Declarative Gesture Spotting using Inferred and Refined Control Points

Lode Hoste, Brecht De Rooms and Beat Signer
Web & Information Systems Engineering Lab, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium

Keywords: Gesture Spotting, Gesture Recognition, Continuous Rule Language.

Abstract: We propose a novel gesture spotting approach that offers a comprehensible representation of automatically inferred spatiotemporal constraints. These constraints can be defined between a number of characteristic control points which are automatically inferred from a single gesture sample. In contrast to existing solutions which are limited in time, our gesture spotting approach offers automated reasoning over a complete motion trajectory. Last but not least, we offer gesture developers full control over the gesture spotting task and enable them to refine the spotting process without major programming efforts.

1 INTRODUCTION

Over the last few years, we have witnessed an increasing interest in gesture recognition due to new devices such as tablet computers or Microsoft's Kinect controller. Template and machine learning-based gesture recognition approaches have been subject to research for many years. However, most prominent statistical analysis-based solutions require the definition of the start and end points of potential gestures which is typically enforced by letting the user execute a special action while performing a gesture. This segmentation of a continuous motion trajectory data stream into a number of gestures defined by their start and end points is called *gesture spotting*.

Gesture spotting is a challenging problem which has seen limited exploration to date (Just, 2006), especially in so-called "always on" user interfaces where the gestures are mixed with noise in the form of continuous non-gesture data. Applications like multi-touch-based window managers, controller-free home automation solutions or surveillance applications have to process a vast amount of continuous motion data often containing only a few meaningful gestures. Simple ad-hoc solutions such as motion thresholding can easily result in a major processing overhead for statistical classifiers or, if defined too strictly, miss some of the gestures.

We propose a novel gesture spotting approach for continuous streams of two- or three-dimensional Cartesian coordinates, offering fine-grained control over the segmentation process based on spatial and temporal constraints between a number of automatically inferred control points. Control points are characteristic points describing the curvy areas or larger directional movements of a given motion. When describing a gesture, the gesture designer has to find a trade-off between a detailed definition and the necessary flexibility in terms of gesture variability. The presented solution for automatic control point detection can be further augmented with expert knowledge to refine spatiotemporal constraints between control points. Our approach focusses on the three aspects of *processing efficiency*, the *external representation* of automatically inferred control points and support for the *incorporation of expertise*.

Gesture spotting takes place before the gesture recognition process and our gesture spotting solution should be optimised for a high recall in order to minimise the number of missed gestures. This might imply that we are going to achieve a lower precision which is not a major problem since the spotted gestures are verified via existing gesture classification solutions. The recall performance is further increased by supporting a number of variation properties to relax the spatial constraints between control points.

We start in Section 2 by introducing our continuous gesture spotting solution. An evaluation in the form of experimental results is provided in Section 3. In Section 4, we discuss related gesture spotting approaches, before concluding with a discussion of our gesture spotting solution in Section 5.

Hoste L., De Rooms B. and Signer B. (2013).
Declarative Gesture Spotting using Inferred and Refined Control Points.
In *Proceedings of the 2nd International Conference on Pattern Recognition Applications and Methods*, pages 144-150
DOI: 10.5220/0004238301440150

2 GESTURE SPOTTING

Our gesture spotting approach is based on an incremental evaluation to find a sequence of control points in a large amount of trajectory data. The control points are automatically inferred based on a single well representative gesture sample. The current implementation uses a tangent-based calculation where major changes in a small section of the trajectory are stored as potential characteristic control points. The top m points are then chosen while preserving a good spatiotemporal distribution over the trajectory to ensure that not only distinctive curves but also longer straight lines are used for differentiation.

There are for example no distinctive spatial cues in the *flick right* gesture shown in Figure 1, but our control point inferencing mechanism provides a satisfying result with the fairly distributed control points `c1` to `c4`. By encoding spatial and temporal constraints between detected control points, a developer has full control over which parts of a gesture should be matched closely and where variation is desired.

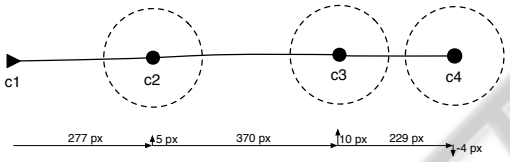

Figure 1: Flick right gesture with control points.

We opted for a simple but effective solution to automatically infer control points where the result can be visualised and manually refined by the gesture developer. Gesture spotting focuses on segmentation rather than classification which implies that we should aim for a high recall by introducing potential variation. However, the spotting process should still impose clear gesture-specific restrictions in order to minimise the computational classification overhead.

2.1 External Representation

As highlighted by Kadous (Kadous, 1999), the comprehensibility of existing gesture spotting and recognition approaches is rather limited and it is hard to know when a black box classifier is trained sufficiently in terms of generality or preciseness. We offer an external representation of the automatically inferred control points and the resulting human-readable program code can be refined or tailored to a given application scenario. The automatically generated declarative program code for the *flick right* gesture in Figure 1, with a given default value for the circular areas surrounding the control points, is shown in Listing 1.

Listing 1: Semi-automatic flick right gesture spotting rule.

```
1  (defrule FlickRight
2     ?p1 ← (Point2D)
3     ?p2 ← (Point2D)
4     (test (< ?p1.time ?p2.time))
5     (test (inside_control_point ?p1 ?p2 277 5 76))
6     ?p3 ← (Point2D)
7     (test (< ?p2.time ?p3.time))
8     (test (inside_control_point ?p1 ?p3 647 15 76))
9     ?p4 ← (Point2D)
10    (test (< ?p3.time ?p4.time))
11    (test (inside_control_point ?p1 ?p4 946 11 76))
12    ; Manual refinement
13    (test (< (- ?p4.time ?p1.time) 1000))
14    (not (and ; Bounding Box
15              (Point2D (y ?b_y) (time ?b_time))
16              (test (> ?b_time ?p1.time)) ; After p1
17              (test (< ?b_time ?p4.time)) ; Before p4
18              (test (> (abs (- ?p1.y ?b_y)) 245)))) ; ΔY
19 =>
20    (call DynamicTimeWarping
21          (select-between ?p1.time ?p4.time)
22          (gesture-set "flick-right")))
```

The declarative code shown in Listing 1 uses unbound variables denoted by a question mark (`?`) to express a number of constraints to which the `Point2D` events have to adhere. The `FlickRight` rule starts with the open starting point `p1` and searches for a second point `p2` which matches the temporal and spatial constraint based on the distance between `p1` and `p2` (*lines 4* and *5*). *Line 4* states that the timestamp of the event matching `p1` should be smaller than the timestamp of `p2` and the matching points should be ordered in time. For multi-touch or full-body gesture recognition there are multiple options to apply these temporal constraints. Either the inferencing can be extended to deal with the analysis of movements happening at the same time or developers can use software composition to build more complex gesture patterns. *Line 5* makes use of our built-in C function `inside_control_point`, which performs a translation of the `x` and `y` coordinates of the first argument (point `p1`) with the given values of `277` and `5` pixels. The function returns `true`, if the second argument (point `p2`) lies within a circular area around point `p1` with a radius of `76` pixels. The same strategy is used for the remaining $m-2$ control points. For three-dimensional trajectories, we overloaded this function with a version with six arguments performing similar operations in three-dimensional space. We also provide spatial functions such as Euclidean distance and new functions can be implemented by the developer.

Finally, there is another temporal gesture constraint ensuring that all matched points occurred within a timespan of one second (*line 13*). Note that

this temporal constraint is adjusted manually since it strongly depends on the given gesture and scenario. *Lines 14* to *18* show additional constructs which control the spotting process when the trajectories leave a certain bounding box as described later in Section 2.5. If all the conditions are satisfied, an existing gesture recogniser is called with the spotted range of events as a parameter (*lines 20* to *22*). Note that a detailed definition of the applied rule language can be found in Scholliers et al. (Scholliers et al., 2011).

Our control point-based gesture spotting approach automatically matches a combination of events adhering to the defined constrained trajectory at spotting time without any lossy preprocessing steps. It further provides a number of powerful features described in the following subsections.

2.2 Non-subsequent Event Matching

Events that do not match the specified constraints are skipped and we call this the non-subsequent event matching property. It is important to note that a skipped event is not discarded but can form the starting point `p1` of another spotting or be part of an intermediate match with a combination of other events. This leads to a next property of our spotting approach with respect to overlapping submatches.

2.3 Overlapping Submatches

The overlapping of subparts from different gestures is a complex gesture spotting problem. Existing solutions require the specification of an exhaustive list of overlapping subgestures and the gesture spotting engine needs to block and wait for subsequent events before the spotting of a gesture can be finished. While in Alon et al.'s approach (Alon et al., 2009) this subgesture list is automatically generated by an offline classifier during the training phase, there are still two implicit cases where possible gestures are incorrectly rejected. First, for each frame only the best scoring candidate gesture is added to a candidate list. Second, in many cases the subgesture does not follow the exact trajectory of the supergesture and if the supergesture for example fails at a later stage, subgestures might be incorrectly rejected from the candidate list.

Similar problems in detecting overlapping gestures exist with state machine-based solutions. Whenever new data triggers the transition to the next state, subsequent data will not be used as a potential start transition. This is illustrated in Figure 2 showing the gesture to be spotted on the left-hand side and the ongoing processing on the right-hand side. Initially, the transitions to consecutive states are valid. However, at state `s3` the single state machine has to decide whether to start from `s1` or continue to wait for future data so that `s4` might still be reached. As future data is not available at the decision point, state machine approaches might miss valid spottings.

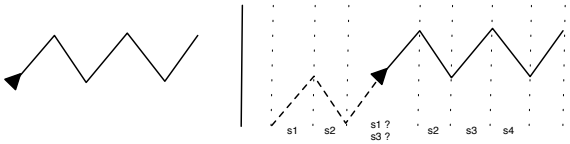

Figure 2: Overlapping submatches.

Since the idea underlying our approach is to search for a combination of events matching the declarative definition of gestures to be spotted, we inherently support overlapping submatches. Our current implementation is based on the CLIPS inference engine[1] and all possible paths are automatically stored in an incremental format for efficient processing (Forgy, 1982). With five active gesture spotting rules, we can for example process an average of 31 505 point events per second on an Intel Core i7 with 4 GB of RAM. This illustrates the low processing requirements of our solution for real-time gesture scenarios normally generating around 1200 events per second (20 fingers or joints with 60 Hz sampling rate).

2.4 Relaxed Spatiotemporal Constraints

Partially matched results are stored in a temporary storage. Spatial flexibility for matching noisy gesture variations is achieved by introducing a circular boundary (or a sphere for three-dimensional data) around each control point.

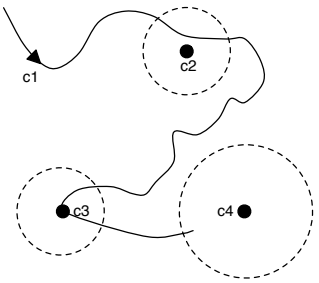

Figure 3: Noisy Z gesture.

Figure 3 shows the relaxed matching of a *noisy Z* gesture where we manually refined the spotting rule and increased the radius of the bounding circle for control point `c4` in order to be more flexible around that point. Most existing feature extraction-based spotting approaches are prone to false rejections for

[1] http://clipsrules.sourceforge.net

the given example. For instance, if the extracted symbols are too local, a lot of intermediate directional symbols not reflecting the three main directions of the Z gesture (i.e. right, diagonal down-left, right) are generated. This increases the computational overhead and requires an extensive amount of training samples with sophisticated filtering. On the other hand, if the extracted symbols are too global, we might miss small characteristic gesture movements.

Our declarative gesture spotting approach offers developers the flexibility to add user-defined spatial relations, such as changing the bounding circle to a rectangular or elliptic form. Similarly, temporal flexibility is provided for incorporating additional temporal relations. For instance, *line 13* of Listing 1 shows a refined temporal constraint between all points, while *lines 15* to *17* represent an expressive encoded temporal constraint (i.e. after `p1` but before `p4`). The relaxing of constraints is gesture dependent and human knowledge can be exploited to further control the spotting process via constructs such as negation.

2.5 Negation

Negation is a software engineering construct lacking in most statistically-based recognisers which typically use negative training or sample data to guide the classification in a certain direction. We argue that explicit negation is beneficial for both performance and accuracy. To illustrate this, let us have a look at the curved line shown in Figure 4.

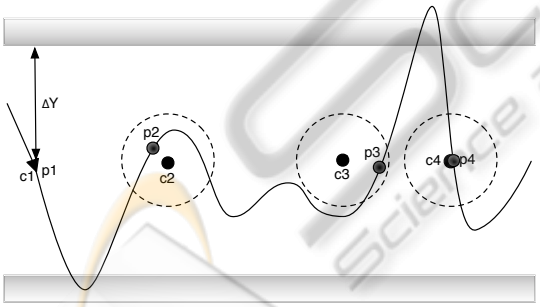

Figure 4: Curved line.

Although the motion does not describe a straight line, it would match the *flick right* gesture rule shown earlier in Figure 1. Fortunately, the incorporation of expert knowledge can be used to resolve this issue. In Listing 1, *lines 14* to *18* are negated to ensure that there is no point `q` between `c1` and `c4` whose difference on the y-axis (ΔY) is larger than 245 pixels.

2.6 Coupled Recognition Process

Whenever a gesture is spotted, an existing gesture classifier such as Dynamic Time Warping (DTW) (Darrell and Pentland, 1993) is applied with targeted template data. We call this synergy a coupled recognition process. Besides the fact that it is hard to find a single gesture spotting technique for the entire gesture set, the reuse of gesture spotting information is valuable for the final recognition process. This is shown on *lines 20* to *22* of Listing 1, where the optional `gesture-set` parameter defines a set of gestures for the template-based recogniser.

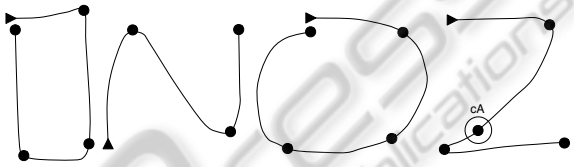

Figure 5: Automatically inferred control points.

This coupled recognition process can typically be used to deal with potentially conflicting gestures, such as circles and rectangles, which are not trivial to distinguish since the control points can be very similar as outlined in Figure 5. In this case, we might generate a single spotting rule and rely on the gesture classifier for distinguishing between circles and rectangles.

3 EVALUATION

In order to evaluate our spotting approach, we used the experimental data set by Wobbrock et al. (Wobbrock et al., 2007) consisting of 16 unistroke gestures and a total of 1760 gesture samples which have been captured by 10 subjects on a pen-based HP Pocket PC. While the data set consists of segmented unistroke samples, we concatenated the data with additional noise (5%) to simulate a single stream of continuous two-dimensional data input.

Table 1: Declarative gesture spotting performance.

r	**RC** (%)	**PR** (%)	**RC-E** (%)	**PR-E** (%)
22	77.50	52.10	78.75	56.50
24	83.13	47.16	84.38	52.53
26	90.63	42.40	91.25	46.79
28	93.75	39.47	94.38	43.26
30	97.50	35.37	97.50	39.29
32	98.75	32.78	98.75	36.41

For each of the 16 gestures, we used a single representative sample to infer the control points. Ta-

ble 1 highlights the performance of our gesture spotting approach with 4 to 6 control points per gesture and the angular method with a sliding window of 160 events. The default spatial variance of the control points is represented by the radius (r). The results in Table 1 consist of the recall (RC) as well as the precision (PR). Columns RC-E and PR-E represent the recall and precision of spotted gestures after applying expert knowledge to the single initial sample (e.g. more flexible matching for certain control points or use of negation to invalidate certain trajectories).

As we can observe in Table 1, the use of 4 to 6 automatically inferred control points per gesture allows for a high recall. The few non-spotted gestures originate from differences in the angle in which they were performed, which is a current limitation of our approach. However, in the near future, we plan to investigate methods to incorporate rotation invariance features. Note that our approach reasons over the complete trajectory, while still being able to process more than 400 times faster than real time for 60 Hz input and a gesture set consisting of 16 different two-dimensional gestures. The relatively high number of invalidly spotted gestures is caused by the fact that several gestures, such as the *left curly bracket* and *right curly bracket* are similar to spot as the *left square bracket* and *right square bracket*. Additionally, the *check* gesture is frequently found as a partial match of other gestures. However, we argue that a gesture spotting solution should be optimised for a high recall since the filtering of submatches can be done at the classification level.

To demonstrate the power of expert refinements, we modified the *right curly bracket* rule to prevent points between the first and final control point to be too far off to the left and did some other small refinements for other gestures. These minor changes to the model took only a few minutes but resulted in an increase of the precision without reducing the recall. In a broader context, such as full body gesture recognition where multiple concurrent trajectories are to be processed, the expression of additional conditions is of major importance for reducing invalid spottings and to improve the performance.

4 RELATED WORK

The classification of motion trajectories has been a research subject for many years and influential solutions such as Rubine's algorithm (Rubine, 1991), Dynamic Time Warping (DTW) (Darrell and Pentland, 1993), Neural Networks (NN) (Pittman, 1991) or hidden Markov models (HMM) (Wilson and Bobick, 1999) achieve good results for well-segmented motion trajectories. However, in online settings where gestures have to be classified while new data is being captured, these recognisers cannot be used as is.

Gesture segmentation is a complex task which is often addressed via ad-hoc solutions. Simple motion thresholding is an approach that is based on low-level parameters, including the velocity and change in direction, where users are asked to hold their hand still for a few seconds in between gestures. Lee and Kim (Lee and Kim, 1999) as well as Elmezain et al. (Elmezain et al., 2009) extended HMM by modelling continuous interaction via the addition of a garbage state. Nevertheless, this solution shows some problems in dealing with overlapping submatches and requires an increased number of training samples for both, gesture as well as non-gesture data, which further has to be tailored to the scenario.

The use of grammar rules (Holden et al., 2005; Kelly et al., 2011) significantly improves the spotting process by aiming for an initially high recall which then gets reduced by the grammar before extensive classification. However, in our scenario where gestures are mostly atomic commands to control various user interfaces, such grammar rules cannot be constructed. There are also no details about the computational overhead of the grammar rule-based approach and no external representation is offered.

Last but not least, Alon et al. (Alon et al., 2009) propose a spotting method that uses a continuous dynamic programming approach via pruning and subgesture reasoning. Similar to the HMM-based threshold model, the interaction is delayed for potentially overlapping gestures, which might not be optimal for certain interaction scenarios. In addition, the spotting process is reset after a gesture has been spotted, resulting in a loss of potentially overlapping gestures that have not been annotated.

5 DISCUSSION

We argue that current gesture spotting methods for continuous multi-touch or skeleton data streams should adhere to three main requirements. First, they should help to drastically reduce the vast amount of training data required for statistical-based methods since the data is too expensive to be acquired; especially when prototyping real-world applications. Second, gesture spotting algorithms should be comprehensible or offer an external representation allowing developers to visualise and refine automatically inferred results. Finally, the gesture segmentation process should aim for a minimal computational over-

head in order to process information in real time.

Our approach uses a single representative gesture sample to automatically infer a number of control points capturing the characteristic parts of the gesture. By offering an external representation of these control points, developers can visualise and further refine these points. Implicit support for overlapping submatches, relaxed spatiotemporal operators and additional programming constructs such as negation and user-defined conditions are key factors to ease the gesture spotting development. This includes the optimisation for a high recall, precision or processing performance based on the application scenario.

The manual refinement of gesture rules helps to achieve better results in the gesture spotting process. By automatically inferring *m* control points from a single gesture sample and compiling them into an extensible declarative rule, we support gesture developers in obtaining the intended continuous gesture recognition results. Inspired by mathematical line simplification schemes such as B-spline curve fitting (Cham and Cipolla, 1999), we plan to improve the current angle-based control point computation.

Given the use of expert knowledge, we plan to provide a graphical tool for three-dimensional trajectories based on ideas of Holz and Feiner (Holz and Feiner, 2009), where relaxed selection techniques can be annotated and manipulated graphically to ease the development process. While Holz and Feiner focussed on creating an interface for time series graphs with a single dimension, our graphical gesture development tool will address at least three dimensions.

As highlighted in Figure 5, the angle-based control point inferring technique is able to extract characteristic points from a sample trajectory. However, in this specific case, the control point `cA` is not optimal and might negatively influence the spotting performance. Another limitation of our current implementation is the lack of scale invariance. We can also not choose between a sub- or supergesture spotting. This application-dependent problem can be solved in the post-classification process, while the gesture spotting phase should focus on a high recall.

Our main goal was to improve the spotting of potential gestures in continuous data streams. By only requiring a single gesture sample and due to the possibility to programmatically refine the spotting process by loosening or tightening spatial and temporal constraints, we distinguish ourselves from existing spotting solutions. The external declarative representation of inferred control points has shown to be beneficial and complementary to programming constructs such as spatiotemporal operators, negation, user-defined functions and the invocation of coupled recognisers. Last but not least, due to the use of an efficient incremental evaluation engine the computational overhead of our gesture spotting approach is minimal.

ACKNOWLEDGEMENTS

The work of Lode Hoste is funded by an IWT doctoral scholarship.

REFERENCES

Alon, J., Athitsos, V., Yuan, Q., and Sclaroff, S. (2009). A Unified Framework for Gesture Recognition and Spatiotemporal Gesture Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9).

Cham, T.-J. and Cipolla, R. (1999). Automated B-Spline Curve Representation Incorporating MDL and Error-Minimizing Control Point Insertion Strategies. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(1).

Darrell, T. and Pentland, A. (1993). Space-Time Gestures. In *Proceedings of CVPR 1993*, New York, USA.

Elmezain, M., Al-Hamadi, A., and Michaelis, B. (2009). Hand Gesture Spotting Based on 3D Dynamic Features Using Hidden Markov Models. *Signal Processing, Image Processing and Pattern Recognition*, 61.

Forgy, C. L. (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19(1).

Holden, E.-J., Lee, G., and Owens, R. (2005). Australian Sign Language Recognition. *Machine Vision and Applications*, 16.

Holz, C. and Feiner, S. (2009). Relaxed Selection Techniques for Querying Time-Series Graphs. In *Proceedings of UIST 2009*, Victoria, Canada.

Just, A. (2006). *Two-Handed Gestures for Human-Computer Interaction*. PhD thesis, École Polytechnique Fédérale de Lausanne. Diss No. 3683.

Kadous, M. W. (1999). Learning Comprehensible Descriptions of Multivariate Time Series. In *Proceedings of ICML 1999*, Bled, Slovenia.

Kelly, D., McDonald, J., and Markham, C. (2011). Recognition of Spatiotemporal Gestures in Sign Language Using Gesture Threshold HMMs. *Machine Learning for Vision-Based Motion Analysis*.

Lee, H.-K. and Kim, J. H. (1999). An HMM-based Threshold Model Approach for Gesture Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10).

Pittman, J. A. (1991). Recognizing Handwritten Text. In *Proceedings of CHI 1991*, New Orleans, USA.

Rubine, D. (1991). Specifying Gestures by Example. In *Proceedings of SIGGRAPH 1991*, Las Vegas, USA.

Scholliers, C., Hoste, L., Signer, B., and Meuter, W. D. (2011). Midas: A Declarative Multi-Touch Interaction Framework. In *Proceedings of TEI 2011*, Funchal, Portugal.

Wilson, A. D. and Bobick, A. F. (1999). Parametric Hidden Markov Models for Gesture Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9).

Wobbrock, J. O., Wilson, A. D., and Li, Y. (2007). Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of UIST 2007*, Newport, USA.