Trajectory Clustering for Motion Prediction

Cynthia Sung, Dan Feldman, Daniela Rus

***Abstract*— We investigate a data-driven approach to robotic path planning and analyze its performance in the context of interception tasks. Trajectories of moving objects often contain repeated patterns of motion, and learning those patterns can yield interception paths that succeed more often. We therefore propose an original trajectory clustering algorithm for extracting motion patterns from trajectory data and demonstrate its effectiveness over the more common clustering approach of using k-means. We use the results to build a Hidden Markov Model of a target's motion and predict movement. Our simulations show that these predictions lead to more effective interception. The results of this work have potential applications in coordination of multi-robot systems, tracking and surveillance tasks, and dynamic obstacle avoidance.**

I. Introduction

In multi-robot systems, task execution often requires task hand-off. This could occur because of exigent circumstances, such as robots requiring maintenance or refueling, or because of the task itself, as when tracking a moving object would take a robot outside its acceptable range of motion. In all these cases, a plan for sending a replacement to take over the task is necessary. In this paper, we consider tracking and surveillance tasks, where a robot must plan to intercept another robot or moving vehicle for rendezvous.

Classical approaches to interception generally assume a worst-case scenario: no information about the future is known [1], [2]. Planning then requires short-term, flexible paths, often based on assumptions such as constant velocity or acceleration. This model, however, is restrictive in that it ignores information about motion tendencies that can usually be learned. In this paper, we propose a data-driven approach to predicting motion. We argue that moving objects often follow typical motion patterns and that historical data about their movement can be used to predict their future locations.

This idea has been suggested before. Vasquez et al. [3] used Growing Hidden Markov Models to analyze car behavior in a parking lot, identify common states, and predict trajectories by estimating a car's intention. Bennewitz et al. [4] similarly analyzed the motion paths of people in an office space. These approaches require discretization of the target's state space to fit a Hidden Markov Model (HMM). Our key contribution is to represent state as short trajectories rather than static positions. This higher level of abstraction provides greater flexibility to represent not only position, but also velocity and intention. Since we are interested in tracking or surveillance tasks, we also do not assume input trajectories to have clearly identifiable stop points.

This paper presents our solution to extracting motion patterns from historical data and using them to generate interception paths. Section II describes previous efforts in classifying trajectory data. Section III contains our main contribution, a novel trajectory clustering algorithm for identifying motion patterns. Section IV demonstrates how this clustering can be used to train a Hidden Markov Model and predict motion for an interception task. Simulations and discussion are given in sections V and VI.

II. Related Work in Trajectory Classification

Advances in GPS and tracking technology have motivated large efforts in classifying trajectories. The methods currently in the literature follow two general trends.

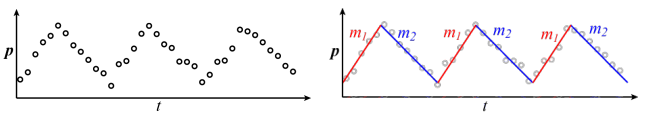
One branch of trajectory analysis aims to find a measure of similarity between trajectories.Unlike the distance between two points or between a point and a line, what the distance between two curves should be is unclear. The majority of metrics proposed, such as Euclidean distance [5] or dynamic time warping [6], are arbitrary, require tuning of multiple parameters, and fail to capture a human's intuition of similarity. In addition, while this line of work is appropriate for short trajectories, it cannot discover common subsequences in longer trajectories, which can be dissimilar on the whole, or accommodate data streams containing multiple instances of the same subsequence, as would be the case in a long-term tracking task.

The other branch of trajectory classification aims to generate a spatial map of typical motions in a trajectory. For example, Lee et al. [7] approximate trajectories by line segments and cluster these segments to find the object's typical motions. However, this approach also suffers from the difficulty of defining a similarity metric for line segments, is sensitive to input parameters, and does not translate well to higher than two dimensions. Additionally, since only the spatial information in the trajectories is analyzed and preserved, all original temporal information is lost.

The lack of an effective distance metric motivates us to find a new approach to trajectory clustering. In particular, intervals in one-dimensional space suggest an intuitive similarity measure: the amount of overlap between them [8]. We use interval comparison to perform a more intuitive and robust clustering. We also present a technique for projecting d-dimensional subtrajectories as one-dimensional intervals.

Support for this project has been provided in part by the Future Urban Mobility project of the Singapore-MIT Alliance for Research and Technology (SMART) Center, with funding from Singapore's National Research Science Foundation, by ONR MURI Grants, N00014-09-1-1051, and N00014-09-1-1031, and by NSF award IIS-1117178. We are grateful. We thank Oxford Mobile Robotics Group for sharing the Oxford data set.

C. Sung, D. Feldman, and D. Rus are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA emails: {crsung, dannyf, rus}@csail.mit.edu



(a) Trajectory (b) Clustered Output

Fig. 1. The goal of trajectory clustering is to identify common motion patterns in a trajectory. In this case, the trajectory (a) contains two motion patterns, m_1 and m_2, that alternate.

III. Learning Subtrajectory Patterns

In this section, we explain our approach to detecting common subsequences in a trajectory.

A. Definitions and Problem Statement

We define a *timestamped point* to be a pair $(t, \mathbf{p})$ consisting of a time $t > 0$ and a point $\mathbf{p} \in \mathbb{R}^d$. A *trajectory* is a sequence T of n timestamped points $\{(t_1, \mathbf{p}_1), (t_2, \mathbf{p}_2), ...(t_n, \mathbf{p}_n)\}$ ordered so $t_i < t_j \ \forall i < j$. A *subtrajectory* of T is a subsequence $S = iTj \subseteq T$ containing the elements $\{(t_i, \mathbf{p}_i), (t_{i+1}, \mathbf{p}_{i+1}), ...(t_j, \mathbf{p}_j)\}, 1 \leq i, j \leq n$.

We define a *motion pattern* of T to be a structure, consisting of a line segment and a duration, that represents a typical pattern found in T.

The *trajectory clustering problem* is as follows: Given a trajectory T, find the minimum set $\mathcal{R}$ of motion patterns such that T can be approximated by a sequence of elements of that set (see Fig. 1). Equivalently, we would like to partition T and cluster the resulting subtrajectories such that clusters, which we denote C_k, each contain only instances of a unique motion pattern. We call k the *cluster index* of subtrajectory S if $S \in C_k$. We call the motion pattern corresponding to C_k its *cluster representative*.

B. Subtrajectory Pattern Detection

Our algorithm consists of the following four steps, illustrated in Fig. 2:

1) Line simplification
2) k lines projection
3) Interval clustering
4) Calculation of representatives

We choose motion patterns to be line segments because they are the simplest structure that can contain both position and velocity information. In that case, the first step is to partition the trajectory into subsequences that can be well-approximated by line segments. Due to small variations in target motion, noise in the original data, or variable sampling rate, outputted segments will not be exact replicas of motion patterns, so we use clustering to detect segments that are approximately the same. Since it is not clear what distance metric is appropriate for line segments, our clustering is in two stages. We reduce the situation to 1-D by projecting the segments onto the trajectory's k-lines center and then cluster the resulting intervals on the k-lines. Finally, we can extract the motion patterns from the clusters. In the following sections, we expound on each of these steps.

Step 1: Line Simplification. The first step involves partitioning the trajectory at its critical points, where its behavior changes, so that each of the resulting independent subtrajectories $S_1 = 1Tc_1, S_2 = c_1Tc_2, ...S_{n_c} = c_{n_c}Tn$ can be approximated by a line segment. This process, commonly referred to as "line simplification," is a well-studied computational geometry problem for which multiple algorithms (see [9] and references therein) exist. To produce subtrajectories of approximately constant velocity, we perform line simplification with time as an added dimension, that is, on the sequence $\{[\mathbf{p}_i \ t_i]\} \in \mathbb{R}^{d+1}$. In our experiments, we implemented a variation of the Douglas-Peucker algorithm that minimizes regression distance.

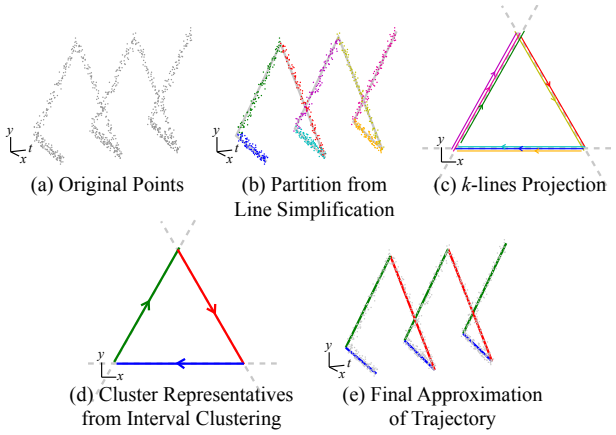


Fig. 2. Example run of the algorithm on a repeated triangular trajectory. The input trajectory is partitioned (Step 1) into subtrajectories (shown in different colors) that are then projected onto the trajectory's k-lines center (Step 2) for second-stage clustering (Step 3).

Input Parameter. For this step, a single input parameter ϵ_{LS} dictates the maximum allowable synchronized Euclidean distance between a point and the line simplification, and can be used to influence the resolution of the output.

Step 2: k-Lines Projection. The output of Step 1 is sets of consecutive points in time that are approximately linear. We translate the problem of determining similarity between these subtrajectories into 1-D by projecting them onto the k lines that best represent the trajectory's spatial data.

Let the error between a set of points S and a line ℓ be

$$\text{err}(S, \ell) = \frac{1}{|S|} \sum_{\mathbf{p} \in S} \text{dist}(\mathbf{p}, \ell)$$

where $\text{dist}(\mathbf{p}, \ell)$ is the Euclidean distance between point $\mathbf{p}$ and line ℓ. The problem of k-lines projection is: Given n sets of points in $\mathbb{R}^d$, find the set $\mathcal{L}$ of k lines ℓ that minimizes

$$\max_S \min_{\ell \in \mathcal{L}} [\text{err}(S, \ell)]$$

This problem is similar to k-lines center, which is NP-hard [10]. While multiple approximation algorithms for traditional k-lines center exist, to our knowledge, this specific variant has not been treated. We thus developed a heuristic for approximating the solution (see section III-C).

Input Parameter. Although the problem description is in terms of k, it is possible to specify instead the maximum per-set cost and perform binary search over k until the error reaches the desired value. We suggest the use of a

single input parameter ϵ_{kL}, the maximum allowable average distance between a set of points and its assigned line. In general, choosing $\epsilon_{kL} \sim \epsilon_{LS}$ ensures that the errors in the two steps are approximately the same and that little computation effort is wasted. Note that this step uses position information only, without considering time as Step 1 does.

Step 3: Interval Clustering. Once the subtrajectories from Step 1 have been projected onto the lines generated in Step 2, the situation is reduced to clustering intervals in 1-D. Similarly to [8], we define an interval dissimilarity metric

$$\mathrm{ds1}([a,b],[c,d]) = \frac{|a-c|+|b-d|}{D}$$

where $D > 0$ is the total length of the shortest connected interval containing both $[a,b]$ and $[c,d]$. This expression measures the proportion of two intervals that do not overlap. Since we are also concerned with the direction of travel, we use the modified dissimilarity metric

$$\mathrm{ds2}([a,b],[c,d]) = \begin{cases} +\infty & (b-a)(d-c) < 0 \\ \mathrm{ds1}([a,b],[c,d]) & \text{else} \end{cases}$$

For each line and corresponding set of projections, we use an expectation-maximization (EM) clustering algorithm to find the minimum number of intervals that represent the projections. The algorithm alternates between finding the best representative for the intervals in a cluster and redistributing intervals to the closest representative. As an EM algorithm, it is guaranteed to converge to a local minimum of cost.

Input Parameter. The input parameter for this step is the per-interval cost ϵ_{IC}, which represents the fraction of two intervals that must overlap in order to be considered the same. Unlike the previous two parameters, ϵ_{IC} is not dependent on the scale of the input.

Step 4: Calculation of Representatives. The result of the previous three steps is a clustering of subtrajectories. To find the representative line segment for a cluster, we take the mean of the endpoints of the original segments in the cluster. Representatives are also assigned durations, the mean duration of the participating segments. This last compression of duration information is justified through our experiments, which showed that the distribution of durations for line segments in a cluster were approximately normal.

Iteration. It is possible to refine via iteration. Using the motion patterns outputted at Step 4, the process can repeat from Step 1 by partitioning the trajectory into subtrajectories such that each is well represented by one of the (now known) motion patterns. In this way, the algorithm as a whole becomes an EM algorithm, alternately partitioning the trajectory and clustering the resulting subtrajectories.

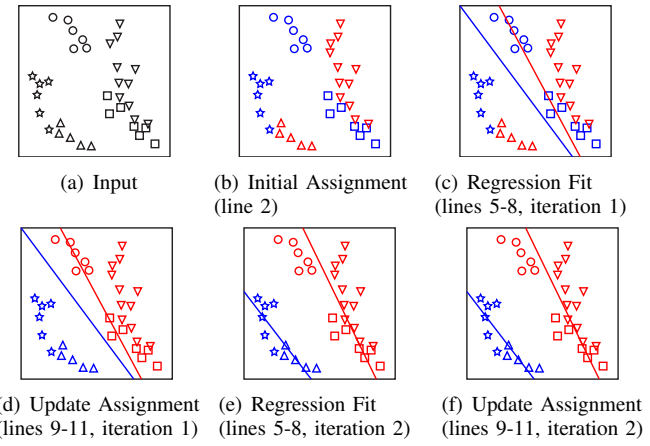
C. EM Algorithm for k-lines Variant

Our approximation algorithm for the k-lines variant proposed in Step 2 (see Alg. 1 and Fig. 3) is an application of the standard EM procedure to the modified k-lines center. At each iteration, lines are updated to best represent the sets of points assigned to them using orthogonal regression. The line assignments are then recalculated to minimize the average distance for a set of points to its assigned line. This EM algorithm will converge to a local minimum in cost over $\mathcal{L}$.

Algorithm 1: k-Lines Approximation

Data: n_c sets of points $S_1 = \{\mathbf{p}_1, \mathbf{p}_2, ...\mathbf{p}_{c_1}\}$, $S_2 = \{\mathbf{p}_{c_1}, \mathbf{p}_{c_1+1}, ...\mathbf{p}_{c_2}\}, ...$
Result: k lines $\mathcal{L} = \{\ell_1, \ell_2, ..., \ell_k\}$, line assignment for S_i, idx

```
// random initial line assigments
```
1 $oldidx = \mathbf{0}$; $idx = \mathrm{randint}(k, n_c)$;
2 **while** $oldidx \neq idx$ **do**
3 $oldidx \leftarrow idx$;
```
   // Update lines using orthogonal
      regression
```
4 **forall the** $\ell_j \in \mathcal{L}$ **do**
5 $P_j = \bigcup_{i|idx(i)=j} S_i$;
6 $\ell_j = \mathrm{mean}(P_j) + \mathrm{OrthogonalRegression}(P_j)$;
7 **end**
```
   // Update line assignments
```
8 **forall the** S_i **do**
9 $idx(i) = \underset{j}{\mathrm{argmax}}\left(\sum_{\mathbf{p}\in S_i} \mathrm{dist}(\mathbf{p}, \ell_j)\right)$;
10 **end**
11 **end**



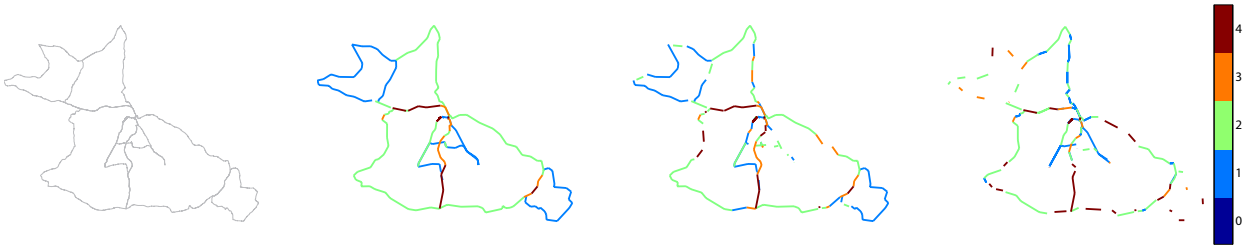
(a) Input (b) Initial Assignment (line 2) (c) Regression Fit (lines 5-8, iteration 1)

(d) Update Assignment (lines 9-11, iteration 1) (e) Regression Fit (lines 5-8, iteration 2) (f) Update Assignment (lines 9-11, iteration 2)

Fig. 3. Example run of Alg. 1 on 5 point sets for $k = 2$. Each set is shown in a different shape. Assignments to the 2 lines are shown in colors.

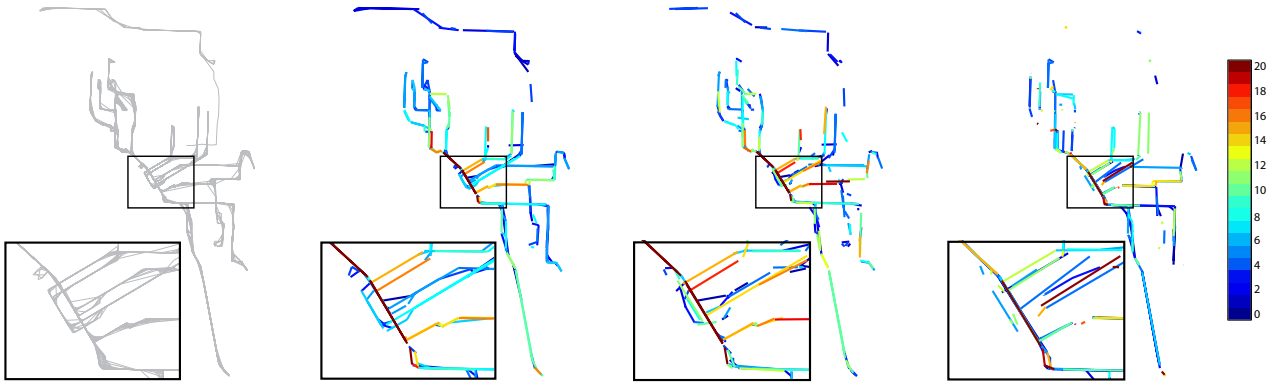
D. Evaluation

We tested the algorithm on two sets of GPS data: (Oxford) a 1000km data set containing 346,797 GPS points sampled at 5Hz, provided to us by the Mobile Robotics Group at Oxford University [11]; and (Rice) a Rice Community trace data set containing 7,277 GPS points, sampled about once every 30 seconds, of bus routes in Seattle, WA [12]. Following line simplification, the data sets were clustered manually and using the proposed clustering algorithm. We also compared the results to that of k-means using the distance metric proposed by Lee et al. [7], which is a function of the distance between segment endpoints and the angular difference.



(a) Original trajectory (b) Manual clustering (c) Our algorithm (d) k-means

Fig. 4. Sample of results for the Oxford data. Frequency plots show the motion patterns colored by the number of times they were traversed, as indicated by the scale on the right. (a) original trajectory data; (b) frequency plot for the trajectories as determined by a manual clustering; (c) frequency plot resulting from our algorithm; (d) frequency plot from using k-means.



(a) Original trajectory (b) Manual clustering (c) Our algorithm (d) k-means

Fig. 5. Sample of results for the Rice Community data. Frequency plots show the motion patterns colored by the number of times they were traversed, as indicated by the scale on the right. Overlays show magnifications of the rectangular area. (a) original trajectory data; (b) frequency plot for the trajectories as determined by a manual clustering; (c) frequency plot resulting from our algorithm; (d) frequency plot from using k-means.

Fig. 4 and 5 show frequency plots for the Oxford and Rice data respectively. The bold line segments are the cluster representatives, and the colors indicate the number of times each representative was traversed (i.e., the number of line segments in the cluster). Our clustering algorithm produces a structure very similar to that of the manual clustering, while k-means often misses sections of trajectory that were traversed infrequently, such as the bottom right of the Oxford trajectory and the top of the Rice trajectory. The magnified overlays in Fig. 5 compare performance in high-density areas. Our algorithm splits some clusters in the loop on the bottom right because the segments there are short and exhibit large angular variation. However, this is preferable to the tendency of k-means to merge clusters that should be distinct and even in opposite directions.

As a quantitative measure of the quality of clustering, we computed the purity measure, as defined in [13],

$$P = \frac{\sum_{c_i} \#\text{ segments assigned to } c_i \text{ that belong to } c_i}{\#\text{ segments}}$$

which gives the fraction of segments that were clustered correctly. In all cases, matchings between true and produced clusterings were assigned to maximize the purity score. Input parameters for both clustering schemes were also adjusted to achieve maximum purity. The results for 10 runs are in Table I. Our clustering outperforms k-means, correctly classifying 16.3% more of the line segments in the case of the Oxford data, and 21.4% in the case of the Rice data.

TABLE I
PURITY VALUES (%)

Clustering Scheme	Oxford	Rice
Our algorithm	84.9 (0.71)	75.9 (1.81)
k-means	68.6 (3.48)	54.5 (2.09)

mean (standard deviation) values for 10 runs each of our algorithm and k-means on the Oxford and Rice Community data.

IV. PREDICTION AND PLANNING FOR INTERCEPTION

In this section, we present a simple example of how trajectory clustering can be used to predict future motion and perform interception. We fit a Hidden Markov Model (HMM) to the sequence of cluster representatives traversed by the moving target and, using this model, project the movement of the target forward in time to determine the optimal interception point. We assume a point robot whose velocity can be directly controlled, subject to a velocity limit.

Step 1: Fitting a Hidden Markov Model. The output of trajectory clustering is a set of representatives $\mathcal{R}$ and their sequence in time. We assume that patterns in this sequence are time-independent and that the Markov assumption is valid. We also assume that the true state of the target is also a member of $\mathcal{R}$ but that it may not be exactly the same

as the observation due to errors in the clustering stage. We can therefore fit an HMM to the observation sequence, the representatives in the order in which they were traversed, to find patterns in the target trajectory. In our experiments, we used the standard Baum-Welch algorithm [14].

Step 2: Motion Prediction. Using the HMM, it becomes possible to predict where the target will be in the future. The first step is to determine which motion pattern the target currently is on given its position $\mathbf{p}_t$ and velocity $\dot{\mathbf{p}}_t$. This is a well-studied problem in itself and is commonly known in the GIS community as "map matching" (see [15] and references therein). For demonstration, in our experiments, we used the naïve approach of projecting $\mathbf{p}_t$ onto the closest state in $\mathcal{R}$ and setting that state as the current prior. We then used the HMM to find the most likely sequence of future states.

Step 3: Interception Planning. In order to plan an interception, we translate the predicted sequence of state transitions into an actual trajectory for the target. Since duration and velocity for all line segments belonging to a particular representative are approximately the same, we assign each state in the sequence a duration equal to the mean duration for the cluster and use linear interpolation to predict the target's future location $\mathbf{p}_{pred}$. To find the optimal interception point, we calculate the first time t such that $\frac{1}{t}\text{dist}(\mathbf{p}_r(t), \mathbf{p}_{pred}(t)) \leq u_{max}$. This is the earliest time when the target will be reachable to the robot. If no such t exists, the robot moves towards the location that minimizes $\text{dist}(\mathbf{p}_r(t), \mathbf{p}_{pred}(t))$.

V. Results

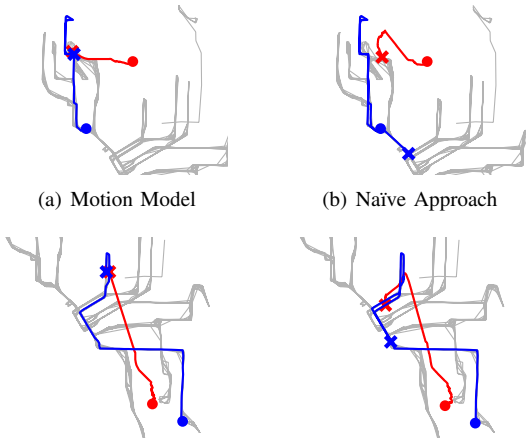
We simulated this setup for 3 situations. For each data set, we used the first half to detect motion patterns and train an HMM. We then tested interception with the target following a subtrajectory in the second half. The maximum speed of the robot was set purposefully lower than that of the target so that we could analyze situations where interception is not necessarily guaranteed for the naïve tracking controller.

The first situation we tested (Fig. 6) was a synthetic example showing the potential benefits of using historical data. The target (blue) moves back and forth with constant velocity along a horizontal line, and the robot (red) beginning at the bottom of the workspace, moves upwards to intercept the target. Our trajectory clustering algorithm correctly extracted the two states of the target (moving left and moving right) and the fitted HMM also yielded that the states alternate deterministically. The resulting trajectory taken by robot is shown in Fig. 6(a). The robot that knows the target's motion pattern moves towards the target's future position and intercepts it after only two back-and-forth motions. On the other hand, the robot who assumes constant velocity for the target (Fig. 6(b)) must always move towards the target, yielding multiple oscillations in the robot's own trajectory and a large increase in the amount of time until interception.

We next simulated the system using real data from the Rice Community traces. A comparison of the results with and without the motion model for two initial conditions are

(a) Motion Model (b) Naïve Approach

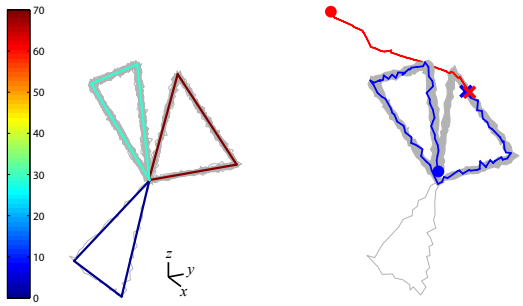
Fig. 6. Trajectories taken by the robot (red) to intercept the target (blue), where the target moves back-and-forth along the blue line. Starting locations are filled circles; ending locations are Xs. (a) results using the HMM-based prediction; (b) results for a constant-velocity assumption.



(a) Motion Model (b) Naïve Approach

(c) Motion Model (d) Naïve Approach

Fig. 7. Trajectories taken by the robot (red) to intercept the target (blue), where the target is a bus from the Rice Community trace data. Starting locations are filled circles; ending locations are Xs. The bus trajectory is shown in gray. (a,c) results using the HMM-based prediction; (b,d) results for a constant-velocity assumption. The simulations in (b) and (d) were halted before interception when it became clear the target had escaped.



(a) Frequency Plot (b) Interception Path

Fig. 8. Results of (a) trajectory clustering and (b) interception planning for a trajectory in 3D. The target (blue) chooses randomly between 3 triangular trajectories with nonuniform probability. The robot (red) chooses a path that intercepts the most common loop. This time, it succeeds.

shown in Fig. 7. In both cases, we can see that interception planning benefits from motion prediction. Using the motion model (Fig. 7(a,c)), the robot is able to predict that the target, upon reaching the end of a road, will turn around and is thus able to intercept it. The robot using the naïve approach (Fig. 7(b,d)) is unable to predict this U-turn, moves upwards too far in pursuit, and allows the target to escape.

Finally, we demonstrate how the trajectory clustering algorithm performs at higher dimensions using a synthetic trajectory for a target moving in 3-D space. In this example, the target chooses randomly between one of three triangular

loops with nonuniform probability. The frequency plot and an example interception path are shown in Fig. 8. Our k-lines center EM algorithm is agnostic to the number of dimensions and the interval clustering always occurs in 1-D, so using the algorithm on a trajectory of any dimensionality is straightforward and the quality of the results therefore do not differ significantly from any of the 2-D examples.

VI. Discussion

We contributed a novel trajectory clustering algorithm that yields higher quality results than basic k-means using arbitrary distance functions. It is among the first to identify and extract frequently traversed subtrajectories from long trajectory data without clear stopping points. Furthermore, to our knowledge, no other trajectory classification scheme takes time or directional information into account. One of our requirements for a cluster was that segments that occupy the same location in space but that are traversed in opposite directions be placed in different clusters. This allows segments to inherently encode some notion of intention, as compared to other point-based prediction schemes where velocity must be included as an additional state variable.

Our algorithm still requires some hand-tuning of parameters, but we believe the input parameters are more intuitive than Lee et al.'s [7] since they represent physically significant values rather than weights for combinations of distance functions. In addition, our algorithm was robust against parameter variation; input parameters could be changed by up to 50% without significantly affecting purity values. In the event that ϵ_{kL} cannot be determined, we also found that reasonable output could be produced for a range of small k between 5 and 30.

Our results show that motion prediction can be an effective tool in interception planning or tracking using velocity-limited vehicles, but many improvements remain to be made. The most common cause of failure in the trajectory clustering is short segments at odd angles, produced when large time gaps between observations cause trajectories to cut corners, as in the Rice data (see Fig. 5(a)). We would like to identify these segments separately so they are not considered motion patterns and potentially not included in the clustering at all.

Our goal is to use this system to coordinate task hand-off in a multi-robot system. The model for predicting target movement in this example is oversimplified for this goal. Identifying the current state of the target consisted simply of choosing the closest motion pattern. In experiments, this approach was correct about 80% of the time. However, as discussed in Step 2 of section IV, more sophisticated techinques exist that would provide more accurate estimations of the target's state. Similarly, interception planning was performed by choosing the most likely path to be taken by the target. However, the Hidden Markov Model gives us access to an entire probability distribution over potential trajectories the target will take. Motion planning algorithms such as those proposed in [16], [17] can be used instead to maximize the probability of interception, to minimize the expected time until interception, or to accomodate motion pattern durations that are not normally distributed.

VII. Conclusions

In this paper, we presented a data-driven approach to motion prediction and robotic interception. We proposed a novel trajectory clustering algorithm for identifying motion patterns in trajectory data and showed how the results could be used to build a simple model of a target's movement patterns. Our experiments showed that our approach provides advantages, for both trajectory clustering and motion prediction, over traditional methods. In addition, we suggest directions of further development that will improve the model for future applications in planning task hand-off.

References

[1] Z. Lin, V. Zeman, and R. Patel, "On-line robot trajectory planning for catching a moving object," in *Proc of the 1989 IEEE Intl Conf on Robotics and Automation*, 1989, pp. 1726–1731.

[2] F. Belkhouche and B. Belkhouche, "On the tracking and interception of a moving object by a wheeled mobile robot," in *Proc of the 2004 IEEE Conf on Robotics, Automation and Mechatronics*.

[3] D. Vasquez, T. Fraichard, and C. Laugier, "Growing Hidden Markov Models: An incremental tool for learning and predicting human and vehicle motion," *Intl J of Robotics Research*, vol. 28, no. 11-12, pp. 1486–1506, 2009.

[4] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning motion patterns of people for compliant robot motion," *Intl J of Robotics Research*, vol. 24, no. 1, pp. 31–48, 2005.

[5] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databases," *Foundations of Data Organization and Algorithms*, pp. 69–84, 1993.

[6] E. J. Keogh and M. J. Pazzani, "Scaling up dynamic time warping for datamining applications," in *Proc of the 6th ACM SIGKDD Intl Conf on Knowledge Discovery and Data Mining*, 2000, pp. 285–289.

[7] J. G. Lee, J. Han, and K. Y. Whang, "Trajectory clustering: a partition-and-group framework," in *Proc of the 2007 ACM SIGMOD Intl Conf on Management of Data*, 2007, pp. 593–604.

[8] D. Lymberopoulos, A. Bamis, and A. Savvides, "A methodology for extracting temporal properties from sensor network data streams," in *Proc of the 7th Intl Conf on Mobile Systems, Applications, and Services*, 2009, pp. 193–206.

[9] N. Hönle, M. Grossmann, S. Reimann, and B. Mitschang, "Usability analysis of compression algorithms for position data streams," in *Proc of the 18th ACM SIGSPATIAL Intl Conf on Advances in Geographic Information Systems*, 2010, pp. 240–249.

[10] N. Megiddo and A. Tamir, "On the complexity of locating linear facilities in the plane," *Operations Research Letters*, vol. 1, no. 5, pp. 194–197, 1982.

[11] M. Cummins and P. Newman, "Highly scalable appearance-only SLAM - FAB-MAP 2.0," in *Proc of Robotics: Science and Systems*, 2009.

[12] J. G. Jetcheva, Y.-C. Hu, S. PalChaudhuri, A. K. Saha, and D. B. Johnson, "CRAWDAD data set rice/ad_hoc_city (v. 2003-09-11)," Downloaded from http://crawdad.cs.dartmouth.edu/rice/ad_hoc_city.

[13] M. Nanni and D. Pedreschi, "Time-focused clustering of trajectories of moving objects," *J of Intelligent Information Systems*, vol. 27, no. 3, pp. 267–289, 2006.

[14] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[15] S. Funke and S. Storandt, "Path shapes: an alternative method for map matching and fully autonomous self-localization," in *Proc of the 19th ACM SIGSPATIAL Intl Conf on Advances in Geographic Information Systems*, 2011, pp. 319–328.

[16] D. Bertsekas and J. Tsitsiklis, "An analysis of stochastic shortest path problems," *Mathematics of Operations Research*, pp. 580–595, 1991.

[17] S. Lim, H. Balakrishnan, D. Gifford, S. Madden, and D. Rus, "Stochastic motion planning and applications to traffic," *Intl J of Robotics Research*, vol. 30, no. 6, pp. 699–712, 2011.