Alternative Representations and Abstractions for Moving Sensors Databases*

J. Eisenstein, S. Ghandeharizadeh, C. Shahabi, G. Shanbhag, and R. Zimmermann
Computer Science Department
University of Southern California
Los Angeles, California 90089

Abstract

Moving sensors refers to an emerging class of data intensive applications that impacts disciplines such as communication, health-care, scientific applications, etc. These applications consist of a fixed number of sensors that move and produce streams of data as a function of time. They may require the system to match these streams against stored streams to retrieve relevant data (patterns). With communication, for example, a hearing impaired individual might utilize a haptic glove that translates hand signs into written (spoken) words. The glove consists of sensors for different finger joints. These sensors report their location as a function of time, producing streams of data. These streams are matched against a repository of spatio-temporal streams to retrieve the corresponding English character or word.

The contributions of this study are two folds. First, it introduces a framework to store and retrieve "moving sensors" data. The framework advocates physical data independence and software-reuse. Second, we investigate alternative representations for storage and retrieval of data in support of query processing. We quantify the tradeoff associated with these alternatives using empirical data from RoboCup soccer matches.

1 Introduction

Applications involving moving *objects* and *sensors* pose numerous challenges for both information retrieval and database communities. *Moving objects* applications store the spatial coordinates of objects that change position over time [42, 12, 14, 8, 6, 24, 38, 33, 43, 34]. Example applications include those that query the location of trains, cars and planes during a time interval. One challenge investigated by recent studies is how to model the large spatio-temporal data to track the position of any object at any given time (either in the past, present or future). For example, storing new instances of an object every time that its location or shape changes (as proposed in [24, 6]) results in large and growing databases for continuously changing objects such as cars with GPS devices. Other studies propose techniques that only keep track of the current and predicted future positions of the moving objects [33] thus reducing the target data set size.

Moving sensors applications collect data from sensory devices that move in an environment. These sensors might be robots on the surface of Mars, MEMS-based devices inside the human body performing remote surgery, etc. This application class constitutes the focus of this study. While

*This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0091843 (SGER), and ITR-0082826, NASA/JPL contract nr. 961518, DARPA and USAF under agreement nr. F30602-99-1-0524, and unrestricted cash/equipment gifts from BMC, NCR, IBM, Intel and SUN.

it shares common characteristics with those of moving object applications, it has its own unique features. First, the number of sensors (or objects) for a given application is almost always fixed. That is, during the lifetime of an application, sensors are added and removed infrequently. Second, data from one sensor might provide context for the other sensors. In essence, the data from all sensors might convey more information than the sum of its pieces (specially across multiple time instances). Of course, the *independent* movement of a single sensor may still be of interest, but unlike cars with GPS devices, analyzing the location/value of all sensors in a frozen time frame (or a short time interval) becomes an important query. Here are three example applications for *moving sensors.*

American Sign Language (ASL) recognition: ASL is a gestural language used by hundreds of thousands of North Americans as their primary means of communication [23]. Machine recognition of ASL is still at a research level, and hidden Markov models [35, 40] along with classification and clustering techniques [7, 32] have been studied as possible approaches to recognize signs. One approach is to analyze the haptic data [29, 30] generated by a haptic device (e.g., CyberGlove from *http://www.virtex.com/products/hw_products/cybergrasp.html*) that is worn by the individual when signing. A high-end model of CyberGlove constitute of 22 sensors plus three sensors on the *Pholemus Fastrak* system for reporting hand coordinates in space. Hence, the number of sensors is fixed and small. In addition, to recognize a sign, the value of all sensors should be analyzed collectively.

RoboCup Soccer: RoboCup [17] is an annual robot competition in which teams of autonomous robots compete in a fast-changing environment on a well-known game: Soccer. RoboCup data is intrinsically spatio-temporal in nature, and automated analysis of this data is a significant problem for pattern recognition systems [1, 26]. A robot can be represented as a moving sensor in a fixed environment. Our proposed framework can capture the spatio-temporal characteristics of these moving sensors and facilitate query processing. An example is presented in Section 2.1. This environment consists of 23 sensors, one the ball and one for each of the 22 players (11 per team). The collection of sensors may be analyzed to identify if a team is in offside position or for statistical analysis.

Avatar Animation: In order to produce realistically animated models of humans in immersive environments [29], several sensory devices such as video cameras with tracking capabilities monitor and capture real human movements (e.g., the VaRionettes project at Lucent presented at SIGGRAPH 2000). For example, in support of teleconferencing using a low bandwidth communication channel, researchers in computer graphics have proposed the tracking of only a few feature points [9, 22], to be utilized for avatar animation. Figure 1 illustrates an example set of collected sensor data for the human face. In this case, although all the feature points are captured by a single sensor (a video camera), one can conceptualize each feature point as a "moving sensor." The avatar sensory data can be analyzed and queried for example for emotion recognition. The total number of tracked features (i.e., sensors) is about 18 and does not change. The collection of features must be analyzed in order to recognize face emotions.

The contributions of this study are two fold. First, we present a framework for these applications that advocates physical data independence and software reuse. Second, we focus on the three lowest layers of this framework and investigate alternative approaches to represent data produced by moving sensors. We evaluate a physical implementation of these approaches using data from the RoboCup Soccer matches.

To elaborate on the second contribution, in addition to trajectory models employed for moving

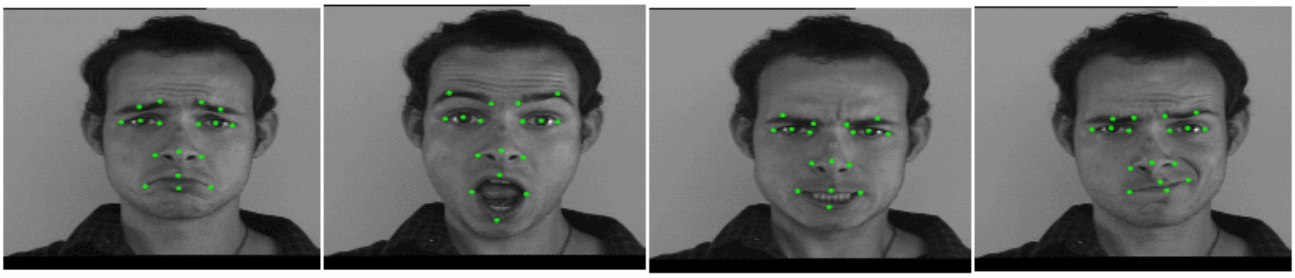

Figure 1: The avatar data type - a set of facial feature points as a function of time

objects (e.g., [12]), we propose a new *conceptual* representation, termed *frame-based*, that groups sensors per time instance. As demonstrated in Section 4, the frame-based representation improves the response time of certain query types. We study several *logical* representations to store moving sensor data in an object-relational database:

- MULT employs a single table to encapsulate the changing positions of multiple sensors over time. Each row of this table denotes data pertaining to a single sensor per time instance.
- UNIT also employs a single table. However, there is now one column for each sensor; a row corresponds to the value of all sensors per time instance. This representation is closest to the frame-based representation. It is less flexible than MULT because it is not trivial to introduce new sensors into the environment. This is not of significant importance for the class of applications that constitute the focus of this study.
- TBINS constructs one table for each sensor. Each row of this table contains the value of a specific sensor for a given time instance.

Of course, our evaluation considers alternative index structures with each design to enhance query processing time. UNIT and one of the indexed versions of MULT, MULT(t,s), are the closest to the frame-based model. The other indexed version of MULT, MULT(s,t), strives to cluster together the trajectory information of each sensor; TBINS is an extreme version of this approach.

Our experimental results reveal that UNIT and MULT are superior to TBINS. Queries which involve short time intervals and several sensors are more efficiently evaluated with UNIT and MULT(t,s), while MULT(s,t) is more appropriate for the opposite types of queries. Hence, the representation choice depends on the database workload. Given the availability of cheap mass storage devices, one may store data redundantly and choose the appropriate representation at runtime to process a specific query.

The remainder of this paper is organized as follows. Section 2 outlines a framework to represent moving sensors. In Section 3, we describe our several alternative approaches to represent moving sensors. Section 4 uses RoboCup as an example application to demonstrate the tradeoff between MULT and UNIT. Section 6 offers brief conclusions and our future research directions.

2 A Framework

Our framework for representing moving sensors is based on two fundamental concepts:

- Physical data independence: the ability to modify the physical characteristics of the sensors without a rewrite of the software that implements either the spatio-temporal constructs or the application layers that employ this data. As an example, assume our target application is a system that recognizes the American Sign Language, ASL. Physical data independence enables the framework to function with both an expensive haptic sensing device that consists of many sensors, e.g, Virtual Technologies CyberGlove with 22 angular sensors, and more economical devices that provide fewer sensors, e.g., Mattel's PowerGlove with only four angular sensors.

- Software modularity and extensibility to facilitate re-use: the ability to reuse representations and constructs employed at different data granularity. For example, the spatio-temporal representation of data is a challenge at each hierarchy level, e.g., sensory data, objects that constitute a hand, etc. This motivates the design and implementation of a general purpose software that is extended and specialized for re-use in different layers.

Figure **??** shows the different layers of our framework. As one traverses its different layers starting with the bottom one, each layer becomes more abstract away from the physical sensors and more relevant to the target application. We start with a description of the layers. Next, we describe RoboCup as an example to demonstrate each layer. Finally, we describe the transformation procedures that facilitate data exchange between these layers.

Sensor Data: is the lowest level of the hierarchy and describes the data produced by the sensors. This data is dependent on the input device. While some preprocessing is allowed at this layer, e.g., smoothing, no application-specific features are extracted. For example, one may store the data as a table in a relational database management system where the number of columns is equal to the total number of sensors offered by the device and the rows represent the samples collected as a function of time. In the Cyberglove, there are 31 columns, corresponding to sensors for angle, position, and velocity. The Mattel Powerglove, a more impoverished sensing device, provides only seven columns of data - an angular sensor for four fingers, along with position sensors for the entire hand. (Section 3 explores other alternatives that represent a sensor value as a table row instead of a column.)

We do not describe application specific processes, algorithms and filters at this level for three reasons. First, this layer is entirely device-dependent; if a new sensing device becomes available, none of the previously gathered data can be reused. Second, querying data at this low level of representation is nonintuitive, since the data is expressed in the vocabulary of the sensing device, rather than in the vocabulary of the application itself. For example, it is more intuitive to ask, "when was the user's index finger bent?" instead of asking, "when was the value of angular sensor A_8 greater than 60 degrees?" Finally, we believe that automatic clustering and classification algorithms may perform better when presented with a more abstract representation scheme. In our future work, we intend to investigate whether a richer representation does indeed improve the performance of those classification systems studied in [7].

Object Data: defines a set of objects, which are composed of points and regions. A set of standard predicates and operations can be applied to objects, regions, and points. Some of the predicates include touch, pass_through, and distance; the operators include border, intersection, union, etc. A good description of these predicates for 2-dimensional spatio-temporal data is provided in [11].

The objects that are defined at this level are specified by the application. For example, if we were interested in collecting data about the human hand, then relevant objects would include the position of each fingertip and the palm, and the angles at various joints. With the Powerglove, some of this data might be approximated from the sensor data. If instead, we are concerned with a set of RoboCup soccer playing robots, the relevant objects would be the position and orientation of each robot (derived from the angle the robot is facing), along with the ball, and perhaps the goals and sideline. While the objects found at this level are based on features of the task, the representation scheme is neutral with respect to the task, as are the predicates and operators. In addition, the representation is neutral with respect to the device. The positional coordinates of each object are determined from the sensors using the methods described in the ASL example of the previous section, so as to obtain physical data independence.

The advantage of working at this level of representation is that it is the most basic common denominator between various tasks and devices. It is independent of task, since the predicates and operations are general. It is also independent of the physical input device. Consequently, applications that are built around data at this level of description are the most portable and reusable.

Subject Data: In the ASL example, the device-neutral representation was informally described in terms that are specific to the human hand. For example, we spoke about objects such as the index finger, knuckles, and the wrist, and properties such as extension. In RoboCup, we would speak in terms such as goalie, ball, pass, and goal. This stands in contrast to the Object Data level of representation, in which queries had to be formulated in generic terms, such as "When was object O_3 touching object O_7?" but not in terms of subject-specific features, such as "When was the index_finger extended?"

In order to pose queries at this level, the user must: a) define a taxonomy of object-specific terms and predicates, and b) map these terms onto features at the lower level of representation. For example, the term "Index Finger" can be mapped onto objects O_3, O_5, and O_8, which might represent the positions of the fingertip and two knuckles. Higher-level predicates, such as *Extended(finger)* can be computed from the positions and characteristics of the lower-level objects. The programmer specifies how these functions are computed, although some automatic solution based on machine learning may be possible. In this case, the user would present examples of each predicate to a classification system, which could infer general rules for determining the value of each predicate.

There are several advantages to working at this relatively high level of abstraction. First, it corresponds well to the way humans think about problems such as hand sign recognition. Many tutorials in sign language express their instructions in terms that are amenable to this level of representation. It is hard, if not impossible, to imagine a sign language tutorial working in the low level terms of object data. Furthermore, we hope to show that this level of abstraction is more useful than raw data for machine learning systems, which are known to benefit from feature extraction [4].

Task data: In our sign-language example, the previous level of representation made it possible to formulate queries about the hand itself. The representations used at this level are still somewhat general. They are not specific to the sign language, and could just as easily have been used in a telepresence application such as virtual surgery. A fourth and final level of representation is necessary if the user wishes to formulate queries that are specific to the task itself. In the sign language case, the user might ask when a particular sign was made; in the virtual surgery case,

a teacher might want to query when a medical student made a particular incision. Such queries provide greatest value to a user who is unfamiliar with the specifics of the system, and who therefore may not have the ability to formulate queries at the lower levels of abstraction.

Again, a taxonomy of application-specific terms and predicates must be defined. At this high level of abstraction, it seems unlikely that the derivational rules for connecting with the lower levels of abstraction could be specified by hand. Some kind of pattern-recognition will likely have to be employed. Still, it is hoped that the highly structured nature of our data-representation scheme will enable this pattern recognition to be far more simple than a traditional approach where classification is performed on the original, low-level data.

2.1 An Example: RoboCup Soccer

Thus far, we have described the framework using the sign-language as our target application. This section demonstrates the power of our data representation hierarchy using the RoboCup soccer application. We use the high-level Task data queries to describe general features of the entire field of play, rather than of a single player. For our example, let's suppose that a RoboCup "coach" is looking at her team's game logs, trying to tighten up the defense. One relevant query is "select all events in which the opposing team scored by achieving a breakaway."

The first step is to decompose this query into one or more queries on the Subject level. The query can be rewritten as a conjunction: "select all events such that the opposing team scores with a breakaway formation," or more formally "select goal (opponents) and breakaway (opponents)." Goal is already a Subject Data query, and require no further decomposition. We can define Breakaway(A) to be true when a robot from team A is closer to team B's goal than any robot from team B, except the goalkeeper[1]. More formally, at time T, we define: Breakaway (A) $\equiv$ For all robots B_i $(2 \leq i \leq 11)$, there exists some robot A_j $(1 \leq j \leq 11)$ such that Distance $(B_i, Goal_B) >$ Distance $(A_j, Goal_B)$.

Team B's goalie is B_0, so this definition permits the goalie to be the closest player to the goal. Note that in this case it was possible to perform to derivations between the Subject and Task Data by hand, and without the help of any pattern recognition techniques. The distance predicate used in this derivation is an Object Data predicate, since it is general enough to apply to almost any type of spatio-temporal data, and is not specific to RoboCup. Hence, the decomposition of this part of the query to the Object Data representation level is trivial. However, the decomposition of the goal predicate, which occurs on the level of Subject Data, is more interesting. At time T: Goal_scored $(goal_n) \equiv$ pass_through (ball, $goal_n$).

Finally, we can decompose all of these queries into Sensor Data. Informally, pass_through (O_1, O_2) is true iff at time T_1, intersection $(O_1, O_2) =$ null, and then at some later time T_2, inside $(O_1, O_2) \neq$ null. Thus, pass_through is defined in terms of other Object Data queries. Intersection can be defined by directly examining the sensors values, thus bringing us all the way back to the level of Sensor Data. The distance predicate can also be defined directly in terms of Sensor Data.

[1]We are not insisting that the breakaway player has the ball. This was left out for clarity; it should be obvious how this additional condition might be modeled.

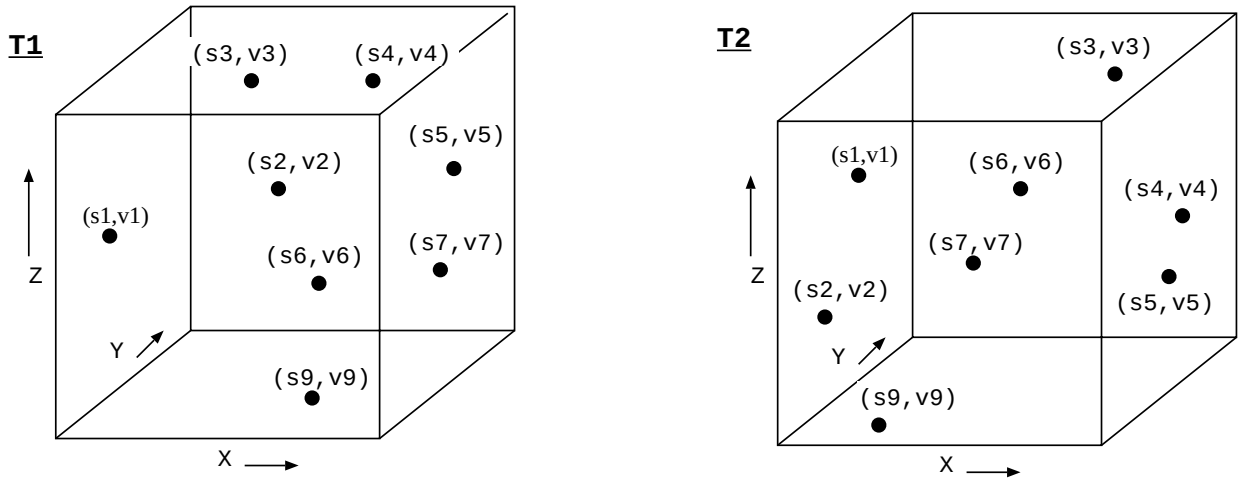

Figure 2: A 3-D representation of the sensors at 2 different time frames. Position is represented by the x, y, z dimensions. The value of each sensor is given as a ($sensor_{id}$, value) pair.

2.2 Transformation Procedures

Transformation procedures for generating higher level predicates from low-level sensor information is clearly one of the most important issues for users of the proposed framework. Such procedures can be relatively trivial, as in the case of the goal_scored predicate in the example above. Such procedures can easily be handcrafted by many users. In other cases, these procedures are more complex. For example, the trigonometric analysis required to find the position of a fingertip from the values of several angular sensors. Finally, some cases will require the use of automatic pattern recognition algorithm. The recognition of ASL signs from Subject Data describing the configuration of the hand seems to be such a case. In our future work, we hope to build systems that support the user in the transformation of data between the levels of this representation hierarchy.

In the next section, we present the conceptual design for spatio-temporal constructs that store and retrieve the moving sensory data.

3 Conceptual Representation of Sensor Data

This section outlines our approaches to represent the moving sensor data. These representations can be used at the three lowest layers of our hierarchical framework[2]. The central idea of encapsulating the data derived from a small, fixed number of moving sensors leads to two orthogonal conceptual representations. We term these as the frame based and trajectory based representations.

The frame based representation captures the data from all the sensors in a sequence of single, frozen time frames. Each time frame is a 3-dimensional (x, y, z) cube that describes the sensors along with their data. Figure 2 illustrates this representation. The trajectory based representation captures the data for each sensor over time. This can be illustrated by a solid line in 3-dimensional space, where each point on the line represents the position and value of the sensor at a particular time. Figure 3 illustrates the trajectory based representation.

[2]We did not discuss these representations at the task data layer because that layer is application specific.

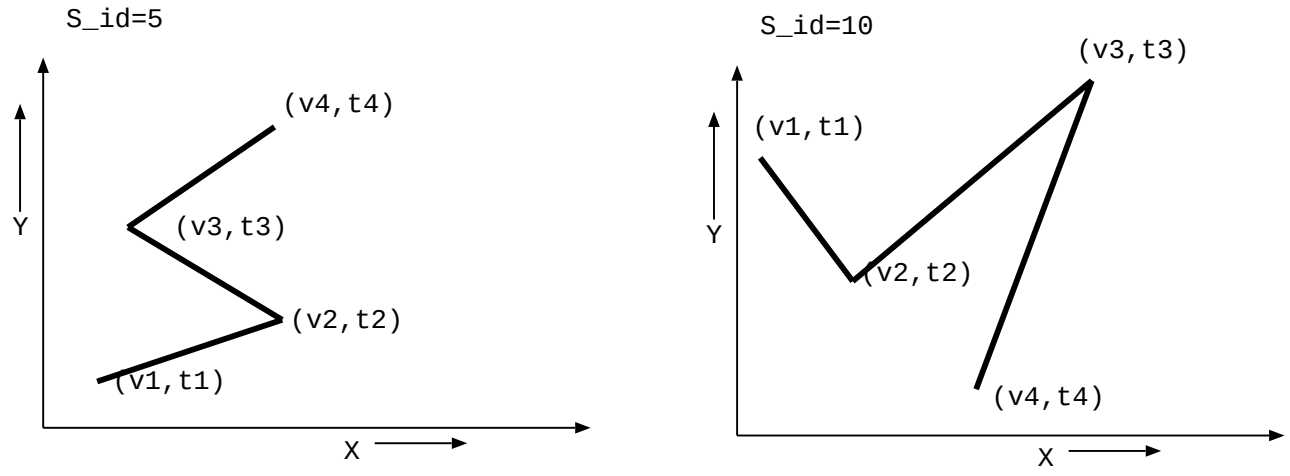

Figure 3: A 3-D representation of two sensors (s_{id}=5 and s_{id}=10), position is represented by two dimensions X and Y, and value of the sensor is labeled on the line.

Based on these conceptual representations, we propose three logical representations: MULT, UNIT, and BINS. All are based on the relational (or object-relational) model.

3.1 The MULT representation

An obvious choice of representation is to create a single table with one row per sensor and time instant. This results in multiple rows for each of the sensors over time. We term this representation as a MULTiple Time/Sensor encapsulation (MULT). Figure 4 illustrates a MULT representation for a group of moving sensors.

time	$sensor_{id}$	x	y	Value
100	0	100	104	-77
124	1	102	15	102
131	2	101	115	12
141	1	102	15	98
...	...	...	...	...

Figure 4: MULT representation

One may construct index structures to minimize the number of disk I/O required to process a query. We investigated a B-tree index using a composite key consisting of sensor and time. We analyzed the two possible combination: 1) MULT(s,t) employs sensor as the first portion of the composite key, and 2) MULT(t,s) with time as the first portion of the composite key. Intuitively, MULT(s,t) clusters together all information for each sensor and MULT(t,s) clusters together all information for a time frame. Figures 5(a) and 5(b) illustrate the MULT(s,t) and MULT(t,s) representations. Thus, MULT(t,s) is similar to the frame based representation and MULT(s,t) is similar to the trajectory based representation. In our experiments (see Section 4), we used Informix Dynamic Server (IDS) 9.2 to implement MULT. Although we cannot confirm the detailed implementation of composite index structures with IDS, our results support our intuition about the clustering behavior of MULT(t,s) and MULT(s,t).

In our experiments, we also considered scenarios where there is only a single B-tree index on time attribute, MULT(t), or on the $sensor_{id}$ attribute, MULT(s), or two B-trees on both attributes, MULT(t/s). However, all these alternatives were outperformed by either MULT(t,s) or MULT(s,t). Hence, we do not discuss them any further. We also considered R-tree index on sensor coordinates and/or time attributes. However, this performed drastically worse than B-tree because of how IDS implements R-trees[3]. We intend to investigate multidimensional index structures as part of our future research studies.

The MULT representation provides flexibility to add/delete sensors easily because it simply involves adding or removing a tuple from the table. The MULT representation results in less redundancy if we assume that sensors move and/or emit information independently. For example, it is possible that one sensor moves while others stay still. In this case, only time tuples corresponding to the moving sensor needs to be appended to MULT. However, if all sensors move and emit information collectively (e.g., CyberGlove sensors) then several time tuples need to be added for all the sensors.

$sensor_{id}$	time	x	y	Value
0	100	100	104	-77
0	101	102	15	102
...	...	...	...	...
1	100	101	115	12
1	101	102	15	98
...	...	...	...	...

a. MULT(s,t)

time	$sensor_{id}$	x	y	Value
100	0	100	104	-77
100	1	102	15	102
...	...	...	...	...
100	22	10	105	112
101	0	101	115	12
101	1	102	15	98
...	...	...	...	...

b. MULT(t,s)

Figure 5: Indexed MULT representations

3.2 The UNIT representation

For those applications whose sensors move collectively, it may be better to capture the information of all sensors in a time instant into a single object. The UNIT representation encapsulates information for this object over time in a single table. Figure 6 illustrates the UNIT representation for a set of 23 moving sensors. Each individual row in the table contains the data from all the sensors in a particular time instant. This abstraction of information from all sensors into one object closely resembles the frame based conceptual representation. A B-tree index is kept on the time attribute ensuring efficient retrieval of the objects for temporal queries.

UNIT is not flexible for adding/deleting sensors as it involves adding/deleting columns in the table. In addition, a large number of sensors results in a large tuple size, which impacts the query response time drastically. However, for those moving sensors applications that deal with a small and fixed number of collective sensors, the UNIT representation may be the superior alternative. With UNIT, a tuple is added when the position and/or value of a single sensor changes. If we assume that sensors move/emit independently, then this representation results in a great deal of

[3]With IDS, the attributes indexed by an R-tree must be defined as geopoints, a more complex data type (when compared with short integers), resulting in increased query processing and response times.

redundancy. However, with all of the example applications considered in this paper (which we categorize as the class of moving sensors), the positions and/or values of most of the sensors would change per time unit collectively. Moreover, by keeping the information of all sensors for a given time instant together, the query response time of UNIT is expected to be low. Our experimental results verify this fact since with moving sensors class of applications majority of queries probe all the sensors collectively for a short time interval.

Time	s0x	s0y	s0Value	...	s22x	s22y	s22value
1	12	14	-7	...	12	15	-10
2	12	15	-12	...	20	23	2
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Figure 6: UNIT representation

3.3 The BINS representation

Finally, the information related to each sensor or a time instant could be kept in separate tables. This approach creates BINS or repositories of information for time/sensor. We have studied the following representations based on the BINS approach.

- *SBINS*

  SBINS (Sensor BINS) hold information for all the sensors in a single time instant. The SBINS representation for moving sensors has a set of tables depending on the number of discrete time instants. This closely resembles a frame based conceptual representation with data from all sensors in a time instant in one table. The general class of moving sensor applications deal with a large number of time frames. This renders the SBINS approach impractical due to the extremely large number of tables created for moving sensor applications.

- *TBINS*

  TBINS (Time BINS) hold information over time for each sensor. A TBINS representation would thus have a set of tables, each having complete temporal evolution for a sensor. The idea of abstracting information per sensor follows the trajectory based conceptual representation. The TBINS representation is feasible for the general class of moving sensors applications with a small number of fixed sensors. Figure 7 shows the TBINS representation for a group of sensors. For efficient retrieval of data from each of the TBINS, we indexed the data using a B-tree index on the time attribute for each table.

With TBINS, the update operation is as simple as MULT because it only involves appending new tuples to the BINS. In addition, similar to MULT it is more appropriate when sensors move independently (i.e., only the corresponding BIN needs to be updated). However, queries across several sensors are both complicated to write and are expensive to perform because of the multiple join operations involved. We consider TBINS in our experiment for the sake of completeness.

time	position	Value
100	(10,104)	-77
101	(12,15)	102
...	...	...

TBINS(Sensor 0)

...

time	position	Value
100	(3,104)	-7
101	(12,15)	10
...	...	...

TBINS(Sensor 22)

Figure 7: TBINS representation

4 Performance Evaluation

Our experiments were performed on a Sun Sparc Enterprise 250 Server with dual UltraSparc 2 chips and 0.5 gigabyte of memory, running Solaris 2.6. For the implementation of the two representations, we used the Informix Dynamic Server 2000 with a Geodetic Datablade Spatial Extender v3.0.

4.1 RoboCup Data

The dataset consisted of 31 log files taken from the RoboCup domain. Each log file was preprocessed and broken into 23 separate data files - one for each player, and one for the ball. The average size of each of these data files was roughly 100 kilobytes. The data files included 6000 frames, with each frame representing one tenth of a second. After preprocessing the data, we stored five attributes per time frame: player number, x-position, y-position, angle, and game state. Consequently, the size of the data was roughly 21.39 MB; $31 \times 23 \times 6000 \times 5 = 2.139 \times 10^7$. The spatial granularity was 1680 for the x-position and 1088 for the y-position; the angular granularity was 360 degrees.

4.2 Query Performance Cost

A query may reference either sensors, time, or both using either a conjunctive or a disjunctive predicate. In each case, it might perform either an exact-match, a range lookup, or combine the value of different sensors with one another. For this performance evaluation, we considered conjunctive predicates that reference both sensor and time. We considered four possible query types: $S_{Range}T_{Exact}$, $S_{Range}T_{Range}$, $S_{Exact}T_{Range}$, and $S_{Exact}T_{Exact}$. (S stands for sensor, T denotes time, Range implies a range predicate while Exact identifies an exact match predicate.) We eliminate $S_{Exact}T_{Exact}$ query from further consideration because the representation that does well for $S_{Range}T_{Exact}$ (or $S_{Exact}T_{Range}$) with low selectivity for sensor would provide comparable performance. The queries pertaining to the other three possiblities are:

1. $S_{Range}T_{Exact}$ query type: Was team A in offside at time frame X?
2. $S_{Range}T_{Range}$: Was there at least one defender in the goalie box within the 10 seconds (100 frames) prior to the scored goal?
3. $S_{Exact}T_{Range}$: How often was the ball on team A's half court during intervals 30 and 240?
4. $S_{Range}T_{Exact}$: At time X, how many players of team A were within 2 unit distance from the goalie of team B?

Note that both query 1 and 4 are $S_{Range}T_{Exact}$ query types.

SELECT count(*) **FROM** sensor0, sensor1, ..., sensor22 **WHERE**((sensor0.*time*=X AND sensor1.*time*=X AND ... sensor22.*time*=X) AND (TcompareX-teamB(sensor1.*position*, sensor12.*position*,..., sensor22.*position*, sensor0.*position*) OR TcompareX-teamB(sensor2.*position*, sensor12.*position*,..., sensor22.*position*, sensor0.*position*) OR ... TcompareX-teamB(sensor11.*position*, sensor12.*position*,..., sensor22.*position*, sensor0.*position*)));	**SELECT** count(*) **FROM** unittable **WHERE** (S1x<S12x AND S1x<S13x AND ... S1x<S22x) OR (S2x<S12x AND S2x<S13x AND ... S2x<S22x) OR ... (S11x<S12x AND S11x<S13x AND ... S11x<S22x));	**SELECT** count(*) **FROM** multtable **WHERE**(($sensor_{id}$=1 AND time=X AND FcompareX-teamB(1,X)) OR ($sensor_{id}$=2 AND time=X AND FcompareX-teamB(2,X)) OR ... ($sensor_{id}$=11 AND time=X AND FcompareX-teamB(11,X)));
a. TBINS Approach	b. UNIT Approach	c. MULT Approach

Figure 8: SQL Syntax for Query 1

Query 1: To evaluate the first query, the x-position of each player on team A has to be compared with the x-position of the last defender on team B.

Figure 8(a) shows query 1 for the TBINS representation. We perform a join on all the sensor tables on the predicate where the time is X. We count if the row across the joined tables satisfies the offside requirement. If the count is zero then there is no offside, else team A is in offside. Figure 9(a) illustrates the function *TcompareX-teamB*. The function compares each player position of Team A to all players of team B except the goalie and to the ball.

Figure 8(b) depicts query 1 on the UNIT representation. In a single row for time X, we check if any player from team A (sensors 2-11) is behind the last defender of team B (sensors 13 - 22). If there exists any such player, team A is in offside.

Figure 8(c) shows query 1 for the MULT representation. For Time X, we check the x-position of each player of team A (except the goalie) with the x-positions of all players of team B. If any player is behind any player of team B, team A is in offside. Figure 9(b) illustrates the function *FcompareX-teamB*. This function compares the x-position of the player to each player of team B except the goalie and to the ball

Query 2: Figure 10(a) shows query 2 for the TBINS representation. We perform a join on all the sensor tables on the predicate where the time is between T-100 and T. We count if the selected rows across the joined tables has any player of team A (sensors 2-11) behind the goalie box.

CREATE FUNCTION TcompareX-teamB (position point, b1 point, ..., b10 point, ball point) **RETURNS** bool; return((point.X < b1.X) AND (point.X < b2.X) AND ... (point.X < b10.X) AND (point.X < ball.X)); **END FUNCTION;**	**CREATE FUNCTION** FcompareX-teamB (snum int, timex int) **RETURNS** bool; define position integer; select x into position where sensor=snum and time=timex; return((position < SELECT x FROM multtable WHERE time=timex AND $sensor_{id}$=12)AND (position < SELECT x FROM multtable WHERE time=timex AND $sensor_{id}$=13)AND ... (position < SELECT x FROM multtable WHERE time=timex AND $sensor_{id}$=22)); **END FUNCTION;**
a. Function TcompareX-teamB for TBINS	b. Function FcompareX-teamB for MULT

Figure 9: Functions defined for Query 1

Figure 10(b) depicts query 2 on the UNIT representation. We count the number of rows between Time T and 100 frames before it, where there is at least one player in the goalie box. For each row between times T-100 to T, we check if any player of team A (sensors 2-11) is behind the goalie box.

Figure 10(c) shows query 2 for the MULT representation. Between times T-100 and T, we count if any of the defenders (sensors 2-11) were behind the goalie box.

Query 3: For the third query, we need to examine every frame to identify the position of the ball sensor.

Figure 11(a) depicts query 3 for the TBINS representation. It simply counts the number of rows (time frames) where the position of the ball ($sensor_0$) is less than the half line. The count is multiplied by 0.1 (the sampling rate is 10 frames per second) to obtain the actual time the ball

SELECT count(*) **FROM** sensor2, sensor3, ..., sensor11 **WHERE** ((sensor2.time>=T-100 AND sensor2.time<=T) AND ... AND (sensor11.time>=T-100 AND sensor11.time<=T)) AND (sensor2.position.x<=goalx OR ... OR sensor11.position.x<=goalx)	**SELECT** count(*) **FROM** unittable **WHERE** (time>=T-100 and time<=T) AND (s2x<=goalx OR s3x<=goalx OR ... s11x<=goalx);	**SELECT** count(*) **FROM** multtab **WHERE** (time>=T-100 AND time<=T) AND ($sensor_{id}$ >=2 AND $sensor_{id}$ <=11) AND x<=goalx
a. TBINS Approach	b. UNIT Approach	c. MULT Approach

Figure 10: SQL Syntax for Query 2

SELECT 0.1 × count(*) **FROM** sensor0 **WHERE** (time>30 AND time<240) AND (x<halfline-position);	**SELECT** 0.1 × count(*) **FROM** unittable **WHERE** (time>30 AND time<240) AND (S0x<halfline-position);	**SELECT** 0.1 × count(*) **FROM** multtable **WHERE** (time>30 AND time<240) AND ($sensor_{id}$=0) AND (x<halfline-position);
a. TBINS Approach	b. UNIT Approach	c. MULT Approach

Figure 11: SQL Syntax for Query 3

was on A's side.

Figure 11(b) depicts query 3 for the UNIT representation. We count the number of rows (time frames) where the ball sensor ($sensor_0$) is less than the half line. The count is multiplied by 0.1 (the sampling rate is 10 frames per second) to obtain the actual time the ball was on A's side.

Figure 11(c) depicts query 3 for the MULT representation. It counts the number of rows (time frames) where the sensor number is 0 and position of the sensor ($sensor_0$) is less than the half line. The count is multiplied by 0.1 (the sampling rate is 10 frames per second) to obtain the actual time the ball was on A's side.

Query 4: Figure 12(a) depicts query 4 on the TBINS representation. A join is performed on all players from team A and then for time X, the function *calcdist* (see Figure 13(b)) finds all the sensors within 2 units distance to sensor 12.

Figure 12(b) depicts query 4 on the UNIT representation. It involves selecting all players from team A (sensors 2-11) and then counting those players that are within 2 units from the goalie of team B (sensor 12). The function *playcount* (see Figure 13(a)) is used to count the sensors in an unit (tuple).

Figure 12(c) depicts query 4 for the MULT representation. It involves selecting the sensors (2-11) and counting those sensors from them, whose distance from sensor12 is less than or equal to 2 units at the specified time.

SELECT calcdist(sensor2.*position*, sensor3.*position*, ..., sensor12.*position*) **FROM** sensor1, sensor2, ..., sensor12 **WHERE** (sensor1.*time*=X AND sensor2.*time*=X AND ... sensor12.*time*=X);	**SELECT** playcount(s2x, s3x, ..., s11x, s12x) **FROM** unittable **WHERE** time=X;	**SELECT** count(*) **FROM** multtable **WHERE** time=X AND ($sensor_{id}$ >=2 and $sensor_{id}$ <=11) AND (x-(SELECT x FROM multtable WHERE $sensor_{id}$=12 AND time=X)<=2);
a. TBINS Approach	b. UNIT Approach	c. MULT Approach

Figure 12: SQL Syntax for Query 4

CREATE FUNCTION playcount(s2x int, s3x int, ..., s12x int)
RETURNS int;
DEFINE count int; LET count=0;
IF ((s2X-s12X)<= 2) THEN
LET count=count+1; END IF
IF ((s3.X-s12.X)<= 2) THEN
LET count=count+1;END IF
...
IF ((s11.X-s12.X)<= 2) THEN
LET count=count+1;END IF
RETURN count;
END FUNCTION;
	CREATE FUNCTION calcdist(s2 point, s3 point, ..., s12 point)
	RETURNS int;
	DEFINE count int; LET count=0;
	IF (distance(s2.X,s12.X)<= 2) THEN
	LET count=count+1; END IF
	IF (distance(s3.X,s12.X)<= 2) THEN
	LET count=count+1;END IF
	...
	IF (distance(s11.X,s12.X)<= 2) THEN
	LET count=count+1;END IF
	RETURN count;
	END FUNCTION;

a. playcount function for UNIT	b. calcdist function for MULT

Figure 13: Functions defined for query 4.

4.3 Experimental Results

We conducted our experiments on the four queries. The results shown for each query were averaged over 100 runs, where each run was conducted at different time instants and/or sensors to prevent the distortion of the result due to the caching of the previous query results in the database. The coefficient of variance over these runs was between *1% to 10%*, which shows the independence of our results from a specific run.

Query type	TBINS	UNIT	MULT(s,t)	MULT(t,s)
Query 1	5.571	**0.034**	2.534	0.0571
Query 2		**0.057**	1.07	0.189
Query 3	0.0373	0.093	**0.028**	1.14
Query 4	5.998	0.013	0.231	**0.012**

Table 1: Response time of the queries in seconds

As shown in Table 1, Queries 1 and 4 performed significantly better with the UNIT representation. This is because each of these queries can be evaluated by examining several sensors in a single time frame. As discussed in Section 4, MULT(t,s) clusters the sensors for a time frame together and hence shows a fairly good response time for both Queries 1 and 4. Query 1 with the UNIT representation executed more than *75 times* faster than that with the MULT(s,t) or TBINS. Query 4 for a UNIT representation also shows more than *18 times* improvement in response time when compared with MULT(s,t) or TBINS. Query 2 involves a range of sensors over a range of time instants. The TBINS approach fails for this query because of the prohibitive cost of join for a time range across multiple tables.

Query 3 performed significantly better with MULT and TBINS representation because each of these queries is evaluated by examining a single sensor (one player) across several time frames. The UNIT approach still has a fairly good response time for query3. Moreover, the MULT(s,t) representation of Query 3 shows *30 times* improvement in response time over the corresponding MULT(t,s) representation.

We conclude that the UNIT model is best suited for moving sensors applications as it consistently performs well for all the spectrum of queries. However we confirm that a tradeoff exists with

the size of the UNIT (the number of sensors) and the response time. We synthetically increased the number of sensors to over 500. We conducted a test for UNIT and MULT(s,t) over the entire spectrum of queries (Time Exact & Sensor Range to Time Range & Sensor Exact). Figure 14 shows that a tradeoff exists for the number of sensors and the response time.

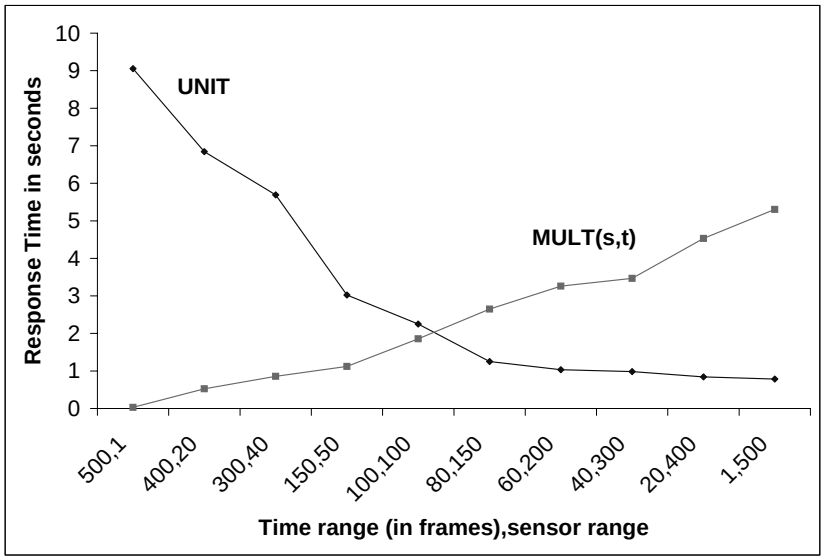

Figure 14: Impact of varying time range/sensor range on response time for UNIT and MULT(s,t)

The results show the following. On one hand, UNIT is superior for those queries that are selective on time, i.e., examine sensor values during a few time instances. On the other hand, MULT is superior for queries that are selective on sensors, i.e., examine a few sensor values across time. From Figure 14, depending on the number of sensors and time range records, a system designer can choose between UNIT and MULT. We intend to develop an analytical model that consumes the queries that constitute the workload of a system to choose between UNIT and MULT.

5 Related work

Majority of research on spatio-temporal data modeling has focused on extensions of either the spatial or temporal models to incorporate the other dimension. Most modeling approaches adopt a trajectory view of the moving objects over space-time axis [11]. Guting et. al [15] propose representations for the temporal development of spatial entities of certain data types such as moving points and moving regions. A few models using time frames have been proposed in [11] but due to the large variation in spatial data over time for moving objects, they have been ineffective in modeling the data. The focus of our paper is on a class of applications that consist of a fixed number of sensors time, making UNIT a viable representation scheme.

In addition to models for conceptualizing spatio-temporal data in real world applications, new

indexing methods are also required to expedite query evaluation on these data sets. The field of spatio-temporal indexing has been largely unexplored. Theodoridis et. al [10] provide specifications for efficient indexing in spatio-temporal databases. The type of indexing method depends on the data model used to conceptualize the spatio-temporal data. Tayeb et. al [36] use PMR-Quadtrees [28] for indexing the future linear trajectories of one dimensional moving point objects as line segments in (x,t) space. Kollios et al. [18] propose a dual transformation approach where a line for a point moving in time is transformed to a point, enabling the use of regular spatial indices. Details on how these approaches can be extended to two or more dimensions is not provided.

R-tree [16] and its variants can be applied to trajectory based models. Jensen et. al [27] use an R^*-tree [3] based indexing method called TPR-tree to index moving objects. TPR-tree outperforms the R-tree in both search performance and I/O. The GR tree [5], an R-tree based index for now-relative bitemporal data, is capable of accommodating efficiently two dimensional regions that grow. The idea is to introduce bounding regions that expand with the growing data regions. An evaluation of three representations of the R-tree, 3-D R-tree [37], $2+3$ R-tree and HR-tree [20] is provided in [21]. Tzouramanis et. al [39] use the overlapping linear quadtree approach based on the concept of overlapping trees [19].

Other related research from moving objects include topics such as uncertainty in positions of moving objects. Wolfson et. al [41] discuss this imprecision problem and how DBMS can provide a bound on the error. Pfoser and Jensen [25] point out that given a speed limit, the possible locations between two sampling positions must be an ellipse instead of a straight line segment. Basch et. al [2] proposes several main memory data structures for mobile objects.

6 Conclusions and Future Research Directions

This paper reports on our preliminary investigation of moving sensors. Our primary contribution is a framework that builds upon two fundamental concepts: physical data independence and software re-use. We investigated trajectory-based and frame-based approaches as two alternatives for representing sensory data. Our performance results demonstrates the following tradeoff: a) the UNIT approach is superior when queries can be evaluated by examining several sensors in a single time instance (frame) or short intervals, and b) the MULT approach is superior when queries can be evaluated by examining a few sensors across several time instances (frames). For a mixed workload, the best approach may be to store data redundantly rendering both representations.

We intend to extend this work in several directions. First, we plan to design the appropriate physical layer for each of the UNIT and MULT representations. Although in our experimental setups we implemented the two representations on top of an object-relational database, our ultimate goal is to build them directly into the pages of an underlying file system. For example, each frame of the UNIT representation can be considered as a disk block. We can benefit from our background in the design of continuous media servers [13] in order to place and schedule these frame blocks to (say) play back a recorded haptic experience. These pages or disk blocks can then be incorporated directly as nodes of a spatial or temporal index structure (i.e., clustered indexing). Such clustering of sensors would reduce the number of disk I/Os and hence improve performance.

Second, we want to demonstrate that pattern recognition techniques can benefit from the structure imposed by our hierarchy of spatio-temporal data representation. By applying pattern recognition techniques between each two layers, rather than attempting to recognize high level information

directly from the sensor data, we hope to improve the speed and accuracy of pattern recognition. In addition, we speculate that our representation hierarchy will allow for a more collaborative approach. Human designers can specify the transformation rules that they are able to hand-code, and then an automated algorithm can be used to find the others. Beyond improving performance, this approach should make it easier for humans to understand the results.

Finally, following our second future plan, as we bridge the gap between users' semantic queries and the raw sensory data, the framework must consider user preferences. This is because the queries in these environments are very subjective to human perceptions. We plan to build on our previous work on using fuzzy-logic to incorporate human perception in querying image databases [31].

References

[1] E. Andre, G. Herzog, and T. Rist. Generating Multimedia Presentations for Robocup Soccer Games. Technical Report DFKIGmbH D-66123, German Research Center for Artificial Intelligence, Saarbrücken, Germany, 1998.

[2] J. Basch, L. Guibas, and J. Hershberger. Data Structures for Mobile Data. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[3] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1990.

[4] M. C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1995.

[5] R. Bliujute, C.S. Jensen, S. Saletnis, and G. Slivinskas. R-tree Based Indexing of Now-Relative Bitemporal Data. In *Proceedings of the 24th VLDB Conference*, 1998.

[6] Claramunt C. and Theriault M. Managing time in GIS: An event-oriented approach. *Recent Advances in Temporal Databases*, pages 23–42, 1995.

[7] J. Eisenstein, S. Ghandeharizadeh, L. Huang, C. Shahabi, G. Shanbhag, and R. Zimmermamnn. Analysis of Clustering Techniques to Detect Hand Signs. In *Proceedings of the International Symposium on Intelligent Multimedia, Video and Speech Processing*, 2001.

[8] Martin Erwig, Ralf Hartmut Gting, Markus Schneider, and Michalis Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInformatica*, 3:269–296, 1999.

[9] I. A. Essa, S. Basu, T. Darrell, and A. Pentland. Modeling, Tracking and Interactive Animation of Faces and Heads using Input from Video. In *IEEE Proceedings of Computer Animation*, 1996.

[10] Y. Theodoridis et. al. Specifications for Efficient Indexing in Spatiotemporal Databases. In *Proceedings of the 10th IEEE International Conference on Scientific and Statistical Database Management*, 1998.

[11] L. Forlizzi, R. H. Guting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Object Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2000.

[12] Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. A data model and data structures for moving objects databases. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, 29(2):319–330, 2000.

[13] S. Ghandeharizadeh and C. Shahabi. Distributed Multimedia Systems. In John G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*. John Wiley and Sons Ltd., 1999.

[14] Ralf Hartmut Gting, Michael H. Bhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and quering moving objects. *ACM Transactions on Database Systems*, 25(1):1–42, 2000.

[15] R.H. Guting, M.H. Bohlen, M. Erwig, and C.S. Jensen. A Foundation for Representing and Querying Moving Objects. Technical Report 238, Fern Universitat Hagen, 1999.

[16] A. Guttman. R-trees: A Dynamic Index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1984.

[17] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Oswaw. RoboCup: The Robot World Cup Initiative. In *Proceedings of the First International Conference on Autonomous Agents*. The ACM Press, 1997.

[18] G. Kollios, D. Gunopulos, and V.J. TSotras. On Indexing Mobile Objects. In *Proceedings of the PODS Conference*, 1999.

[19] Y. Manolopoulos and G. Kapetanakis. Overlapping B+ Trees for Temporal Data. In *Proceedings of the 6th JCIT Conference*, 1990.

[20] M.A. Nascimento and J.R.O. Silva. Towards Historical R-trees. In *Proceedings of ACM Symposium on Applied Computing (ACM-SAC)*, 1998.

[21] M.A. Nascimento, J.R.O. Silva, and Y. Theodoridis. Access Structures for Moving Points. TimeCenter Technical Report TR-33, TimeCenter, 1998.

[22] U. Neumann, J. Li, J.Y. Enciso, D. Fidaleo, and T.Y. Kim. Constructing A Realistic Head Animation Mesh For A Specific Person. Technical Report USC-CS-TR99-691, University of Southern California, 1999. URL http://www.cs.usc.edu/tech-reports/technical_reports.html.

[23] C. Padden. *American Sign Language*, volume 3. McGraw-Hill, 1987.

[24] Donna Peuquet and Niu Duan. An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data. *International Journal of Geographical Information Systems*, pages 7–24, 1995.

[25] D. Pfoser and C.S. Jensen. Capturing the Uncertainty of Moving-Object Representations. In *Advances in Spatial Databases, 6th International Symposium, SSD'99, HongKong, China*, 1999.

[26] T. Raines, M. Tambe, and S. Marsella. Towards Automated Team Aanalysis: A Machine Learning Approach. In *Proceedings of Third International RoboCup Competitions Workshop*, 1999.

[27] S. Saletnis, Ch. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2000.

[28] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley,Reading,MA, 1990.

[29] C. Shahabi, G. Barish, B. Ellenberger, N. Jiang, M. R. Koladouzan, S.-R. Nam, and R. Zimmermann. Immersidata Management: Challenges in Management of Data Generated within an Immersive Environment. In *Proceedings of the Fifth International Workshop on Multimedia Information Systems (MIS'99)*, Indian Wells, California, October 21-23, 1999.

[30] C. Shahabi, G. Barish, M. R. Kolahdouzan, D. Yao, R. Zimmermann, K. Fu, and L. Zhang. Alternative Techniques for the Efficient Acquisition of Haptic Data. In *To Appear in the Proceedings of ACM SIGMETRICS 2001*, June 2001.

[31] C. Shahabi, Y. Chen, and A. Nam. Soft Queries in Image Retrieval Systems. In *SPIE Internet Imaging (EI14), Electronic Imaging 2000, Science and Technology*, 2000.

[32] C. Shahabi, L. Kaghazian, S. Mehta, A. Ghoting, G. Shanbhag, and M. McLaughlin. Analysis of Haptic Data for Sign Language Recognition. In *To Appear in the International Conference on Universal Access in Human-Computer Interaction*, August 2001.

[33] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and querying moving objects. *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 422–432, 1997.

[34] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Querying the uncertain position of moving objects. *Temporal Databases: Research and Practice. (the book grow out of a Dagstuhl Seminar, June 23-27, 1997)*, 1399:310–337, 1998.

[35] T. Starner and A. Pentland. Visual Recognition of American Sign Language using Hidden Markov Models. In *Proceedings of the International Workshop on Automatic Face and Gesture Recognition*, 1995.

[36] J. Tayeb, O. Ulusoy, and O. Wolfson. A Quadtree Based Dynamic Attribute Indexing Method. *The Computer Journal*, 41(3):185–200, 1998.

[37] Y. Theodoridis, M. Vazirgiannis, and T. Sellis. Spatiotemporal Indexing for Large Multimedia Applications. In *Proceedings of 3rd IEEE Conference on Multimedia Computing and Systems*, 1998.

[38] Nectaria Tryfona and Thanasis Hadzilacos. Logical data modelling of spatio temporal applications: Definitions and a model. *IDEAS, International Database Engineering and Applications Symposium*, pages 14–23, 1998.

[39] T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos. Overlapping Linear Quadtrees: a Spatiotemporal Access Method. In *Proceedings of the 6th ACM International Workshop on GIS*, 1998.

[40] C. Vogler and D. Metaxas. Adapting Hidden Markov Models for ASL Recognition by Using Three Dimensional Computer Vision Methods. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 1997.

[41] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendex. Cost and Imprecision in Modeling the Position of Moving Objects. In *Proceedings of the ICDE*, 1998.

[42] Ouri Wolfson, A. Prasad Sistla, Bo Xu, Jutai Zhou, and Sam Chamberlain. Domino: Databases for moving objects tracking. *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 547–549, 1999.

[43] Ouri Wolfson, Bo Xu, Sam Chamberlain, and Liqin Jiang. Moving objects databases: Issues and solutions. *10th International Conference on Scientific and Statistical Database Management, Proceedings, Capri, Italy, July 1-3, 1998*, pages 111–122, 1998.