Deducing Behaviors from Primitive Movement Attributes

Danzhou Liu* and Charles E. Hughes**
School of Computer Science, University of Central Florida, Orlando, FL, USA 32816-2362

ABSTRACT

The research reported here anticipates the future of smart buildings by developing algorithms that categorize the movements of individuals based on such characteristics as motion vectors, velocity vectors, head orientation vectors and predetermined positions. The intended applications include detecting intrusions, helping lost visitors, and changing the artwork on virtual posters to reflect an individual's presumed interests. The vectors we capture represent trajectories in a multi-dimensional space. To make sense out of these, we first segment a trajectory into sub-trajectories, typically based on time. To describe each sub-trajectory, we use primitive patterns of body movement and additional information, e.g., average speed during this interval, head movement and place or object nearby. That is, for each sub-trajectory, we use a tuple of the following form: (interval_ID, body_movement, avg_speed, head_movement, places_passed). Since trajectories may have many outliers introduced by sensor failures or uneven human movement, we have developed a neural network-based pattern extraction subsystem that can handle intervals with noisy data. The choice of these attributes and our current classification of behaviors do not imply that these are the only or best ways to categorize behaviors. However, we do not see that as the focus of the research reported here. Rather, our goal is to show that the use of primitive attributes (low level), neural networks to identify categories of recognizable simple behaviors (middle level) and a regular expression-based means of describing intent (high level) is sufficient to provide a means to convert observable low-level attributes into the recognition of potential intents.

Keywords: Trajectory matching, pattern recognition, artificial neural networks, intrusion detection, pervasive computing.

1. INTRODUCTION

Pervasive Computing (Ubiquitous Computing) refers to a computing environment that ubiquitously provides services to users and that makes the actual computing devices and underlying networks transparent to users. Pervasive computing is a trend that is receiving a great deal of attention in the research community, industry, government and the popular press. For example, the Massachusetts Institute of Technology (MIT) is working on a project called Oxygen [20]. This name reflects a vision of pervasive computing that will be freely available everywhere, like oxygen in the air we breathe. Carnegie Mellon University's Human Computer Interaction Institute (HCII) is working on a similar project called Aura [10], in order to provide each user with an invisible halo of computing and information services that persists regardless of location. Moreover, IBM's project Planet Blue [14] is aimed to create and deploy a technology-assisted immersive environment used by knowledge workers in their daily lives, in which individuals and teams can create, learn, use and share knowledge with few limitations or disruptions, regardless of physical location or context.

With the help of pervasive computing, a smart home or office, controlling many devices within its environment, adapts itself to inhabitants and changing conditions, and communicates with inhabitants in a natural way such as speech and gesture. Several smart home projects have been conducted in the last decade. The Georgia Tech Aware Home [15] and the MIT Intelligent Room [23] employ many sensors to detect user locations and activities within an actual house. The Neural Network House at the University of Colorado [18] uses a neural network system to control basic residential comfort systems—air heating, lighting, ventilation, and water heating, with the goal of energy conservation. Moreover, The MavHome project [4, 6] at the University of Texas at Arlington seeks to maximize inhabitant comfort and minimize operation cost by predicting the mobility patterns and device usages of the inhabitants. With appropriate

* dzliu@cs.ucf.edu; www.cs.ucf.edu/~dzliu
** ceh@cs.ucf.edu; www.cs.ucf.edu/~ceh

sensors, smart homes will soon provide more extensive services, including analyzing the health of each inhabitant and providing security by recognizing the entry of unknown people.

In this paper, we propose algorithms that can be used to add useful services to smart homes and offices. By detecting trajectory patterns, one can recognize behaviors that aid in intrusion detection, help lost visitors, and may even invoke the changing of artwork on virtual posters to reflect an individual's presumed interests.

The movement of individuals can be monitored by sensor networks. Using these, we can obtain or calculate movement characteristics (low-level features) such as spatial positions, motion vectors, velocity vectors and head orientation vectors. Continuous movements of individuals form trajectories. By analyzing the movement characteristics of those trajectories, we can mine useful knowledge. For example, in the computer science building at the University of Central Florida (UCF), graduate student labs with bay windows are on the first floor, and professors' offices, the general office and the computer equipment room are on the second. If a person goes around on the first floor, stops in front of many windows and moves his head to face windows at 2:00PM, he is likely to be looking for some graduate student. If a person goes up to the second floor, and walks for a while and stands still near a professor's office, he is probably waiting for that particular professor. If he gazes intently at the papers on one part of the professor's bulletin board, he is showing interest in that area of research. If a person is in front of the computer equipment room at 3:00AM, and moves his head from time to time, he may be looking to steal equipment. This is even more probable if he moves in a determined fashion away from the room each time another person enters the hallway and then returns as soon as no one else is present.

In a literature review, we found that almost all existing algorithms for trajectory matching focus on low-level trajectory matching (i.e., shape matching), and therefore are not suitable for semantic-level trajectory pattern matching, which are more meaningful in the applications we are considering. We also found that there has been little research on deriving high-level representations (e.g. going straight, standing still, turning left, turning right, circling, and zigzagging) from low-level trajectory features (e.g., positions, and velocity vectors), and using these high-level representations for discovering similar trajectories and detecting various typical trajectory patterns.

In this paper, we address the problem of detecting trajectory patterns in pervasive computing environments. The main contributions of this paper can be summarized as follows. First we develop a neural-network based feature extraction system to derive high-level representations from low-level trajectory features. Second, we propose a way to express patterns using regular expressions. Lastly, we present a pattern detection algorithm using these high-level representations and pattern expressions.

The remainder of this paper is organized as follows. Section 2 reviews related work. In Section 3, we present our approach to detect trajectory patterns. Section 4 describes our performance experiments and presents current results. Finally we conclude the paper and identify directions for future research in Section 5.

2. RELATED WORK

With the rapid growth of GPS techniques and mobile computing, collecting large volume of spatiotemporal data is now possible. In [7], several trajectory representations are discussed. The trajectory of a moving object is usually represented as a sequence of successive locations in a 2D or 3D space. Discovering similar trajectories of moving objects can be very useful for moving objects' pattern matching and pattern prediction. For example, by tracking animals' trajectories, we can determine migration patterns of certain groups of animals. In a traffic management system, we can mine trajectories of various vehicles to figure out typical traffic patterns. Approaches to measure the similarity between trajectories include the use of the Euclidean distance, the Dynamic Time Warping (DTW) distance [26], the Longest Common Subsequence (LCSS) [25], and the Normalized Edit Distance (NED) [8]. The Euclidean distance and DTW distance are very sensitive to noise introduced due to anomalies in the sensors. To cope with noise, the LCSS, a variation of the edit distance, matches two sequences by allowing them to stretch, without rearranging the sequence of the elements, but allowing some elements to be unmatched. Meanwhile, the NED, another variation of the edit distance, converts the trajectories into a symbolic representation (movement pattern strings). Movement pattern strings encode both the movement directions and the movement distance information of the trajectories, and then the distances that are computed in a symbolic space are lower bounds of the distances of the original trajectory data. Although all four major

approaches are able to remove the shifting and scaling effects after some normalization and/or transformation, they cannot remove the rotation effect. For example, if one trajectory is a straight line and the other is generated by rotating the previous one, the distance between the two trajectories may be very large and then the two trajectories are treated as not similar. Discovering similar trajectories unaffected by rotation is discussed in [5, 16] using low-level features. Specifically, a trajectory is represented as the path and speed curves in [16], and this representation separates positional information from temporal information. The path curve records the position of the tracked object while the speed curve records the magnitude of its velocity. From these, the curve features, invariant to scale, translation and rotation, are derived. Furthermore, in [3], a trajectory is partitioned into sub-trajectories, and each sub-trajectory is then modeled by a set of spatial and temporal translation invariant attributes for motion trajectory matching. Those low-level trajectory matching algorithms, however, are not suitable for semantic-level trajectory pattern matching. In this paper, we will address trajectory pattern matching using high-level features.

Mining sequential patterns, an important problem in data mining, refers to discovering all sequential patterns with a user specified minimum support, where the support of a pattern is the number of data-sequences that contain the pattern [1]. An example of a sequential pattern is "20% of customers bought a laptop in one transaction, followed by purchasing a portable USB Flash Drive in a later transaction." In [22], the problem of mining sequential patterns is generalized to consider 1) time constraints that specify a minimum and/or maximum time period between adjacent items in a pattern, 2) time windows that allow the items of a sequential pattern to span a set of transactions within a user-specified time window, and 3) item taxonomies that allow sequential patterns to include items across all levels of a user-defined taxonomy (is-a hierarchy). In order to reduce the computational cost and overwhelming volume of potentially useless sequential patterns, some user-specified constraints in the form of episodes and regular expressions, and how to push such constraints inside the pattern mining process (i.e., prune the search space of sequential patterns during computation), are also studied in [17]. The "Apriori property" employed in association rule mining can be applied to mining sequential patterns because, if a sequential pattern of length k is infrequent, its superset (of length k+1) cannot be frequent. Therefore, most of the methods for mining sequential patterns adopt variations of Apriori-like algorithms, although they may consider different constraints. Another approach for mining sequential patterns is to use the frequent-pattern growth (FP-growth) technique without candidate generation [11]. In this paper, we will use one of the above algorithms plus our intuition to derive some common sequence for a specific trajectory pattern.

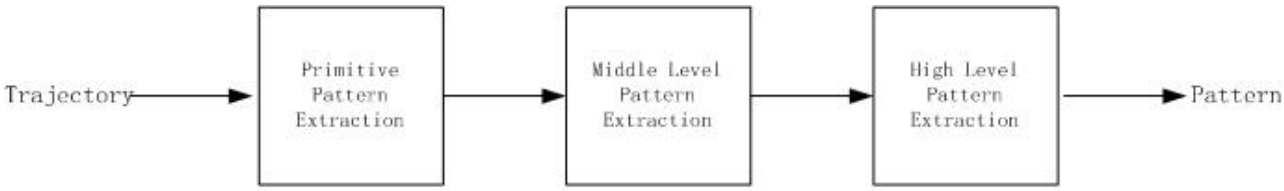

Figure 1. Pattern Extraction Architecture

3. TRAJECTORY PATTERN DETECTION

In this section, we outline our trajectory pattern detection system. It consists of three parts: primitive pattern extraction, typical pattern expression, and pattern detection. Generally, pattern extraction can have three layers as shown in Figure 1. In this paper, we focus on human trajectories in a smart building. We assume that the movement of individuals in a smart building is monitored by some sensors, and that movement characteristics such as spatial positions, motion vectors, velocity vectors and head orientation vectors can be obtained or derived. We also assume the trajectory is represented as a sequence of successive locations in a 2D space. With some modifications, our system should be able to handle other moving objects' trajectory patterns in a 3D space.

3.1 Neural Network-based Primitive Pattern Extraction

Trajectories may have many outliers introduced by sensor failures or by the unsteady motion exhibited in normal human movement. In this paper, a neural network-based primitive pattern extraction subsystem is proposed, which can be trained for various primitive patterns and can tackle trajectories with noisy outliers.

We first segment a trajectory into sub-trajectories based on time interval (e.g. 10 seconds), and each sub-trajectory $S = \{p_1, p_2, ..., p_k\}$ where $p_i \in S$ is the 2D location during the i^{th} interval. To describe each sub-trajectory (i.e., each interval), we use some primitive patterns of body movement and additional information (e.g., average speed during this interval, head movement and place or object passing by). Specifically, we first define six primitive patterns for human body movement: G(go straight), L(turn left), R(turn right), S(stop), C(circle), Z(zigzag), as shown in Figure 2. For each primitive pattern of human body movement, we can use speed features to describe it: H(high speed), M(medium speed), and S(slow speed). As a human moves, his head also moves and 5 primitive patterns for head movement are as follows: F(look forward), L(look left), R(look right), U(look up), D(look down). Moreover, for places or objects passing by a human, several primitives are as follows: P(professor office), S(student lab), C(computer equipment room), O(poster or other object of art), W(wall), T(treasure or cash office). That is, for each sub-trajectory, we can use a tuple to describe it: (*interval_ID, body_movement, avg_speed, head_movement, places_passing*).

For example, the tuple (1, G, M, F, P) represents: during time interval 1, a person went straight with medium speed and looking forward head pose, passing by a professor's office. From this we can create a relational database for a trajectory with those tuples, and create indices, and use SQL to evaluate queries.

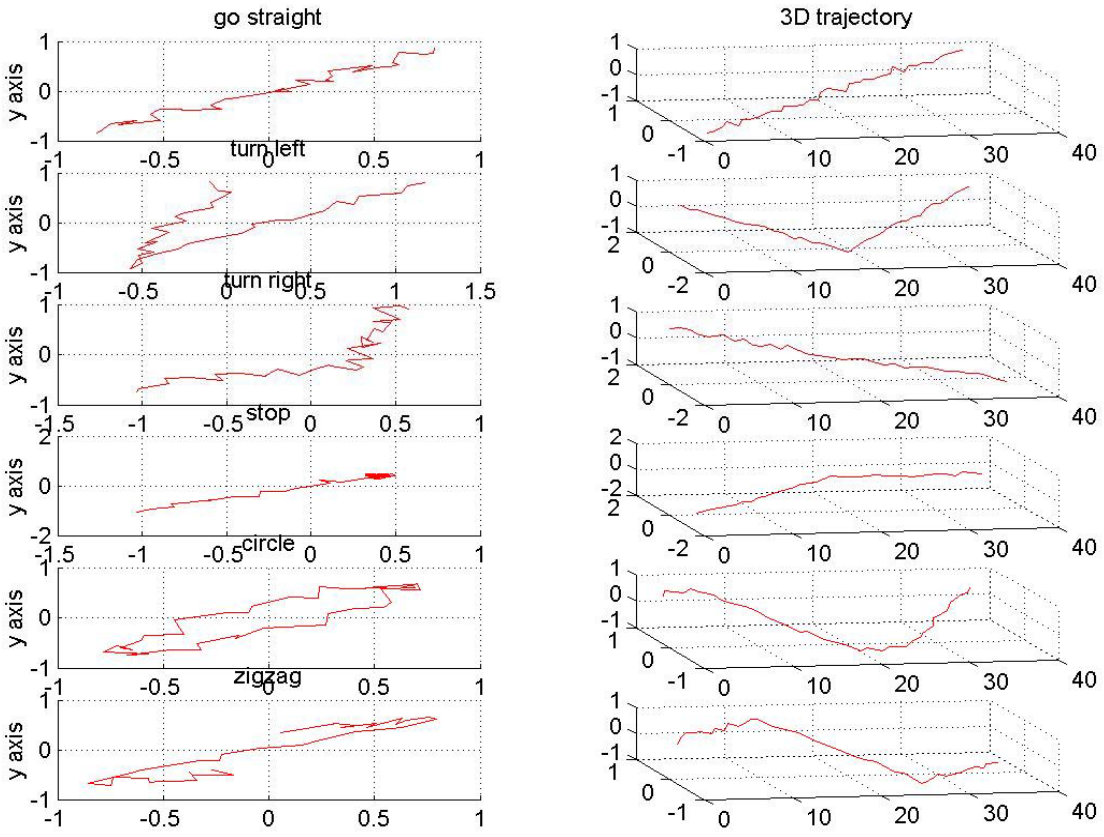

Figure 2. Primitive Patterns of Body Movement

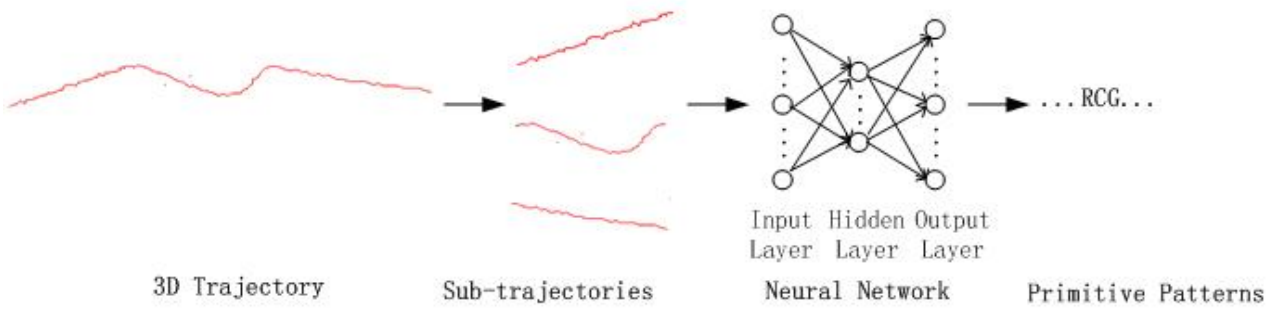

Figure 3. Neural Network-based Primitive Pattern Extraction

This then gives rise to the central issue: how to extract primitive patterns and other relevant information. For body movement, we can employ a neural network approach as explained later. For average speed, we can calculate the length of the sub-trajectory, divide the length by the time interval, and then compare the result with user-specified speed thresholds. For head movement, we can use the head orientation vectors. For places or objects passing by, we can scan through all objects with spatial positions and find the nearest one during the interval if the number of objects is small, or employ some nearest neighbor query algorithms [13, 21, 24], with spatial indices if the number of objects is large.

For body movement, we use a multilayer feed-forward neural network with back propagation for classification as shown in Figure 3. A trajectory is first segmented into sub-trajectories, and each sub-trajectory S is normalized as follows:

$$p' = \frac{p - mean(S)}{std(S)}, \forall p \in S.$$

That is, each point subtracts the mean of S, and then is divided by the standard deviation of S [25]. After normalization, each sub-trajectory is sequentially fed into the input layer, and the input layer size is determined by the number of points in S. The output layer size is 6 because there are 6 predefined primitive patterns of body movement. As shown in Figure 3, the trajectory is segmented into 3 sub-trajectories, and 3 corresponding primitive patterns are G, C, and R respectively.

3.2 Pattern Expression using regular expression

After defining some primitive trajectory patterns represented as ASCII characters, we can use them to define more complex trajectory patterns. As stated in [2, 9, 12, 19], the *regular expression* (RE) notation is valuable for describing searches for interesting sequential patterns, possesses sufficient expressive power for specifying a wide range of interesting, nontrivial patterns, and has exactly the same expressive power as *deterministic finite automata*. By using regular-expression notation, we can describe a pattern at a high level. In this paper, we adopt the Unix-style notation.

To make this notation clear and unambiguous, we first recall some basic rules for UNIX-style regular expressions:

- . matches any character.
- [set] matches one of the characters in the set. A shorthand C-Z is used to specify a set of characters from C to Z.
- The operator | is to denote union.
- The operator * means "zero or one of."
- The operator + means "one or more of."

Using regular expression and primitive trajectory patterns, we can define a trajectory pattern P as (*body_movement_RE, avg_speed_RE, head_movement_RE, places_RE*).

For example, we can define the pattern P_1 to find a professor in the computer science department building as:
(G+.*S+, Φ, Φ, .*P+)
where Φ indicates the empty regular expression. P_1 says that a tracked person walks for a while and stands still near a professor's office.

Another pattern P_2, to potentially steal items from the computer equipment room, might be defined as :
(G+.*S+, Φ, .*[LRUD]+.*, .*C+).
P_2 says that a tracked person walks for a while, his head moves from time to time, and finally he stops by the computer equipment room.

Moreover, we define an event to consist of a set of patterns. Using operations for regular sets, we define seven operators over events. Of these the first six do not go beyond normal operations under which regular sets are closed [12]; the seventh (concurrency) is an extension and is a major area for our continuing research.

- The union of two events E_1 and E_2, denoted as $E_1 \cup E_2$, is the set of patterns that are in either E_1 or E_2, or both.
- The intersection of two events E_1 and E_2, denoted as $E_1 \cap E_2$, is the set of patterns that are in both E_1 and E_2.
- The exclusive union of two events E_1 and E_2, denoted as $E_1 \oplus E_2$, is the set of patterns that are in either E_1 or E_2, or both.
- The concatenation of events E_1 and E_2, denoted as $E_1 E_2$, is the set of patterns that can be formed by taking any patterns in E_1 and concatenating it with any patterns in E_2.
- The closure of an event E is denoted E^* and represents the set of those patterns that can be formed by taking any number of patterns from E, possibly with repetitions and concatenating all of them.
- The complement of event E, denoted as $\sim E$, is the set of patterns that are not in E.
- The concurrency of two events is expressed as $E_1 \parallel E_2$.

For example, we can first define some higher level expressions as follows:
Body Movement Higher Level Expressions:
Idle = { } empty set
Moving = (G + L + R + C + Z)*

Speed Higher Level Expressions:
Relaxed = (M + S)+
Hectic = ((M + S)* H (M + S)*)+

Now combining these, we can then get more complex patterns (i.e., events):
EasyWalk = Moving || Relaxed
AgitatedWalk = Moving || Hectic

As shown above, we can use primitive trajectories to describe a wide range of interesting complex patterns. Furthermore, we can use operators over those complex patterns, and compose more complex patterns (i.e., events).

3.3 Concurrency
The representation we are presenting in this paper requires us to use the concurrency operator to express the desire to temporally link two or more of the four dimensions we are recording. Without the concurrency operator, our current scheme treats each dimension independently. We could have used an alternate representation that retains the data as points in a four-dimensional space. In such a four dimensional representation, we could use an expression like
(< G , . , . , .>+ < . , . , . , . >* < S , . , . , P >+)
for the professor seeking pattern. This retains the concurrency of stopping in front of the professor's office, whereas
(G+.*S+, Φ, Φ, .*P+)
could be matched even when the stopping has no overlap with the position at the professor's office. Of course, the concurrency operator could account for this, but that introduces complexity in the matching algorithm.

There is, however, a problem with this new notation when it comes to the thievery case. There, we previously used
(G+.*S+, Φ, .*[LRUD]+.*, .*C+)
This nicely captures the fact that the scanning action of the head might occur anywhere in the process of going to the computer room, even at the computer room. In the alternate notation, we might use
(< G , . , . , .>+ < . , . , . , . >* < S , . , . , P >+) ∩ (Φ , Φ , .*[LRUD]+.* , Φ)
But the inclusion of intersection adds complexity.

Clearly, there are pluses and minuses of each representation. Further experimentation is required to determine which is more effective under what circumstances. However, even if we determine that the four-dimensional representation is better, we will still retain the concurrency operator, but its use will be limited to representing concurrency of events associated with separate objects (typically people) whose movement is being tracked. This gives us the opportunity to represent actions that seem to be influenced by the presence or absence of others who might observe a person's behavior.

3.4 Pattern Detection

In this section, we will present the algorithm for pattern detection as shown in Figure 4. Here we assume that the trajectory information is extracted into 4 strings: body_movement_string, avg_speed_string, head_movement_string, and place_passing_string, and the interested patterns using regular expression are specified by the user. Our goal is to detect several candidate patterns based on these 4 strings. No attempt is made in this algorithm to deal with complex events.

DETECTPATTERN (*PE, BS, AS, HS, PS*)

Input:

pattern expressions *PE*
body_movement_string *BS*
avg_speed_string *AS*
head_movement_string *HS*
place_passing_string *PS*

Output:

A set of candidate patterns *P*

```
01 P ← Ø
02 for each pattern expressions pe ∈ PE  do
03      pattern_exist ← FALSE;
04      generate regular expressions of pe: be for BS, ae for AS, he for HS, and ps for PS.
05      for each occurrence (i,j) of be  in BS do
06          if ae, he, ps  occur in AS(i,j), HS(i,j), and PS(i,j) respectively
07              add (i,j) into the interval list ilist
08              pattern_exist ← TURE;
09          endif
10      enddo
11      if pattern_exist is TURE
12          calculate the maximum number count, denoted as maxcount, of intervals without overlapping in ilist
13          p.name ← GETNAME(pe);
14          p.maxcount ← maxcount;
15          add p into the priority queue P
16      endif
17 enddo
18 return P
```

Figure 4. Pattern Detection Algorithm

As shown in Figure 4, the set of candidate patterns P is initialized to be empty. In the *for*-loop (from lines 2 to 17), each interested pattern is examined to determine whether the pattern occurs or not, and if the pattern occurs, we also calculate the frequency of the pattern. The higher the frequency of the pattern, the more likely the traced person acts in this way. Specifically, at line 4, for each pattern *pe*, we decompose it based on body movement, average speed, head movement, and places passing by. From lines 5 to 10, we examine whether the pattern occurs. From lines 11 to 16, if the pattern exits, we put the pattern name with the pattern's frequency into the priority queue P. P orders patterns from highest to lowest frequency. The first pattern in P is regarded as the most likely pattern.

4. EXPERIMENTS

In our experiments, we use Unreal II, a game engine, to generate trajectories. Given a pattern or a task, several subjects are asked to use the system similar to playing games as shown in Figure 5. For this trial, we assume that we are only interested in two tasks: finding a professor, and stealing items from the computer equipment room. That is:
P_1 = (G+.*S+, Φ, Φ, .*P+)
P_2 = (G+.*S+, Φ, .*[LRUD]+.*, .*C+).

For each pattern detection test, students can do either P_1 or P_2 or any other tasks. The output of our system is either P_1 or P_2 or unknown pattern P_3. As students go around the building, navigating with a keyboard and mouse, our system records the shooter's position at each time interval. The serial positions are used to generate an average speed string and body movement strings by employing a neural network. Furthermore, we treat the pistol movement as the head movement, recording the direction of its movement, and using this to generate a head movement string. Meanwhile, we predefine some hot spots. When the shooter passes by those hot spots, they are recorded to generate the place passing string. When a subject completes the task, our system employs the pattern detection algorithm in Figure 4 to guess the subject's possible patterns. In addition, for each primitive pattern, we generate 5,000 neural network training data sets with various noise ratios. Thus, 30,000 data sets are used to train the neural network.

Figure 5. Experimental Environment

Since our goal here is studying preliminary algorithms, not doing a full-fledged validation on a large variation of behaviors, we invited a small number of subjects, 4 students in our department, to assist in the test. Each student was asked to run our system 30 times (i.e., testing each pattern 10 times). The experimental results are shown in Table 1. The detection rate for P_1 is more then 90%, the one for P_2 is slightly better than 80%, and the one for P_3 is less than 70%. The fail cases for P_1 include standing still in front of the professor room for a very short time. Unfortunately, the granularity of our system was not fine enough to observe behaviors of short duration. The fail cases for P_2 include going

straight towards the computer equipment root without any head movement. Occasionally, unknown patterns P_3 are detected wrongly as P_1 or P_2. In order to reduce the detection error, we can improve the sampling rate with the attendant cost of data overhead, and find the more accurate pattern expression with sequential pattern mining algorithms using large amounts of real data.

TABLE I
EXPERIMENTAL RESULTS

Pattern	Detection Rate
P_1	92.5%
P_2	82.5%
P_3	67.5%

5. CONCLUSIONS

We have identified that almost all existing algorithms for trajectory matching are not suitable for semantic-level trajectory pattern matching. We have therefore proposed a novel pattern detection algorithm. Experimental results show that this algorithm can achieve good detection rate (i.e., more than 80%) for predefined patterns.

Some interesting topics for future research are as follows:

- Creating a real environment in a building to collect huge amount of real data, and using these real data to help derive more accurate pattern expression for each interested pattern.
- Experimenting with the efficacy of multi-dimensional representations (those that retain concurrency information) in comparison to those describing each dimension independently.
- Extending the algorithm presented here to handle event composition, especially as regards concurrency of patterns associated with multiple tracked objects.
- Using evolutionary computing techniques to evolve a population of strings representing each behavior of interest. An issue here is how we would compute the fitness function, but we believe that can be resolved with a large volunteer group recruited on the Web. Interestingly, those few who would try to subvert the quality of the fitness function may actually help us by generating mutations.

ACKNOWLEDGMENTS

This work is supported in part by Research in Augmented and Virtual Environments (RAVES), a project funded by the Naval Research Laboratory (NRL) VR LAB. We would like to thank Yun Fan, who helped develop an experimental environment using Unreal II to generate trajectories.

REFERENCES

1. R. Agrawal and R. Srikant, "Mining sequential patterns," *Proceedings of the 11th International Conference Data Engineering (ICDE 1995)*, Taipei, Taiwan, 3-14, March 1995.
2. C. L. A. Clarke and G. V. Cormack, "On the use of regular expressions for searching text," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, **19(3)**:413-426, 1997.
3. W. Chen and S. F. Chang, "Motion trajectory matching of video objects," *Proceedings of SPIE Electronic Imaging*, January 2000.
4. D. J. Cook, M. Youngblood, E. O. Heierman III, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja, "MavHome: an agent-based smart home," *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, 521-524, March 2003.
5. S. Dagtas, W. Al-Khatib, A. Ghafoor, and R. Kashyap, "Models for Motion-based Video Indexing and Retrieval," *IEEE Transactions on Image Processing*, **9(1)**:80-101, January 2000.

6. S. Das, D. Cook, A. Bhattacharaya, E. Heierman, and T. Lin, "The Role of Prediction Algorithms in the MavHome Smart Home Architecture," *IEEE Wireless Communications*, **9(6)**:77-84, December 2002.
7. N. Dimitrova and F. Golshani, "Motion Recovery for Video Content Classification," *ACM Transactions on Information Systems*, **13(4)**:408-439, October 1995.
8. L. Chen, M. T. Özsu, and V. Oria, *Symbolic Representation and Retrieval of Moving Object Trajectories*, CS Technical Report CS-2003-30, University of Waterloo, September 2003.
9. M. Garofalakis, R. Rastogi, and K. Shim, "Mining Sequential Patterns with Regular Expression Constraints," *IEEE Transactions on Knowledge and Data Engineering*, **14(3)**:530-552, May/June 2002.
10. D. Garlan, D.P. Siewiorek, A. Smailagic, and P. Steenkiste, "Project Aura: toward distraction-free pervasive computing," *IEEE Pervasive Computing*, **1(2)**:22 - 31, April-June 2002.
11. J.W. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001.
12. J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 2001.
13. G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Transactions on Database Systems*, **24(2)**:265-318, June 1999.
14. IBM Planet Blue Project. URL: http://www.research.ibm.com/compsci/planetblue.html.
15. V. Lesser, M. Atighetchi, B. Benyo, B. Horling, A. Raja, R. Vincent, T. Wagner, X. Ping, and S. X. Zhang, "The intelligent home testbed," *Proceedings of the Autonomy Control Software Workshop*, 1999.
16. J. J. Little and Z. Gu, "Video retrieval by spatial and temporal structure of trajectories," *Proceedings of SPIE Storage and Retrieval for Media Databases*, San Jose, 2001.
17. H. Mannila, H. Toivonen, and A. Verkamo, "Discovering frequent episodes in sequences," *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD'95),* Montreal, Canada, 210-215, August 1995.
18. M. Mozer, "The neural network house: An environment that adapts to its inhabitants," *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, 110-114, 1998.
19. G. Navarro, "Pattern matching," *Journal of Applied Statistics*, **31(8)**:925-950, October 2004.
20. Nissanka B. Priyantha, Allen K. L. Miu, Hari Balakrishnan, and Seth Teller, "The cricket compass for context-aware mobile applications," *Proceedings of 7th ACM Conference on Mobile Computing and Networking (SIGMOBILE)*, Rome, Italy, July 2001.
21. N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, CA, USA, 71-79, May 1995.
22. R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," *Proceedings of the Fifth International Conference on Extending Database Technology (EDBT)*, 1996.
23. M. C. Torrance, "Advances in human-computer interaction: The intelligent room," *Working Notes of the CHI 95 Research Symposium*, 1995.
24. Y. Tao, D. Papadias, and Q. Shen, "Continuous Nearest Neighbor Search," *Proceedings of International Conference on Very Large Data Bases (VLDB)*, Hong Kong, 287-298, August 2002.
25. M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, 673-684, 2002.
26. B. K. Yi, H. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," *Proceedings of the 14th International Conference on Data Engineering (ICDE)*, 23-27, 1998.