Efficient Point-based Pattern Search in 3D Motion Capture Databases

Christian Beecks*‡ and Alexander Grass‡
*University of Münster, Data Management and Analytics, Germany
*Email: christian.beecks@uni-muenster.de
‡Fraunhofer Institute for Applied Information Technology FIT, Germany
‡Email: {christian.beecks,alexander.grass}@fit.fraunhofer.de

***Abstract*—3D motion capture data is a specific type of data arising in the Internet of Things. It is widely used in science and industry for recording the movements of humans, animals, or objects over time. In order to facilitate efficient spatio-temporal access into large 3D motion capture databases collected via internet-of-things technology, we propose an efficient *2-Phase Point-based Trajectory Search Algorithm* (2PPTSA) which is built on top of a compact in-memory spatial access method. The 2PPTSA is fundamental to any type of pattern-based investigation and enables fast and scalable point-based pattern search in 3D motion capture databases. Our empirical evaluation shows that the 2PPTSA is able to retrieve the most similar trajectories for a given point-based query pattern in a few milliseconds with a comparatively low number of I/O accesses.**

***Keywords*-3D motion capture data; spatio-temporal trajectories; point-based patterns; query processing; spatial indexing**

I. Introduction

3D motion capture data is considered to be a specific type of data arising in the Internet of Things. It captures and digitizes the movements of humans, animals, or objects over time in a three-dimensional Euclidean space. This type of data is widely used in academia and industry, for instance, for entertaining purposes, medical applications, film-making, and video game development. 3D motion capture data has the advantage of reflecting the aforementioned movements with the highest possibly degree of accuracy.

The spatio-temporal movements are recorded over time by means of one or multiple markers which are attached to the subjects' relevant parts. Each of these markers thus leads to a finite sequence of multi-dimensional points, referred to as *trajectory*, in the 3D motion capture data space. Besides individual multi-dimensional points, trajectories are thus atomic elements within 3D motion capture databases.

When analyzing and investigating 3D motion capture databases, i.e., when aiming at retrieving and mining trajectories in a top-down or bottom-up manner with respect to a user-specific information need, the question of how to formalize and evaluate queries [1], [2], [3] and how to find the most similar trajectories [4], [5], [6], [7], [8], [9], [10] efficiently becomes of major importance. In particular, searching the most similar trajectories with respect to a specific point-based pattern, which is one of the most fundamental operations in nearly any retrieval or mining approach, has to be carried out with the highest possible degree of efficiency and scalability.

In this paper, we propose the *2-Phase Point-based Trajectory Search Algorithm* (2PPTSA), which is a fast query processing algorithm for determining the most similar trajectories with respect to a given point-based query pattern. Supported by an in-memory spatial access method, the 2PPTSA first gathers possible candidate trajectories without any I/O operation and then filters out dissimilar trajectories by accessing the underlying motion capture database. In this way, the 2PPTSA minimizes the number of database accesses and maintains scalability. To sum up, we make the following contributions:

- We propose the *2-Phase Point-based Trajectory Search Algorithm* (2PPTSA) for fast and scalable query processing.
- We investigate the internal parameters of our proposal on a 3D motion capture benchmark database and empirically show the high efficiency of our approach.

The remainder of this paper is structured as follows: The problem of point-based trajectory search is introduced in Section II. Related work is presented in Section III. While the in-memory spatial index structure is discussed in Section IV, we propose the 2-Phase Point-based Trajectory Search Algorithm in Section V. Section VI provides the results of our performance analysis before we conclude the present paper with an outlook on future research directions in Section VII.

II. Point-based Trajectory Search

The problem of point-based trajectory search is characterized by a spatio-temporal query pattern and the task of determining the most similar trajectories with respect to that pattern. In this paper, we abstract from complex query patterns and formalize a *point-based query pattern* Q over the three-dimensional Euclidean space $\mathbb{R}^3$ as follows:

$$Q = \{q_i\}_{i=1}^{m} \subset \mathbb{R}^3 \tag{1}$$

In addition to a point-based query pattern, a *trajectory* $t \in \mathbb{T}$ of length n is defined as a finite sequence of points in $\mathbb{R}^3$ abstracting from concrete time information as follows:

$$t : \{1, \ldots, n\} \rightarrow \mathbb{R}^3, \tag{2}$$

where $t(i) = (x_i, y_i, z_i) \in \mathbb{R}^3$ represents the coordinates of trajectory t at time $i \in [1, \ldots, n]$. The *trajectory space* $\mathbb{T} = \bigcup_{k \in \mathbb{N}} \{t | t : \{1, \ldots, k\} \rightarrow \mathbb{R}^3\}$ comprises all trajectories and is a superset of a finite 3D motion capture database $db \subset \mathbb{T}$.

Given a point-based query pattern Q and a single trajectory $t \in db$, we model the dissimilarity $D(Q, t) : \mathcal{P}(\mathbb{R}^3) \times \mathbb{T} \rightarrow \mathbb{R}^+$ between Q and t by making use of a ground distance $\delta : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}$ as shown in the following equation:

$$D(Q, t) = \sum_{i=1}^{m} \min\{\delta(q_i, t_j) | 1 \leq j \leq n\} \tag{3}$$

Intuitively, each query point $q_i \in Q$ is matched to the most similar point t_j from trajectory t with respect to the ground distance δ, and these matching dissimilarities are linearly combined in order to determine the similarity relation between Q and t. The closer a trajectory passes the query points, the higher its similarity value and vice versa.

Based on this similarity model, we can now formally define the corresponding point-based trajectory search problem via a *k-nearest-neighbor query* as the smallest set $\mathrm{NN}_k \subseteq db$ with $|\mathrm{NN}_k| \geq k$ such that $\forall t \in \mathrm{NN}_k, \forall t' \in db \setminus \mathrm{NN}_k : D(Q, t) \leq D(Q, t')$. Hence, for a given set of query features Q, the task is to efficiently determine the k-most query-like trajectories in a database db such that the dissimilarity values of all other trajectories are greater than or equal to the dissimilarity values of the retrieved trajectories.

Before we show how to solve this problem efficiently by the 2-Phase Point-based Trajectory Search Algorithm, we briefly outline related work in the following section.

III. Related Work

Spatio-temporal patterns arising in 3D motion capture data are mainly investigated in terms of gesture recognition which aims at recognizing meaningful expressions of human motion including hand, arm, face, head, and body movements [11]. An extensive overview of the many facets of gesture recognition can be found for instance in the following surveys: [12], [13], [14], [15], [11], [16], [17], [18], [19].

Frequently encountered approaches for recognizing manual gestural patterns are based on *Hidden Markov Models* [20], [21], [22], [23] or more generally *Dynamic Bayesian Networks* [24]. More recent approaches are for instance based on *Feature Fusion* [25], *Dynamic Time Warping* [26], [27], *Longest Common Subsequences* [28], or *Neural Networks* [29].

The (dis)similarity between two spatio-temporal trajectories can be assessed for instance by the *Levenshtein Distance* [4], the *Minimal Variance Matching* [5], the *Longest Common Subsequence* [6], [7], the *Edit Distance with Real Penalty* [8], the *Edit Distance on Real Sequences* [9], or the *Mutual Nearest Point Distance* [10]. More complex multi-dimensional spatio-temporal patterns can be compared with the *Gesture Matching Distance* [30]. In particular the latter has been investigated in terms of efficient query processing [31], [32], [33], [34].

IV. In-Memory Spatial Index Structure

In addition to the trajectories stored in the underlying motion capture database db, a spatial index structure idx is used in order to aggregate the points of the trajectories and to prune dissimilar trajectories from the search space as early as possible.

We propose to index trajectory points individually by an in-memory spatial index structure. To this end, the trajectory points are aggregated by means of *nodes* $\mathcal{N} = \{n_i\}_{i=1}^{m}$ that are endowed with a *closure operator* $cl(n_i)$ such as a *minimum bounding rectangle*. Together with the closure operator, each node $n_i \in \mathcal{N}$ stores the identifiers of the corresponding trajectory points, which we assume to be integer-valued keys, i.e., $id \in \mathbb{N}$. In this way, each node compactly abstracts from the concrete trajectory points in $\mathbb{R}^3$ to the corresponding keys in $\mathbb{N}$, and allows for further compression. Furthermore, each node n_i supports the computation of the minimum distance $\delta^-(p, n_i)$ and maximum distance $\delta^+(p, n_i)$ between a point $p \in \mathbb{R}^3$ and the node n_i as follows:

- `minDist(Point p)`: This method is used to determine the minimum distance δ between a point p and the convex hull of n_i derived from the closure operator $cl(n_i)$. It is typically invoked with a query point in order to lower bound dissimilarities w.r.t. the indexed points. The result of n_i.`minDist(p)` is defined as:

$$\delta_p^- = \min\{\delta(p, g) \mid g \in cl(n_i)\} \tag{4}$$

- `maxDist(Point p)`: Equivalent to the previous operation, this method determines the maximum distance δ between a point p and the convex hull of n_i derived from the closure operator $cl(n_i)$. It is typically invoked with a query point in order to upper bound dissimilarities w.r.t. the indexed points. The result of n_i.`maxDist(p)` is defined as:

$$\delta_p^+ = \max\{\delta(p, g) \mid g \in cl(n_i)\}. \tag{5}$$

Based on the aggregation of points into nodes $\mathcal{N}$, any spatial index structure idx has to provide the following sorted-access method:

- `next(Point p)`: This method is used to incrementally traverse the index structure and return the next nearest node n_i with respect to the distance n_i.`minDist(p)`. It is typically invoked with a query point in order to perform *sorted access* within the index structure. Repeated calls to this method for the same point p result in an ordered sequence of nodes $n_{\pi(1)}, n_{\pi(2)}, \ldots, n_{\pi(m)}$ such that the following holds for $i \leq j$:

$$n_{\pi(i)}.\mathtt{minDist}(p) \leq n_{\pi(j)}.\mathtt{minDist}(p) \tag{6}$$

We implemented the generic methods above on top of a modified variant of the *R*-tree* [35], where the nodes only store the integer-valued keys of the corresponding trajectories together with their corresponding minimum bounding rectangles, which can be efficiently encoded by the vectors $lower, upper \in \mathbb{R}^3$ reflecting the node's minima and maxima in each dimension. The method `next(Point p)` was implemented by a priority queue as described in [36], in order to minimize the number of node accesses when traversing the index structure incrementally.

V. 2-Phase Point-based Trajectory Search Algorithm

The proposed 2-Phase Point-based Trajectory Search Algorithm (2PPTSA) is a fast query processing algorithm that utilizes a compact in-memory spatial index structure in order to efficiently process point-based queries with a miminum number of I/O cost. Due to the ability of exchanging the index structure and the stage-wise modularity of the algorithm, the 2PPTSA epitomizes a highly customizable and scalable solution for point-based access to large 3D motion capture databases.

The main concept of the 2PPTSA is to determine the most similar trajectories with respect to a point-based query pattern, cf. Section II, by exploiting an efficient consecutive stage-wise processing of indexed trajectory points. In the first phase – *candidate generation phase* – the 2PPTSA aims to generate candidate trajectories, which potentially contribute to the retrieval results, and to approximate their dissimilarities solely based on the in-memory spatial index structure. To this end, the trajectory points are incrementally retrieved from the index structure, i.e., the nodes comprising identifiers are retrieved, and the dissimilarities are approximated accordingly by lower and upper bounds. In the second phase – *candidate refinement phase* – the 2PPTSA aims to refine the candidates with the lowest dissimilarity approximations by computing their exact dissimilarity values $D(Q, \cdot)$, cf. Equation 3, by accessing the underlying motion capture database. To ensure that the number of I/O operations, i.e., number of database accesses, is minimal, the candidate trajectories are strictly refined in ascending order with respect to the dissimilarity approximations.

A. Algorithm

Require: 3D motion capture database db, index structure idx, point-based query pattern $Q = \{q_i\}_{i=1}^{m}$, nearest neighbors k, incremental k-NN range Δ_k
Ensure: k-nearest-neighbors $\mathrm{NN}_k \subseteq db$

1: $matchings \leftarrow$ Map<BitSet>
2: $lb \leftarrow$ Map<Double>
3: $ub \leftarrow$ Map<Double>
4: $candidates \leftarrow$ List<$(id, \widetilde{D}_{id})$>
5: $results \leftarrow$ MaxHeap<(t, D_{id})>
6: $\delta_{\max} \leftarrow idx.diameter()$
7: **repeat**
8: $\quad\theta \leftarrow 0$
9: $\quad$**for all** $q_i \in Q$ **do**
10: $\quad\quad$**for** $j = 0$ to Δ_k **do**
11: $\quad\quad\quad node \leftarrow idx.next(q_i)$
12: $\quad\quad\quad\delta^- \leftarrow node$.minDist($q_i$)
13: $\quad\quad\quad\delta^+ \leftarrow node$.maxDist($q_i$)
14: $\quad\quad\quad$**for all** $id \in node$ **do**
15: $\quad\quad\quad\quad$**if** $id \notin matchings$ **then**
16: $\quad\quad\quad\quad\quad matchings[id] \leftarrow$ BitSet(m)
17: $\quad\quad\quad\quad\quad ub[id] \leftarrow m \cdot \delta_{\max}$
18: $\quad\quad\quad\quad\quad candidates$.add($id, ub[id]$)
19: $\quad\quad\quad\quad$**if** $!matchings[id][i]$ **then**
20: $\quad\quad\quad\quad\quad matchings[id][i] \leftarrow true$
21: $\quad\quad\quad\quad\quad lb[id] \leftarrow lb[id] + \delta^-$
22: $\quad\quad\quad\quad\quad ub[id] \leftarrow ub[id] - \delta_{\max} + \delta^+$
23: $\quad\quad\quad\theta \leftarrow \theta + \delta^-$
24: $\quad candidates$.sortBy($\widetilde{D}_{id}$)
25: **until** $\theta > candidates[k]$
26: upperToLower($candidates$)
27: **while** $!candidates$.isEmpty() **do**
28: $\quad\langle id, \widetilde{D}_{id}\rangle \leftarrow candidates$.pop()
29: $\quad$**if** $results$.size() $= k \ \wedge$
$\quad\quad results$.peek().distance() $\leq \widetilde{D}_{id}$ **then**
30: $\quad\quad$**return** $results$
31: $\quad$**else**
32: $\quad\quad t \leftarrow db$.getTrajectory($id$)
33: $\quad\quad D_{id} \leftarrow$ dissimilarity(Q, t)
34: $\quad\quad results$.push($\langle t, D_{id}\rangle$)
35: $\quad\quad$**if** $results$.size() $> k$ **then**
36: $\quad\quad\quad results$.pop()
37: **return** $results$

After initializing the variables and data structures [lines 1-6], the algorithm executes the candidate generation phase [lines 7-25] prior to the candidate refinement phase [lines 26-37]. In doing so, the algorithm maintains the following variables: the variable $matchings$ contains the ids of the trajectories that have already been processed during the candidate generation phase and indicates the query points to which the minimum distances have been computed. The variables lb and ub store the lower and upper bound approximations of the trajectories. These variables are incrementally adapted. The variable $candidates$ comprises the candidate trajectory ids together with their dissimilarity approximations, while the variable $results$ contains the trajectories with their exact dissimilarity values. The latter is only used during the candidate refinement phase.

In the candidate generation phase [lines 7-25], the algorithm iterates over all query features $q_i \in Q$ and retrieves the next Δ_k nodes from the index structure idx, while updating the variables $matchings$, $candidates$, lb, and ub with the

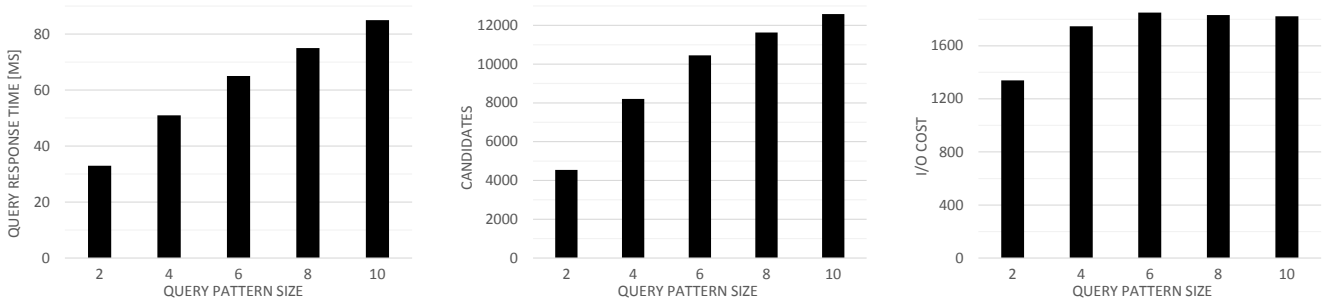

Figure 1. Performance results of the 2PPTSA by varying the query pattern size between 2 and 10.

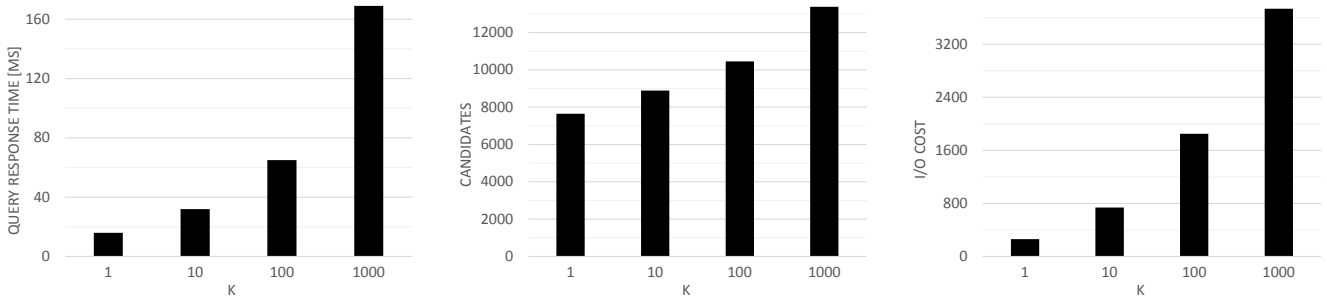

Figure 2. Performance results of the 2PPTSA by varying parameter k of the result size between 1 and 1000.

trajectory ids appropriately. In this way, the candidate trajectories are gathered with increasing radii around the query points and their lower and upper bound approximations are refined. The threshold θ is used to indicate the maximum dissimilarity of completely processed trajectories, i.e., the exact dissimilarity values of trajectories that have been completely processed and are below the threshold θ, and to early terminate the candidate generation phase. This phase terminates when k candidate trajectories have been found, whose upper bound approximations are smaller than the current threshold θ [line 25].

After terminating the candidate generation phase, all upper bound approximations are converted to lower bound approximations [line 26], and the candidate refinement phase is initialized. The algorithm then iterates over the $candidates$ in ascending order with respect to the lower bound dissimilarity approximations as long as better results can be found [line 29] and updates the $results$ with trajectories loaded from the underlying motion capture database db and their exact dissimilarity values where appropriate [lines 32-36].

It can be shown that the 2PPTSA is correct and that the algorithm returns the exact k-nearest-neighbors as specified in Section II. Moreover, based on the information gathered in the candidate generation phase and the processing of trajectories in the candidate refinement phase, it can also be shown that the number of database accesses, i.e., the number of I/O operations, is minimized by the 2PPTSA. We thus conclude, that the proposed algorithm provides an efficient and scalable solution for point-based query processing.

VI. Performance Analysis

In this section, we evaluated the performance in terms of efficiency of the proposed 2PPTSA by utilizing the *3D Iconic Gesture Dataset*[1] [37]. This dataset comprises 1,739 iconic gestures from 29 participants depicting entities, objects, and actions. Based on the provided 3D skeleton motion capture data, which was recorded via Microsoft Kinect, we extracted more than 35k trajectories from all available markers in the three-dimensional Euclidean space $\mathbb{R}^3$ which we additionally normalized to the interval $[0, 1]^3 \subset \mathbb{R}^3$.

The 2PPTSA was implemented in Java and the experiments were conducted on a 2.5 GHz machine equipped with 16 GB main memory. The default parameters are as follows: query pattern size $|Q| = 6$, number of nearest-neighbors $K = 100$, incremental k-NN range $\Delta_k = 40$, and a query workload of 100 randomly generated point-based query patterns.

The performance results of the 2PPTSA with respect to a varying query pattern size between 2 and 10 are shown in Figure 1. As can be seen in the figure, by increasing the size of the query patterns, i.e., the number of points in Q, from 2 up to 10, the query response times increase from 33ms to 85ms on average. Concomitant with this increase in time, the number of candidates that are examined in the candidate generation phase grows from 4,546 up to 12,583

[1] http://projects.ict.usc.edu/3dig/

on average. The I/O cost, however, approximately stay at an average value of 1,800 database accesses.

By varying the number K of nearest-neighbors, as shown in Figure 2, the aforementioned values increase. When the result size grows by a factor of 10, i.e., from $K = 100$ to $K = 1,000$, the query response time only increases by a factor of 2.6 from 65ms to 169ms, while the number of candidates and the I/O cost grow with an even smaller factor.

To sum up, our performance evaluation shows that the proposed 2PPTSA is able to process point-based queries efficiently with a minimal amount of I/O cost. In particular when indexing the utilized 3D motion capture database with an in-memory spatial index structure, as described in Section IV, our proposal enables fast point-based pattern search in the order of milliseconds and is thus a fundamental access method for further analyzing and mining 3D motion capture databases.

VII. Conclusions and Future Work

In this paper, we have addressed the problem of point-based trajectory search in 3D motion capture databases, as an example for a specific type of data arising in the Internet of Things. To this end, we have proposed the 2-Phase Point-based Trajectory Search Algorithm (2PPTSA) for fast and scalable query processing and shown that our approach is able to efficiently determine the most similar spatio-temporal trajectories with respect to point-based query patterns in the order of milliseconds.

The results reported in this paper are promising and indicate the performance of the proposed approach for searching and analyzing not only 3D motion capture data but also other internet-of-things data. As this approach is ongoing work, we intend to address the following research directions in the future: (i) investigation of different spatial, metric, and ptolemaic access methods [38] as well as SQL and NoSQL databases for indexing and storing trajectories, (ii) the investigation of complex query patterns, including the weighting and ordering of query points, (iii) the development of trajectory mining approaches on top of the 2PPTSA, and (iv) an in-depth empirical evaluation based on large-scale internet-of-things databases.

Acknowledgment

The project underlying this paper has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 723145 (COMPOSITION). This paper reflects only the authors' views and the commission is not responsible for any use that may be made of the information it contains.

References

[1] M. R. Vieira, P. Bakalov, and V. J. Tsotras, "Querying trajectories using flexible patterns," in *EDBT*, vol. 426, 2010, pp. 406–417.

[2] C. du Mouza, P. Rigaux, and M. Scholl, "Efficient evaluation of parameterized pattern queries," in *CIKM*, 2005, pp. 728–735.

[3] M. Hadjieleftheriou, G. Kollios, P. Bakalov, and V. J. Tsotras, "Complex spatio-temporal pattern queries," in *VLDB*, 2005, pp. 877–888.

[4] M. Hahn, L. Krüger, and C. Wöhler, "3d action recognition and long-term prediction of human motion," in *Computer Vision Systems*, 2008, pp. 23–32.

[5] L. J. Latecki, V. Megalooikonomou, Q. Wang, R. Lakaemper, C. A. Ratanamahatana, and E. Keogh, "Elastic partial matching of time series," in *PKDD*, 2005, pp. 577–584.

[6] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh, "Indexing multi-dimensional time-series with support for multiple distance measures," in *SIGKDD*, 2003, pp. 216–225.

[7] M. Vlachos, G. Kollios, and D. Gunopulos, "Elastic translation invariant matching of trajectories," *Machine Learning*, vol. 58, no. 2-3, pp. 301–334, 2005.

[8] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *VLDB*, 2004, pp. 792–803.

[9] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD*, 2005, pp. 491–502.

[10] S. Fang and H. Chan, "Human identification by quantifying similarity and dissimilarity in electrocardiogram phase space," *Pattern Recognition*, vol. 42, no. 9, pp. 1824–1831, 2009.

[11] S. Mitra and T. Acharya, "Gesture recognition: A survey," *Trans. Sys. Man Cyber Part C*, vol. 37, no. 3, pp. 311–324, May 2007.

[12] N. A. Ibraheem and R. Z. Khan, "Article: Survey on various gesture recognition technologies and techniques," *IJCA*, vol. 50, no. 7, pp. 38–44, 2012.

[13] R. Z. Khan and N. A. Ibraheem, "Survey on gesture recognition for hand image postures." 2012, pp. 110–121.

[14] J. LaViola, "A survey of hand posture and gesture recognition techniques and technology," *Brown University, Providence, RI*, 1999.

[15] J. Liu and M. Kavakli, "A survey of speech-hand gesture recognition for the development of multimodal interfaces in computer games," in *ICME*, 2010, pp. 1564–1569.

[16] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 1–54, 2015.

[17] S. Ruffieux, D. Lalanne, E. Mugellini, and O. A. Khaled, "A survey of datasets for human gesture recognition," in *Human-Computer Interaction. Advanced Interaction Modalities and Techniques - 16th International Conference*, 2014, pp. 337–348.

[18] R. Watson, "A survey of gesture recognition techniques," Trinity College Dublin, Department of Computer Science, Tech. Rep., 1993.

[19] Y. Wu and T. S. Huang, "Vision-based gesture recognition: A review," in *Gesture-based communication in human-computer interaction*. Springer, 1999, pp. 103–115.

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

[21] Y. Nam and K. Wohn, "Recognition of hand gestures with 3d, nonlinear arm movement," *Pattern recognition letters*, vol. 18, no. 1, pp. 105–113, 1997.

[22] A. Psarrou, S. Gong, and M. Walter, "Recognition of human gestures and behaviour based on motion trajectories," *Image and Vision Computing*, vol. 20, no. 5, pp. 349–358, 2002.

[23] H.-I. Suk, B.-K. Sin, and S.-W. Lee, "Hand gesture recognition based on dynamic bayesian network framework," *Pattern Recognition*, vol. 43, no. 9, pp. 3059–3072, 2010.

[24] ——, "Recognizing hand gestures using dynamic bayesian network," in *Automatic Face & Gesture Recognition*, 2008, pp. 1–6.

[25] J. Cheng, C. Xie, W. Bian, and D. Tao, "Feature fusion for 3d hand gesture recognition by learning a shared hidden space," *Pattern Recognition Letters*, vol. 33, no. 4, pp. 476–484, 2012.

[26] T. Arici, S. Celebi, A. S. Aydin, and T. T. Temiz, "Robust gesture recognition using feature pre-processing and weighted dynamic time warping," *Multimedia Tools Appl.*, vol. 72, no. 3, pp. 3045–3062, 2014.

[27] S. Bodiroža, G. Doisy, and V. V. Hafner, "Position-invariant, real-time gesture recognition based on dynamic time warping," in *International Conference on Human-Robot Interaction*, 2013, pp. 87–88.

[28] H. Stern, M. Shmueli, and S. Berman, "Most discriminating segment–longest common subsequence (mdslcs) algorithm for dynamic hand gesture classification," *Pattern Recognition Letters*, vol. 34, no. 15, pp. 1980–1989, 2013.

[29] H. Hasan and S. Abdul-Kareem, "Static hand gesture recognition using neural networks," *Artificial Intelligence Review*, vol. 41, no. 2, pp. 147–181, 2014.

[30] C. Beecks, M. Hassani, J. Hinnell, D. Schüller, B. Brenger, I. Mittelberg, and T. Seidl, "Spatiotemporal similarity search in 3d motion capture gesture streams," in *SSTD*, 2015, pp. 355–372.

[31] C. Beecks, M. Hassani, B. Brenger, J. Hinnell, D. Schüller, I. Mittelberg, and T. Seidl, "Efficient query processing in 3d motion capture gesture databases," *Int. J. Semantic Computing*, vol. 10, no. 1, pp. 5–26, 2016.

[32] D. Schüller, C. Beecks, M. Hassani, J. Hinnell, B. Brenger, T. Seidl, and I. Mittelberg, "Automated pattern analysis in gesture research: Similarity measuring in 3d motion capture models of communicative action," *Digital Humanities Quarterly*, vol. 11, no. 2, 2017.

[33] C. Beecks, M. Hassani, F. Obeloer, and T. Seidl, "Efficient query processing in 3d motion capture databases via lower bound approximation of the gesture matching distance," in *IEEE International Symposium on Multimedia*, 2015, pp. 148–153.

[34] ——, "Efficient distance-based gestural pattern mining in spatiotemporal 3d motion capture databases," in *IEEE International Conference on Data Mining Workshop*, 2015, pp. 1425–1432.

[35] N. Beckmann and B. Seeger, "A revised r*-tree in comparison with related index structures," in *SIGMOD*, 2009, pp. 799–812.

[36] G. R. Hjaltason and H. Samet, "Ranking in spatial databases," in *SSD*, vol. 951, 1995, pp. 83–95.

[37] A. Sadeghipour, L.-P. Morency, and S. Kopp, "Gesture-based object recognition using histograms of guiding strokes," in *Proceedings of the British Machine Vision Conference*, 2012.

[38] M. L. Hetland, T. Skopal, J. Lokoč, and C. Beecks, "Ptolemaic access methods: Challenging the reign of the metric space model," *Information Systems*, vol. 38, no. 7, pp. 989 – 1006, 2013.