described first by Horton [10] and was further formalised and thoroughly analysed in the dissertation by Lazarova-Molnar [11]. This method is utilised in the algorithm for solving the evaluation task for CHnMMs that were developed by Buchholz [3] and that will be adapted for this work. Therefore this section gives a short overview and a mathematical description of Proxels.

A concise description of the Proxel method is given by Buchholz [3, p. 22]:

> The Proxel method is a state-space based simulation algorithm that uses supplementary variables to handle non-Markovian activities. Its basic approach is to observe the model at equidistant points in time and for those times to determine the probabilities of the model to be in each possible state. Those probabilities are computed inductively by using the probabilities of every possible state from the previous point in time and

computing the probabilities for every possible completion of a single activity during the time step in order to determine individual state probabilities for each state at the end of the time step.

The term Proxel stands for probability element, which is the computational unit of the algorithm. A Proxel represents a certain state of the system together with the probability for this state at a certain point of time. Formally, a Proxel is a tuple $\rho = (S, \vec{\tau}, \alpha)$, where S is the discrete state of the system and α reflects the probability of being in this state. In order to describe the complete system state, an additional vector $\vec{\tau}$ is needed that contains the ages of all active transitions that have *aging* set to true. The current point in time t is known from the Proxel set R_t that contains all the Proxels of that point in time, i.e. R_t contains all possible system states and their probabilities for a given time t.

2.2. Related work

In this section this work is embedded into the current state of the art of gesture recognition by presenting and discussing similar approaches.

The research field for gesture recognition is quite broad and in most cases it is about detecting defined movements of human arms and hands via a camera that records the movement. HMMs have become a widespread method in this research area besides other tools like particle filtering [12] and condensation [13] methods, soft computing [14] and other statistical approaches [15]. Also famous learning algorithms like artificial neural networks [16] and support vector machines [17] have been utilized for gesture recognition tasks, but apparently a gesture set with similar shaped gestures that are performed with different speeds has never been considered.

Probably the first work that used HMMs for gesture recognition is by Yamato et al. from 1992 [18] which already states that their HMM gesture recognition is time-scale invariant. This is a desired feature for their use-case, the classification of eight different tennis strokes where it does not matter how fast or slow the tennis stroke is performed. In order for HMMs to work, only the symbol sequence has to be similar but not the time of their emission. With our proposed CHnMM approach we want to introduce a time aware model variant, an aspect rarely investigated for gesture recognition.

Today, HMM-based gesture recognition for a sequence of camera frames still follows the same principle that was proposed by Yamato et al.: For each frame important or relevant features are extracted which are converted to a symbol v that represents an observation o. The resulting trace O is used to evaluate which HMM fits best to it. The HMMs are created before, using a number of example traces of a gesture that is to be represented by that HMM.

There are a lot of examples for HMM gesture recognition systems for hand motion gestures [6,19–21] but HMM-based recognition systems for touch gestures are quite rare. One example however is the already mentioned (see Section 1.1) work of Damaraju and Kerne [1] that describes an HMM multi-touch gesture recognition system that learns its gestures by getting examples of the gesture by several users. According to their experiment this system produces good recognition results. Unfortunately the article is very short and not very detailed. Therefore it is not exactly clear what symbol set was used and how their automatically produced HMMs would look like. Nevertheless it seems ideal as a touch gesture recognition system that defines new gestures by examples. But unfortunately the distinction of gestures with different execution speeds was not considered and is likely to be problematic due to the time-scale invariance of HMMs.

Another example of an HMM-based touch gesture recognition system is from Yang and Xu [22] that employs the usual approach for HMM gesture recognition systems, which is also used in this work. It consists of the following five steps:

1. Define meaningful gestures.
2. Describe each gesture in terms of an HMM.
3. Collect training data.
4. Train the HMMs through training data.
5. Evaluate gestures with the trained model.

Step 1, the gesture set, is defined in Section 4 while the other steps are covered in Section 3 for the HMM and the CHnMM recognition system that is to be built in this article.

Since HnMMs and CHnMMs are quite new model classes, their practical applications are still rare and an open field of research. One application is presented by Denkena et al. [23] who employ HnMMs for behaviour reconstruction of material flows in small and medium sized manufacturing enterprises. Apart from that only the already mentioned work by Bosse et al. [2] provides a real world application for HnMMs, except for some theoretical and academic applications that already exist. In their work hand movement gestures that are performed with a Nintendo Wiimote were classified using an HnMM-based recognition system. The modelling capabilities of HnMMs are restricted in their work in a way that is similar to the CHnMM modelling class, but that could not be known, since the work of Buchholz[3] was not available at that time. For this work however, the research efforts of Buchholz are utilized and as a result specialized and efficient algorithms for CHnMMs can be employed.

The use of HMMs as generative models is also common in the field of online signature verification where a user is authenticated by writing his signature with a pen on a device that tracks the movements. This online signature verification task is very similar to the recognition of touch gestures, especially in regard to the data that is created by the pen tracking device and the touch device. There are even devices that support both input forms (pen/stylus and finger) like multi-touch tabletops. Due to this similarity, techniques and conclusions from this research area might be helpful. Argones Rúa and Alba Castro [24] for example investigated which features are important for good verification results, using HMMs as generative models.

2.3. Environment and set-up

This section provides information about the technical set-up of the experiments in Section 4 and upon which some implementation details are based.

The touch device was a multi-touch tabletop which used the frustrated total internal reflection (FTIR) principle [25]. A camera is mounted under the table that is able to monitor the infrared light that is reflected from touched points on the table surface. From the recorded frames of the camera, required touch information such as coordinates need to be extracted. This is provided by the Community Core Vision (CCV) open source software in version 1.4 [26] which is visualized in Fig. 1. The image shows an actual infrared camera frame of the tabletop surface top left and to the right of it the detected touches and their ids. A large number of settings are available to fine-tune the touch detection process.

The detected touch data is provided to a connected computer via the TUIO protocol [27]. A Java implementation that receives the TUIO messages is provided with the tabletop. It merges the raw data into a *Stroke* class structure that contains all touch points of a certain finger (more precisely, a certain touch id). Three event handlers are available that inform about new information, namely strokeStarted, strokeUpdated and strokeFinished. All event handlers provide the according stroke object that has been updated or

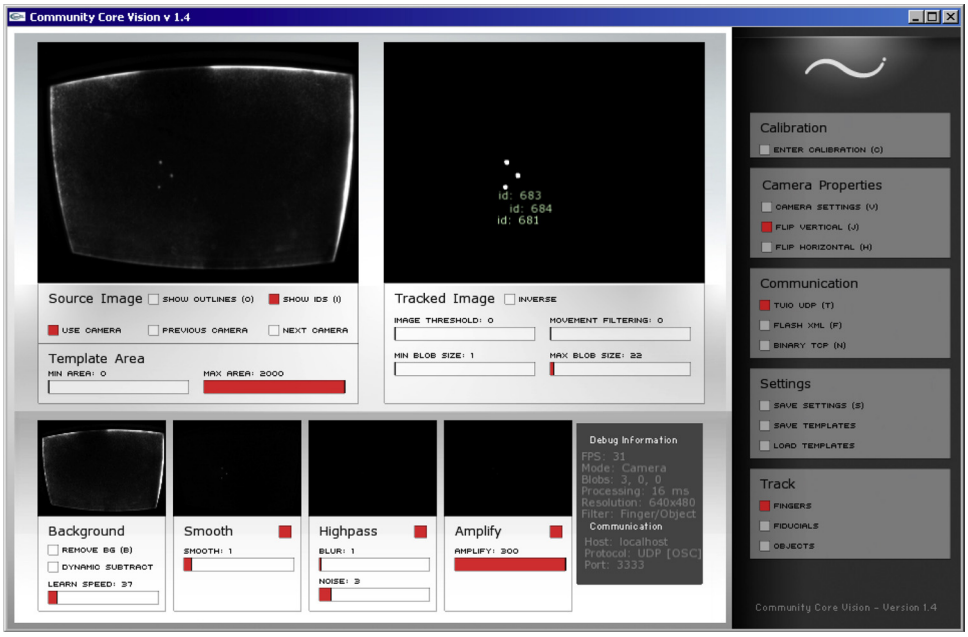

Fig. 1. Community Core Vision dashboard with various configuration settings.

created. This stroke data is used to calculate a symbol trace O; the details of this process are explained and discussed in Section 3.1.

3. Implementation

In this section important aspects are covered that explain how the CHnMM touch recognition system is implemented. The implementation of the HMM touch recognition system that is used as a reference to evaluate the results is covered in parallel. This helps to easily identify differences and similarities. For an overview of the technical set-up of the recognition systems, refer to Section 2.3.

3.1. Output Symbols

Symbols are very important for hidden models, because they are the only information a PODS system creates. As described in Sections 2.1.1 and 2.1.2, hidden models rely on an alphabet of possible output symbols V. In this work the system to be observed is a multi-touch tabletop that periodically sends information when a touch gesture is executed. This information contains x and y coordinates of the touched regions, a finger ID and a timestamp of when the touch was recognized. However, CHnMMs require a discrete symbol set V. Therefore, the continuous raw data from the touch device need to be quantised to discrete values before any computations with CHnMMs can be performed. The following paragraphs explain what symbols were selected for this work and how the raw data from the touch device is preprocessed to a *trace O* for each of both recognition systems.

The discrete symbol set V chosen consists of eight discrete directions: *Up*, *UpLeft*, *Left*, *DownLeft*, *Down*, *DownRight*, *Right*, *UpRight*. Each symbol represents the current direction of movement of a certain finger that touches the device, which is inspired by the symbol set used in the last experiment described by Bosse et al. [2]. In their previous experiments, discrete acceleration values were used, but these symbols contain information about the speed of execution to some degree. To stress the fact that CHnMMs are able to distinguish different timings in the symbol sequence without time information in the symbols the simple symbol set with directions only have been selected.

Although the HMM- and CHnMM-system use the same symbol set V, the way the symbols are calculated and emitted from the raw data is different in each case. This is due to the different ways the symbol emission is modelled. For HMMs, the symbol emission is connected to states ($b_i(v_k)$), whereas with CHnMMs it is connected to state transitions ($tr_i.b(v)$). This implies that a PODS system that is modelled with a CHnMM needs to emit symbols when there is a state change in the system. Thus, the symbol emission process for the CHnMM recognition system will receive the raw data from the touch device, calculate the direction of the finger movement and only emit a direction symbol to the trace O when there is a state change, i.e. a change in the movement direction. Ergo the state changes can happen at arbitrary points in time.

In contrast, an HMM works best with a constant flow of symbol emissions from the PODS system which is usually realized by emitting a symbol periodically with a fixed discrete time step δt. Therefore the symbol emission process that receives the raw data and creates the symbol trace O for the HMM recognition system cannot emit symbols at arbitrary points in time like the process for CHnMMs. In our case, the time step is given by the CCV software that analyses the frames of the camera and that merges the created TUIO [27] messages to stroke events. These events occur regularly every 30–50 ms, so that on each event a symbol (i.e. direction) can be calculated and emitted to the symbol trace O. With this approach, the HMM recognition system at least gets implicit timing information about the symbols, although the mathematical functionality of HMMs does not require any timestamps or durations, whereas CHnMMs need the time of symbol emission $o_t.time$.

The precise calculation of the direction symbols for the HMM system is carried out as follows: every time a strokeUpdated event is received, the vector from the last known position to the current position is calculated and the direction of this vector is classified as one of the eight directions defined in the symbol set. This direction is directly emitted as a symbol after calculation. It is done this way for every active finger separately, i.e. two moving fingers would lead to the emission of two direction symbols per processing step. Timestamps of the emissions are not needed or cannot be used for HMMs, although it seems to be an important source of information for gesture recognition with different execution speeds.

The calculation of symbols for the CHnMM-system is quite similar, but the symbols are only emitted if the current state or direction of the finger changes or if the finger touches the surface for the first time (where a previous moving state of the finger is unknown). Thus, a finger that moves in one direction should not emit any symbols until the direction changes. However, in practice, the data from a moving finger can be noisy, so that symbols are emitted although the intended direction of movement has not changed. To reduce this noise in the symbol stream, stroke updates are not considered for symbol emission if the calculated direction vector is shorter than an empirically defined threshold. For CHnMMs, the timestamp is essential and it is taken from the last known point of a stroke that determines when this point was received as opposed to the time when the symbol was calculated.

The two symbols GestureStart and GestureEnd were used for both systems to determine when the execution of a gesture starts and when it ends. We determined that a gesture starts with the first touch of a finger and ends when the last finger is removed from the surface. This definition was also used by Damaraju and Kerne [1] for their multi-touch gesture recognition system and is the reason why gestures that contain a phase where no finger is touching the surface, such as a double tap gesture, are excluded.

3.2. Modelling

Another important aspect for this work is the creation of HMMs and CHnMMs for a certain gesture. This section explains and discusses all relevant parts that are needed to understand how gesture models are built for our recognition systems.

There are numerous possibilities for the modelling for the recognition systems. One variation is the scope of what a model represents. It could be one of the following:

- A single model represents the whole set of possible gestures.
- A single model only represents a single gesture.
- Multiple models represent a single gesture.

The first variant has the advantage that only one calculation process is needed, because the most likely state or *path* of the model, according to the given symbol trace O of the executed gesture, is sufficient to determine/classify the executed gesture. But adding or removing a gesture from the set of detectable gestures can be very complicated with this approach, and also the training process can be much more difficult.

The third concept was used in the work of Bosse et al. [2], where a single gesture was modelled using three HnMMs, one for each axis of the acceleration sensors. This approach is more flexible for adding and removing gestures from the set of detectable gestures than the first one. But of course the number of calculation processes is notably higher, because a given trace has to be processed by multiple models.

Therefore, we selected the second approach for both recognition systems, so that adding and removing a gesture from the gesture set is just necessitates removing and adding the according gesture model and no further model or system adaptions such as re-training are necessary. The number of calculation operations is smaller than for the third approach but still higher than for the first one, especially with a large number of gestures in the gesture set.

As a next step, we need to understand the states S of the model and what they represent. Since symbols are used that give information about the direction of movement, it seems reasonable to separate a gesture into phases or states of directions. For example, a gesture consisting of a movement to the left and back to the right is modelled with a single state for each of the movement phases *left* and *right*. While the HMMs only consist of these movement states, the CHnMM has two additional states, namely the Start-state and

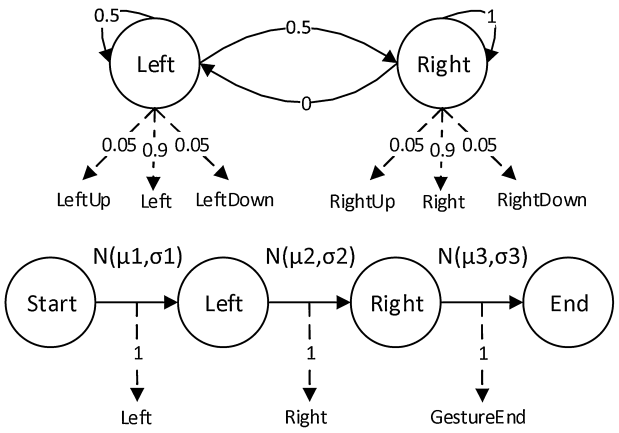

Fig. 2. Example models for a left–right gesture (HMM: top, CHnMM: bottom).

End-state (s_{start} and s_{end}). The Start-state is needed because CHnMMs are state change oriented, i.e. symbols are emitted when a state change occurs. If there were no state before the first movement phase, the first direction symbol could not be incorporated into the model, which would waste important evidence about the current state of the real system. The purpose of the End-state is mainly to allow the model to reject incomplete gestures. More details are given in Section 3.4.

Of course, all the aforementioned states need to be connected. These connections are described using the set A as explained in Sections 2.1.1 and 2.1.2. For HMMs, the connections are expressed by arcs with fixed probabilities that denote how likely it is to remain in a state or to change to another one in every discrete step, i.e. in this case with every periodic symbol emission. The CHnMMs on the other hand are connected with transitions and each of them has a certain probability distribution *dist* that determines when the state change is happening. For the gesture models in this work we assume that the time for a certain state change between two movement phases is normally distributed. Thus the expected value and the standard deviation need to be determined for the probability distribution of every transition in the CHnMM. Furthermore a left-to-right topology is used for both model types, i.e. a system cannot go back to a state that already has been active. Mathematically it means that $a_{ij} = 0, j < i$ for HMMs [6, p. 37] and $a_{ij} = \emptyset, j < i$ for CHnMMs.

Fig. 2 shows what an HMM and a CHnMM for a left–right gesture could look like. The dashed arrows mark possible symbol emissions and their probability. It can also be seen that the HMM only has fixed probabilities for state transitions whereas the CHnMM uses normal distributions. As a result, an HMM is dependent on the discrete time step it was created or trained with and therefore will probably not work reliably with other devices that provide data with different time step. Note that the HMM in Fig. 2 tolerates noise symbols in movement phases by giving them a low probability of occurring. For the CHnMM case, the filtering is not done within the gesture model but during the symbol calculation process as explained in Section 3.1.

This section explained how a certain gesture is modelled with either an HMM or CHnMM. But these gesture models are still missing their parameters. This last step in building a usable model is the training or manual parametrisation, which will be discussed in the next section.

3.3. Training and manual parametrisation

After a model for a gesture has been developed, the parameters for state changes and symbol emission probabilities need to

be determined, so that they match the finger movements of a user performing the gesture. This section describes how this is done for HMMs and CHnMMs.

The first step is to collect example data of the gesture. Therefore the gesture is performed twenty times with the HMM and twenty times with the CHnMM symbol calculation process to create a set of twenty example traces $\mathbb{O} = \{O_1, O_2, \ldots, O_{20}\}$ for each system. To train the HMM gesture models, the standard Baum-Welch algorithm is used. Starting with a base model that is built as explained in the previous section and filled with estimated parameters, the algorithm approaches a local optimum with every iteration. Because of this, it is important to have a good initial base model in order to avoid convergence to a low-quality local optimum.

Since Java source code is available and used for receiving the multi-touch tabletop data, the jahmm [28] HMM library for Java was chosen to execute the necessary HMM calculations. This library is able to train a given base model by supplying the set of example traces $\mathbb{O}$ and using the Baum–Welch algorithm, so no further programming is needed except for creating a digital representation of the base HMM.

However the training of CHnMMs is more complex and cannot be done with the usual Baum–Welch algorithm. Buchholz [3] developed an adapted version for CHnMMs which, however, can occasionally generate even worse models. We therefore decided to manually parametrise the CHnMMs in the manner of Bosse et al. [2]. This means that the example traces $\mathbb{O}$ were analysed, movement phases or states were identified and the expected value and standard deviation for the normal distributions of the transitions were calculated. Since only one symbol is expected to be emitted on each state change, namely the new direction, the probability for emitting this symbol is equal to one ($tr_i.b(v_{expected}) = 1$).

The previous sections covered the creation of the gesture models for the HMM and CHnMM recognition systems. The following sections show how these gesture models are used to classify an executed gesture.

3.4. Classifying the executed gesture

In this section the process of classifying an executed gesture is elaborated. This is the last aspect of the implementation that needs to be done before experiments can be conducted.

So far, we have a set of recognizable gestures $\{G_1, G_2, \ldots, G_n\}$ and their corresponding models $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. For classifying the trace O of the executed gesture a common approach is applied. The evaluation task is solved for every gesture model, i.e. $P_i(O|\lambda_i)$, $i = 1, \ldots, n$ is calculated. The model with the highest probability is the one that best describes the trace O, hence, the corresponding gesture G_i is output as the classified gesture. If all probabilities are zero, no model is able to produce the observed trace O: the performed gesture could not be classified and no gesture is output.

The evaluation of the HMM system is done by the jahmm [28] library method which is based on the forward algorithm from Section 2.1.1. For the CHnMM system, a similar approach is used, but the fact that there is an End-state in each model is exploited. So instead of calculating $P(O|\lambda)$, the forward probability of the End-state at the point of time of the last symbol emission T is taken as a measurement for the probability of a gesture. This forward probability is defined as

$$\alpha_T(s_{end}) = P(O \cap q_T = s_{end}|\lambda)$$

with s_{end} being the End-state and q_T being the state of the model at time T.

With this approach, only those traces are considered that reached the End-state and therefore are likely to represent completely performed gestures, whereas the original evaluation approach would also accept incomplete gestures. For example, a trace of a movement to the left could be classified as a left–right gesture because its model can create such a trace.

This technique could be easily applied to HMMs too, but the jahmm library is not suited for this approach. However, for the goal of this paper this is not a problem, because no incomplete gestures were executed in the experiments.

3.5. Calculating the forward probability

In this section, we describe how the forward probability is calculated for our gesture-describing CHnMMs. The algorithm is not available in programming frameworks as is the case for HMMs. Buchholz [3, pp. 37–49] developed a forward algorithm for CHnMMs based on the HMM forward algorithm and on the proxel method that is used in an adapted version to calculate the forward probability of the End-state $\alpha_T(s_{end})$.

The algorithm can be adapted or simplified respectively due to the fact that the gesture models do not contain any concurrent processes, i.e. there are no transitions with *aging* = *true*. As a result no age information of transitions need to be saved in a Proxel ($\rho.\tau = \emptyset$). Algorithm 1 shows the simplified version which takes the gesture CHnMM λ and an ordered list of observations O as input. Each observation $o \in O$ ($|O| = T$) contains the symbol and the time of its emission by the symbol calculation process.

The algorithm is initialised with the Proxel set R_0 in line 1 that contains a single Proxel that represents the start state s_{start} that has a probability of one, because every gesture is in that state at the beginning. The algorithm then iterates over all the observations (line 2) except the first one which is the dummy symbol GestureStart that indicates the beginning of the trace recording.

In each iteration, the previous Proxel set R_{t-1} is used to calculate the Proxel set R_t of the current one. This is done by creating the child Proxels of each Proxel in R_{t-1}, which is accomplished within the loop in line 6 in the following way: Based on the CHnMM λ and the state of the current Proxel $\rho.S$, all outgoing transitions of the state, i.e. all that are active for this Proxel, are determined and saved in variable *AT* (line 7). Hence, each element of *AT* is element of *TR* and therefore is described by a certain probability distribution *dist* and by a function $b(v)$ returning the output probability for a given symbol v. By knowing all active transitions *AT*, the sojourn probability $P_{sojourn}$ can be computed using the cumulative distribution function *dist*.*cdf* of each active transition (line 8). It represents the probability that there was no state change between $o_{t-1}.time$ and $o_t.time$.

For every active transition in *AT*, a corresponding child Proxel ρ' is created that represents the state which is reached when the transition fires. This successor state is denoted by *next*. The forward probability for the state represented by the Proxel is saved in each of it and denoted by α. Its calculation is done in line 11 and uses the hazard rate function *hrf* (also known as instantaneous rate function [10]) of the current probability distribution. The definition is $hrf(x) = \frac{pdf(x)}{1-cdf(x)}$ where *pdf* is the probability density function. It "represents the continuous rate of flow of the random variable represented by the firing of the corresponding transition" [9].

If the child Proxel ρ' has a forward probability of 0, it is not added to the current Proxel set R_t because the corresponding state cannot be reached with the given trace and model. This is an important part of the algorithm as it saves a lot of unnecessary calculations in the case that the trace of the gesture does not fit the current model. Since all gesture models are evaluated, an early cancellation of the processing for an inappropriate model helps to improve the performance of the recognition system.

Algorithm 1. CHnMMForward.

```
Input: O, λ
R_0 = {(s_start, 1)};
for t = 1 to T do
    v = o_t.symbol;
    Δt = o_t.time − o_{t−1}.time;
    R_t = ∅;
    foreach ρ ∈ R_{t−1} do
        AT = outgoingTransitions(ρ.S, λ);
        P_sojourn = ∏_{tr∈AT} (1 − tr.dist.cdf(Δt));
        foreach tr ∈ AT do
            μ = tr.dist.hrf(Δt);
            ρ' = (tr.next, ρ.α ∗ P_sojourn ∗ μ ∗ tr.b(v));
            if ρ'.α = 0 then continue;
            if ∃ρ'' ∈ R_t with (ρ''.S = ρ'.S) then
                ρ''.α+ = ρ'.α;
            else R_t = R_t ∪ {ρ'};
if ∃ρ ∈ R_T with (ρ.S = s_end) then
    return ρ.α;
else return 0;
```

The algorithm returns the forward probability α of the Proxel representing the End-state s_{end}, if such a Proxel exists or zero otherwise. Thanks to the simplification of the original algorithm, many unnecessary calculations can be avoided and the code can be simplified. As a result, a better performance should be achieved for the evaluation of a gesture CHnMM compared to the original algorithm by Buchholz.

4. Experiments and results

Both recognition systems need to be evaluated in order to compare their recognition quality and performance. Since the recognition systems are classifiers, the commonly used Precision/Recall metric can and will be used, although the classifiers are not binary. However, Sun and Lim [29] have shown how this metric can be applied to n-ary classifiers.

The precision of gesture G_i equals the number of gestures correctly classified as G_i divided by the number of gestures classified as G_i. Whenever the classifier returns G_i the precision value of this gesture states how trustworthy this result is. The number of gestures correctly classified as G_i divided by the number of performed G_i gestures is the recall of gesture G_i. It is a measurement of how complete the classification is.

For the experiments a fixed set of gestures is needed. Fig. 3 shows all six gestures that were chosen. They can be split into two groups of similar gestures: three DownUp and three circular gesture variants. This choice of gestures is significant because they are similar in shape but different in execution speed. Therefore this is a challenging gesture set for any gesture recognition system. Although using a multi-touch capable device, the gesture set does not have a focus on multi-touch gestures, but one is still included to test feasibility.

The expectation for the experiments is that the HMM-system will have problems distinguishing the similar gestures, whereas the CHnMM-system will have acceptable error rates. The necessary computation time is expected to be quite similar. However the CHnMM-system might be more efficient, because the number of computation steps is smaller due to the lower number of symbols created for the CHnMM-system because only state changes cause a symbol emission for the CHnMM-system.

4.1. First experiment

For the first experiment, each gesture was performed twenty times for each system and the symbol traces were recorded to produce training data $\mathbb{O}$ for each gesture and system. With this data for each gesture HMMs and CHnMMs were created and trained or manually parametrized respectively, as described in Section 3.3.

The result is an HMM and a CHnMM recognition system for the defined gesture set. Again, each gesture was performed twenty times for each system and the classified gesture was recorded. In case that no gesture could be fitted (all forward probabilities were zero), the attempt is recorded as not classified.

Table 1 presents an overview of the results obtained for the HMM-system. The recorded results are contained in the confusion

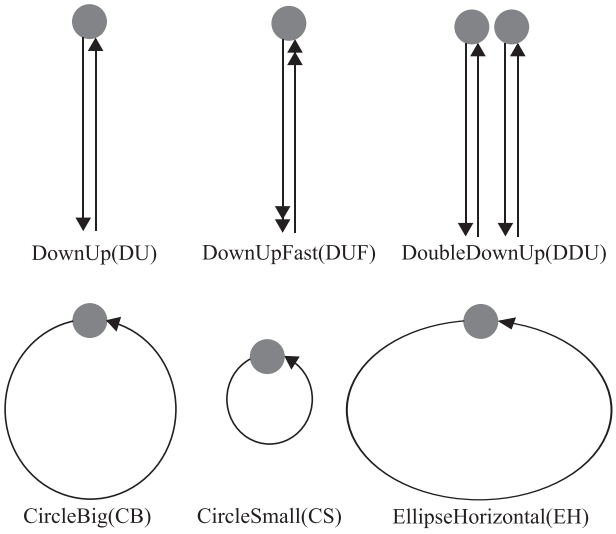

Fig. 3. Gesture set used for the experiments.

Table 1
Results with HMM-system.

Gesture	Correctly classified	Not classified	Precision	Recall
DU	10	–	0.526	0.500
DUF	17	–	1.000	0.850
DDU	9	2	0.409	0.450
CB	19	–	0.704	0.950
CS	17	–	0.944	0.850
EH	15	–	1.000	0.750
	87	2	0.737	0.725

Table 2
Results with CHnMM-system.

Gesture	Correctly classified	Not classified	Precision	Recall
DU	20	–	0.833	1.000
DUF	15	1	1.000	0.750
DDU	19	1	1.000	0.950
CB	19	1	0.950	0.950
CS	16	4	1.000	0.800
EH	17	2	1.000	0.850
	106	9	0.955	0.883

Table 4
Results with CHnMM-system and training data.

Gesture	Correctly classified	Not classified	Precision	Recall
DU	20	–	1.000	1.000
DUF	20	–	1.000	1.000
DDU	18	2	1.000	0.900
CB	20	–	1.000	1.000
CS	16	4	1.000	0.800
EH	16	4	1.000	0.800
	110	10	1.000	0.917

matrices in the Appendix A. The table shows an overall precision of 0.737 and a recall of 0.725. These values are worse than usually measured with HMM gesture recognition systems. The DownUp and DoubleDownUp gestures in particular are problematic and cannot be distinguished sufficiently well. Furthermore, the Ellipse-Horizontal gesture was classified as CircleBig five times. The other gestures reached acceptable recognition rates. In only two cases no classification could be made, probably due to symbol traces that did not occur while training. This phenomenon is also known as overfitting [30].

The results of the CHnMM-system are presented in Table 2 and show a precision of 0.955 and a recall of 0.883, which are clearly better than the ones of the HMM-system. The DownUpFast gesture has the worst recall, because it was classified four times as a DownUp gesture which is why its precision is lower than the rest. The reason for this is probably that the gesture was performed faster in the example data collection phase than it was performed in the experiments.

Another observation is that the general precision is better than the recall. This basically means that the classification results are trustworthy if there are any, since not all gestures were classified. For real applications where a certain recognized gesture activates a certain command or behaviour, the precision needs to be high as possible. Otherwise, a gesture is performed and the wrong command or behaviour is executed, resulting in a bad user experience.

4.2. Experiment with training data

As mentioned in the previous section, a possible source of error is the discrepancy of gestures performed during example data collection and classification. To avoid this, the experiment was repeated, but instead of performing gestures, the collected example traces $\mathbb{O}$ were utilised as input. This simulates the case that identical gestures are performed in the data collection and experimental phases.

The results in Tables 3 and 4 show much better precision and recall values for both systems, as expected. The CHnMM-system even reaches a perfect precision; its recall though is slightly lower than the one of the HMM-system. This is because the gesture models of the CHnMM-system were modelled quite strictly, i.e. they did not tolerate noisy data that occurred a few times in the example data collection process.

Table 3
Results with HMM-system and training data.

Gesture	Correctly classified	Not classified	Precision	Recall
DU	20	–	0.741	1.000
DUF	20	–	1.000	1.000
DDU	13	–	1.000	0.650
CB	20	–	0.952	1.000
CS	19	–	1.000	0.950
EH	20	–	1.000	1.000
	112	-	0.933	0.933

In contrast to the other gestures, the HMM-system has problems recognizing the DoubleDownUp-gesture. The reason seems to be that the HMMs of DownUp and DoubleDownUp are nearly the same, although the DoubleDownUp gesture emits twice as many symbols. However, an HMM cannot express this, because the probability of changing from Down-state to Up-state is approximately the same in both gestures and therefore both gestures are basically represented by the same HMM. This also explains the problems with this particular gesture in the first experiment and also illustrates a weakness of HMMs, even if it recognized the other gestures better than expected.

In both experiments, the time from finishing the gesture to getting the classification result was measured. This time was always 0 ms if the input data could not be classified. Otherwise the values were usually 31 or 32 ms and sometimes 47 ms for both recognition systems. Since the reaction times appear fixed, we assume that this is because of the scheduling side-effects of the operating system (Windows) or of the Java Virtual Machine and that the pure calculation time of the recognition systems is even shorter.

In summary, the results show that the performance of the CHnMM recognition system is comparable to that of the HMM-system and is thus also real-time capable. Furthermore, the experiments expose a weakness of HMMs, because even with the traces from example trace set $\mathbb{O}$ as input, the HMM-system cannot properly classify the DoubleDownUp gesture. It is often classified as DownUp instead, hence, these two gestures cannot be distinguished by the HMM-system, although there is a significant difference between the two traces. The DoubleDownUp gesture emits twice as many symbols compared to the DownUp gesture, because the same gesture is executed but with two fingers. With the exception of this gesture, the HMM-system performs very well, although the gesture set is challenging and no explicit timing information is utilised. This is possible because the HMM is fed with implicit timing information due to the periodic emission of symbols.

5. Conclusion

The goal of this work was to show that CHnMMs are applicable in the field of touch gesture recognition and that they are a viable alternative to HMM recognition systems. The results of the experiments show that the CHnMM-system reached a level of recognition quality that was the same as or better than the HMM. Additionally, both systems are on par regarding recognition speed, which makes CHnMM-systems also feasible for real-time scenarios. Therefore all goals of this work have been achieved.

Further observations suggest that HMMs are not well-suited to distinguishing certain gestures, as was seen with the DownUp and DoubleDownUp gestures, where both are described by nearly the same model. Nevertheless the HMM-system worked better than expected. It can be assumed that although HMMs are timeless, i.e. they do not utilize the timestamps of observations, they are fed with timing information through the periodic emission of symbols. For this reason, they are even able to distinguish similar gestures

with different execution speeds. We assume that systems that do not emit symbols periodically will have more problems when using HMMs.

A big advantage of HMMs is that they can be trained automatically with the Baum–Welch algorithm, making them easier to implement in comparison to the manual parametrisation of CHnMMs. For CHnMMs, an efficient training algorithm has been developed by Buchholz, whose quality in the field of touch gesture recognition needs to be evaluated in future work. If successful, the time-consuming and tedious process of manual parametrisation could become unnecessary.

Further experiments and extensions to this work could be:

- a bigger or more complex gesture set,
- investigating other types of symbols like accelerations or velocities,
- examine other methods for direction symbol calculation,
- more complex multi-touch gestures,
- comparison to other gesture recognition methods (artificial neural networks, dynamic time warping, support vector machines, etc. [31]),
- experiments with different touch devices.

In conclusion, CHnMMs should be considered for gesture recognition whenever a HMM recognition system is considered, because both model types can be utilised when a flexible system is required that easily allows the addition of a new gesture by simply performing it a few times and that also easily allows removal. However, a big advantage of a CHnMM-system is that it is independent of a periodic symbol emission. Thus, whenever it is difficult to create periodic symbol emissions, CHnMM should be preferred over HMM. This could be the case for touch devices where the touch data is read from a web browser via JavaScript. In this case, small time-lags are possible and common. However, the current disadvantage of CHnMMs is the manual parametrisation process that needs to be replaced by an automatic training process in order to make CHnMMs a real alternative to HMM recognition systems.

Appendix A. Confusion matrices of all experiments

See Tables 5–8.

Table 5
Confusion matrix of experiment results with CHnMM-system.

		Classified gesture					
		DU	DUF	DDU	CB	CS	EH
Executed gesture	DU	20	–	–	–	–	–
	DUF	4	15	–	–	–	–
	DDU	–	–	19	–	–	–
	CB	–	–	–	19	–	–
	CS	–	–	–	–	16	–
	EH	–	–	–	1	–	17

Table 6
Confusion matrix of experiment results with CHnMM-system and training data.

		Classified gesture					
		DU	DUF	DDU	CB	CS	EH
Executed gesture	DU	20	–	–	–	–	–
	DUF	–	20	–	–	–	–
	DDU	–	–	18	–	–	–
	CB	–	–	–	20	–	–
	CS	–	–	–	–	16	–
	EH	–	–	–	–	–	16

Table 7
Confusion matrix of experiment results with HMM-system.

		Classified gesture					
		DU	DUF	DDU	CB	CS	EH
Executed gesture	DU	10	–	10	–	–	–
	DUF	–	17	3	–	–	–
	DDU	9	–	9	–	–	–
	CB	–	–	–	19	1	–
	CS	–	–	–	3	17	–
	EH	–	–	–	5	–	15

Table 8
Confusion matrix of experiment results with HMM-system and training data.

3-6		Classified gesture					
		DU	DUF	DDU	CB	CS	EH
Executed gesture	DU	20	–	–	–	–	–
	DUF	–	20	–	–	–	–
	DDU	7	–	13	–	–	–
	CB	–	–	–	20	–	–
	CS	–	–	–	1	19	–
	EH	–	–	–	–	–	20

References

[1] S. Damaraju, A. Kerne, Multitouch gesture learning and recognition system, Tabletops Interact. Surf. (2008), http://ecologylab.cs.tamu.edu/research/publications/tabletop08-posterAbstract1-6.pdf.
[2] S. Bosse, C. Krull, G. Horton, Modeling of gestures with differing execution speeds: are Hidden non-Markovian Models applicable for gesture recognition, in: MAS: The International Conference on Modelling & Applied Simulation, 2011.
[3] R. Buchholz, Conversive Hidden Non-Markovian Models Dissertation (Ph.D. thesis), Otto-von-Guericke-Universität Magdeburg, 2012.
[4] L. Rabiner, B.-H. Juang, An introduction to Hidden Markov Models, IEEE ASSP Mag. 3 (1986) 4–16.
[5] G.A. Fink, Hidden Markov Models, in: Markov Models for Pattern Recognition, Springer, 2014, pp. 71–106.
[6] M.O.S.M. Elmezain, Hand gesture spotting and recognition using HMMs and CRFs in color image sequences (Ph.D. thesis), Otto-von-Guericke-Universität, 2010 http://edoc.bibliothek.uni-halle.de/servlets/DocumentServlet?id=9399
[7] G.D. Forney Jr., The Viterbi algorithm, in: Proceedings of the IEEE, 1973, pp. 268–278.
[8] D. Burshtein, Robust parametric modeling of durations in Hidden Markov Models, IEEE Trans. Speech Audio Process. 4 (1996) 240–242.
[9] C. Krull, G. Horton, Hidden non-Markovian Models: formalization and solution approaches, in: Proceedings of 6th Vienna International Conference on Mathematical Modelling, Vienna, 2009, pp. 682–693.
[10] G. Horton, A new paradigm for the numerical simulation of stochastic petri nets with general firing times, in: Proceedings of the European Simulation Symposium, 2002, pp. 129–136.
[11] S. Lazarova-Molnar, The proxel-based method: formalisation, analysis and applications (Ph.D. thesis), Otto-von-Guericke-Universität Magdeburg, Universitätsbibliothek, 2005.
[12] M.S. Arulampalam, S. Maskell, N. Gordon, T. Clapp, A tutorial on particle filters for online nonlinear/non-GAUSSIAN Bayesian tracking, IEEE Trans. Signal Process. 50 (2002) 174–188.
[13] M. Isard, A. Blake, Condensation-conditional density propagation for visual tracking, Int. J. Comput. Vis. 29 (1998) 5–28.
[14] S. Mitra, T. Acharya, Data Mining: Multimedia, Soft Computing, and Bioinformatics, John Wiley & Sons, 2005.
[15] S. Mitra, T. Acharya, Gesture recognition: a survey, systems, man, and cybernetics, IEEE Transactions on Part C: Applications and Reviews 37 (2007) 311–324.
[16] B. Yegnanarayana, Artificial Neural Networks, PHI Learning Pvt., Ltd., 2009.
[17] I. Steinwart, A. Christmann, Support Vector Machines, Springer, 2008.
[18] J. Yamato, J. Ohya, Ishii K., Recognizing human action in time-sequential images using Hidden Markov Model, in: Proceedings of the 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition 1992, CVPR'92, IEEE, 1992, pp. 379–385.
[19] M. Elmezain, A. Al-Hamadi, J. Appenrodt, B. Michaelis, A Hidden Markov Model-based continuous gesture recognition system for hand motion trajectory, in: 19th International Conference on Pattern Recognition, ICPR 2008, IEEE, 2008, pp. 1–4.
[20] F.-S. Chen, C.-M. Fu, C.-L. Huang, Hand gesture recognition using a real-time tracking method and Hidden Markov Models, Image Vis. Comput. 21 (2003) 745–758.
[21] C.-Y. Kao, C.-S. Fahn, A human-machine interaction technique: hand gesture recognition based on Hidden Markov Models with trajectory of hand motion, Procedia Eng. 15 (2011) 3739–3743 ({CEIS} 2011).

[22] J. Yang, Y. Xu, Hidden Markov model for gesture recognition, Technical Report, DTIC Document (1994).
[23] B. Denkena, B. Dengler, K. Doreth, C. Krull, G. Horton, Interpretation and optimization of material flow via system behavior reconstruction, Prod. Eng. 8 (2014) 659–668.
[24] E. Argones R’ua, J.L. Alba Castro, Online signature verification based on generative models, IEEE Trans Syst. Man Cybern. B 42 (2012) 1231–1242.
[25] J.Y. Han, Low-cost multi-touch sensing through frustrated total internal reflection, in: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology – UIST '05, 2005, pp. 115–118.
[26] N.U.I. Group, Community Core Vision 1.4, 2011 http://ccv.nuigroup.com
[27] M. Kaltenbrunner, T. Bovermann, R. Bencina, E. Costanza, Tuio: a protocol for table-top tangible user interfaces, in: Proceedings of the 6th Int'l Workshop on Gesture in Human–Computer Interaction and Simulation, 2005.
[28] J.-M. François, Jahmm, 2009 http://code.google.com/p/jahmm/
[29] A. Sun, E.-P. Lim, Hierarchical text classification and evaluation, in: Proceedings 2001 IEEE International Conference on Data Mining, 2001.
[30] D.M. Hawkins, The problem of overfitting, J. Chem. Inf. Comput. Sci. 44 (2004) 1–12.
[31] D. Wood, Methods for Multi-Touch Gesture Recognition, 2008.

Tim Dittmar has been working at the Department of Computer Science at the Otto-von-Guericke-University Magdeburg since January 2013 as a research and teaching assistant in the field of Simulation and Modeling after successfully defending his Master thesis in November 2012. His current research interests are Hidden non-Markovian Models, new authentification methods and touch gesture recognition.

Claudia Krull has been working at the Department of Computer Science at the Otto-von-Guericke-University Magdeburg since October 2003 as a research and teaching assistant in the field of Simulation and Modeling. In April 2008 she successfully defended her PhD thesis titled "Discrete-Time Markov Chains: Advanced Applications in Simulation". Her current research interests are Virtual Stochastic Sensors for the analysis of partially observable discrete stochastic systems.

Graham Horton received his PhD in Computer Science in 1991 and his 'Habilitation' in 1998 from the University of Erlangen, in the field of simulation. Since 2001, he has been a Professor of Simulation and Modeling in the Computer Science Department of the Otto-von-Guericke-University of Magdeburg. His research interests include efficient solution algorithms for Markov chains, discrete simulation and modeling of systems, computer aided innovation processes and idea engineering.