2-2005

Efficient Algorithms for the Identification of Potential Track/Observation Associations in Continuous Time Data

Jeremy Kubica
Carnegie Mellon University

Andrew W. Moore
Carnegie Mellon University, awm@cs.cmu.edu

Andrew Connolly
Carnegie Mellon University

Robert Jedicke
Carnegie Mellon University

Efficient Algorithms for the Identification of Potential Track/Observation Associations in Continuous Time Data

Jeremy Kubica, Andrew Moore, Andrew Connolly, Robert Jedicke

CMU-RI-TR-05-10

February 2005

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

In this paper we examine the problem of spatial data association - identifying which track/observations pairs could feasibly be associated. Efficiently and accurately finding these potential associations is vital for most tracking applications, because these associations both identify which target caused a given observation and update the estimate of a target's position and trajectory. However, previous work on efficiently answering this query often makes the limiting assumption that observations arrive in batches at discrete time steps. In many real world applications this may not be the case. Observations may arrive individually or in small batches distributed over a range of time. In this paper we focus on the question of efficiently identifying potential track/observations pairs in data where the observations can occupy a range of times.

We examine the new data structures and algorithms for efficient spatial data association on this type of data. We show that it is possible to adapt algorithms designed for discrete time data, providing the benefits of continuous time while retaining the tractability of discrete approaches. We introduce a novel data structure for dealing with large sets of tracks these queries. Empirically we show that these data structures provide a significant benefit both in decreased computational cost and increased accuracy when contrasted with treating the observations as arriving at a single time. Further, we show that in some cases it is more efficient to treat observations that do arrive at discrete time steps as if it were continuous time data and apply our techniques.

Contents

1 Introduction

One vital and often computationally expensive aspect of multiple target tracking is the question of data association: given a set of current track estimates and new observations, determine which observation corresponds to which track. The determined track/observation associations are used not only to identify which target caused a given observation, but more importantly to update the estimate of a target's position and trajectory. While there has been a vast number of proposed and developed approaches to both the general tracking and data association problems, many of these techniques include the underlying assumption that new sets of observations arrive together at discrete time steps. Below we remove this assumption and examine data structures and algorithms to quickly identify potential associations in "continuous time" data where observations may occupy a range of times. This corresponds to answering a continuous time *gating* query, which finds all points within some gate of the predicted track position.

Below we examine data structures and algorithms for observations that are distributed over a finite time interval. We show that data structures and algorithms designed for the discrete time steps can easily be adapted to function on continuous time observations, providing the benefits of continuous time while retaining the tractability of discrete time approaches. We introduce a novel data structure for dealing with large sets of tracks in continuous time queries. Empirically we show that these data structures provide a significant benefit both in decreased computational cost and increased accuracy. Finally, we discuss these approaches as compared to "flattening" the data to discrete time steps. We show that in some cases flattening the data to discrete time steps can lead to a loss in accuracy and an *increase* in computational cost.

While these approaches are applicable to such fields as target tracking and computer vision, our primary motivating example in this paper is asteroid tracking. Here we wish to determine which observed objects correspond to the same true underlying object from a series of visual observations of the night sky. These linkages can then be used to determine tentative orbits, attribute the observations to a known orbit, and assess potential risk of an asteroid. This domain is particular interesting for several reasons. First, the observations arrive in many small batches spread out over an entire viewing period. Each image covers only a small fraction of the night sky, and thus objects are observed over a range of times. We show that this type of structure is particularly well suited to benefit from the approaches that we present. Second, the asteroid tracking problem is a large-scale multiple target tracking problem. The high numbers of true objects and additional noise observations make exhaustive technique effectively intractable.

2 Problem Description

2.1 Overview of the Tracking Problem

In its most basic form sequential target tracking consists of three fundamental steps: prediction, data association, and filtering. In the first step, the current information

about the targets is used to predict their positions at the next time step. The predicted positions are then used to associate the tracks with new observations. In the final step, the associations are used to refine the information about each target. This process, shown in Figure 1, repeats at each time step with the arrival of new observations. For a good introduction to general target tracking see [Bar-Shalom *et al.*, 2001] or [Blackman and Popoli, 1999].

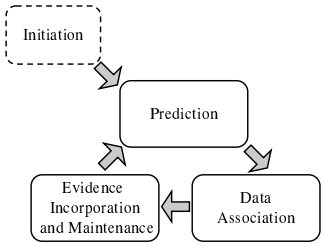

Figure 1: The flow of steps for basic sequential target tracking.

In this paper, we examine the computationally expensive step of data association. This step can easily be broken into two separate subproblems: *spatial* and *combinatorial* data association. In the first, potential track/observation associations are found such that the observation is "close" to the predicted position of its associated track. In the second subproblem, these potential associations are resolved by assigning the observations to tracks. Below we only examine the problem of spatial data association.

2.2 Notation

Tracks. At the heart of the problem we are interested in estimating tracks corresponding to both a set of observations and the trajectory of a true underlying target. We allow a general definition of a track as any function of the independent variable through the D dimensional space. We denote the ith track as $\mathbf{g}_i(t)$ and use N_T to denote the number of tracks. Our discussion below focuses primarily on two types of tracks: linear and quadratic. The quadratic track is simply a quadratic function of time:

$$\mathbf{g}(t) = \mathbf{a} \cdot t^2 + \mathbf{b} \cdot t + \mathbf{c} \tag{1}$$

and can be used to describe physical motions of objects undergoing constant acceleration. The linear track is a linear function of time:

$$\mathbf{g}(t) = \mathbf{b} \cdot t + \mathbf{c} \tag{2}$$

and can be used to describe the physical motion of objects traveling at a constant velocity. In addition, the linear model can be used for such queries as finding lines or edges described by the observations. While much of our discussion and techniques presented

below will also apply to other track models, we restrict the discussion to the linear and quadratic models to keep the discussion simple and consistent.

Observations. The observations take the form of D dimensional points. We denote the ith observation as $\mathbf{x}_i$ and indicate the time of observation as t_i. We use N_X to denote the number of observations.

2.3 Spatial Data Association

The problem of spatial data association can also be phrased as a filtering problem. Given a set tracks **G** and observations **X**, find all track/observation pairs such that the observation is "close" to the predicted position of the track:

$$dist(\mathbf{g}_j(t_i), \mathbf{x}_i) < \delta \tag{3}$$

where *dist* is the given distance measure. This measure may be simple, such as Euclidean distance, or may depend on the accuracy of the estimate of the tracks parameters. For example, Kalman filters provide an estimate of the covariance of the track's state that can be used to define a Mahalanobis distance.

Spatial data association is identical to *gating*, where observations are first filtered by whether they fall within a window or gate around the track's predicted position. An example gate is shown in Figure 2. Gating is used to avoid expensive probabilistic tests on obviously bad matches in many tracking algorithms.

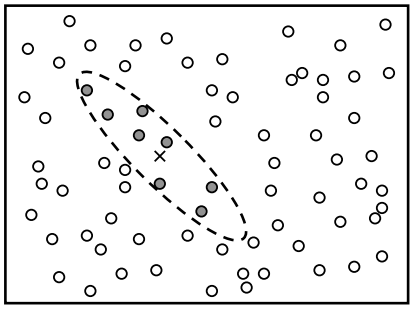

Figure 2: Gating can be used to ignore points that could not be part of the current track. The predicted position of the track is shown as an X and the points that fall within the gate are shaded.

Gating has successfully been used in conjunction with kd-tree structures to provide efficient spatial data association on discrete time data [Uhlmann, 1992, Uhlmann, 2001]. The predicted position of each of the tracks is calculated and stored in a kd-tree. This tree structure then allows us to efficiently find the tracks to which a given observation is "close." Below we look at extending the use of spatial data structures to perform efficient spatial data association when the observations' times span a continuous range.

2.4 Continuous Time Data

In many real world applications, the observations may not arrive together at discrete time steps. Instead they may be spread over an interval of time or reported in batches. For example, distributed sensors may report observations at different times or on different cadences. We call such systems continuous time systems to denote the fact that observations may not be arriving at a few discrete time steps. We define two different types of continuous time systems:

1. *Fully Continuous* - In a fully continuous time system, the observations arrive independently over some time interval. Each observation may occur at any real-valued time over this interval and thus no two observations may share exactly the same time.

2. *Semi-Continuous* - In a semi-continuous time system, observations arrive in discrete batches. However, each batch may contain only a fraction of the observations and thus many batches covering a range of time is required to observe each object.

For the most part we are interested in semi-continuous systems as they arise in many real-world systems. Most importantly astronomical observations are one particular semi-continuous time system. Each image of the night sky only contains a small fraction of the objects of interest and thus a full set of observations may be spread out over a significant period of time.

The distinction between these systems provides a natural hierarchical partition of observations as shown in Figure 3. The observations are partitioned into *viewings* and the viewings are partitioned into *plates*. The plates correspond to observations that are made nearly simultaneously ($t_{plate} - \varepsilon \leq t_i \leq t_{plate} + \varepsilon$), but may only cover a limited region of the space. In the semi-continuous case $\varepsilon = 0$ and in the fully continuous case $\varepsilon > 0$. In contrast, a viewing consists of a set of such plates, covering an extended region of space and time. The viewings do not necessarily need to cover the entire observation space, but may instead consist of a set of plates covering a coherent interval of time. For example, in asteroid tracking each plate may correspond to a single image and each viewing to a set of plates taken on a single night.

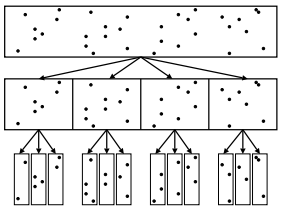

Figure 3: The different levels of observation sets (from top to bottom) all data $\rightarrow$ viewings $\rightarrow$ plates. The horizontal axis corresponds to time.

2.5 Flattening

A natural way to deal with continuous data is simply to *flatten* it onto discrete time steps [Uhlmann, 2001]. Specifically, we reassign each observation's time be that of the closest time step:

$$t_i = ARGMIN_{t \in \mathbf{T}} \left| t - t_i \right| \tag{4}$$

where **T** denotes the set of discrete time steps. Depending on the application we could choose to flatten the time steps into different granularities. Figure 4 shows an example of flattening one dimensional data onto viewings and plates.

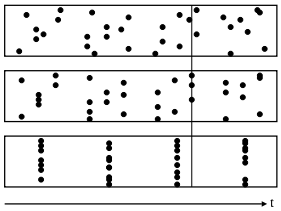

Figure 4: Observations flatten onto different granularities of time steps: all time → plates → viewings (from top to bottom). The midpoint time of a single plate is shown for reference. The horizontal axis corresponds to time.

It is important to note that flattening may introduce systematic noise into the data, by ignoring dynamics of the underlying track. Such an error is shown in Figure 5. However, for many tasks this will not be a realistic issue, as the shift in time and thus the systematic noise will be relatively small. Further it is possible that we can compensate for any error by simply increasing the size of the gate or augmenting the gate to account for potential track motion [Uhlmann, 2001].

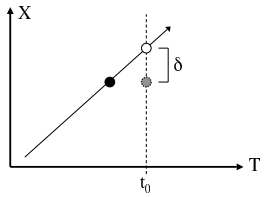

Figure 5: Flattening an observation (solid circle) to a discrete time step (shaded circle) introduces an error between the observed position and the predicted position (open circle).

Even in cases where it may be possible to safely flatten the data, we may be able to make significant computational savings by treating the data as continuous. This is especially true where each plate only covers a small fraction of the targets. By treating

the data as continuous and doing data association on many plates at once, we may be able to greatly reduce the computational cost. This benefit also applies to treating semi-continuous data and is thus applicable to many real-world domains.

3 Data Structures for Tracking with Continuous Time Data

A common and effective approach to making tracking tractable with a large number of targets is to use tree-based data structures to efficiently find potential track/observation associations. Uhlmann describes the use of kd-trees to accelerate spatial data association [Uhlmann, 1992, Uhlmann, 2001]. These trees are constructed on the predicted track positions *at each time step*. New observations are then matched with the tracks via a nearest neighbor type query on these trees. Similarly, it is possible to construct such trees on positions of the new observations.

If the observations do not arrive in discrete time steps, this approach may become less efficient. At the extreme, fully continuous data would require building a new tree for each individual observation, providing no computational advantage. One possibility is to flatten the observations onto discrete time steps introducing some systematic error. However, by developing approaches that work directly on the continuous time data, we can preserve the data's accuracy while possibly *increasing* computational efficiency.

3.1 Temporally Augmented KD-trees

Our first approach is to adapt current spatial data structures to explicitly handle observations with a range of times. Further, we adapt the queries to work with respect to *moving* points instead of just stationary ones. In doing so we can efficiently ask whether any observations fall near a query *trajectory* over some time interval of interest.

We use a tree structure that allows us to consider a range of time without flattening the data [Kubica *et al.*, 2004]. Specifically, we construct a *single* $D+1$ dimensional tree on *all* given observations by incorporating time as a dimension in the kd-tree. As shown in Figure 6, each node in the kd-tree is augmented to contain a bounding box on both the positions and times of observations owned by the node.

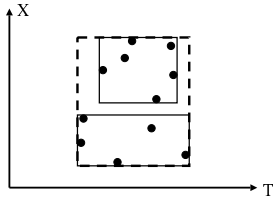

Figure 6: The bounding boxes for two kd-tree nodes is shown (solid boxes) with the bounding box for their parent node (dashed).

The tree is constructed in a recursive top-down fashion. At each level, the set of observations is split into two subsets that are recursively used to build the children nodes. The sets are determined by choosing the widest dimension of the node's bounding box and splitting on the midpoint of that dimension. Nodes are assigned to the left child if they fall below that threshold and the right child if they fall above it.

The search for potential track/observation associations then becomes a question of nearest neighbor to a curve. For a given query track $\mathbf{g}(t)$ we traverse the tree in a depth first search. If we hit a leaf node, we explicitly test whether the observations at that node are close to the track. Finally, we can prune the search if we ever find that the track cannot hit *any* observation contained within the bounds of the node. We can determine whether this criteria is met by taking the bounding box for the tree node (including time) and asking whether the track "comes close to" this box. If it does not, then we can safely prune the search.

3.2 Track-based Data Structures

Instead of placing the observations in a kd-tree, we could place the tracks in a tree structure. This alternative is especially important in cases where it is advantageous to track individual observations as they come in. For example in the astronomy domain, each image will include only a limited number of observations that we will want to match with the many current tracks. Further, we may want do this tracking as each image arrives. By building a tree structure on the tracks, we shift the focus to the larger and thus more important data set.

We use a track-based ball-tree to efficiently answer spatial queries involving trajectories [Kubica *et al.*, 2004]. Like a ball-tree or metric tree, the key idea behind this tree is a tight coupling between the pairwise fit between tracks and tree construction [Ciaccia *et al.*, 1997, Uhlmann, 1991]. Both the bounds for the nodes and the splits of the nodes during construction are determined by this distance measure. Specifically, the bounds of a node are defined with reference to a central *anchor* track, $\mathbf{g}_a$, and a node radius $\mathbf{r}$. The anchor track serves to indicate one possible predicted position of tracks in the node at any given time and the radius indicates the maximum offset from this prediction in each dimension. The radius is defined as the smallest $\mathbf{r}$ such that:

$$\left|\mathbf{g}_a(t)[d] - \mathbf{g}(t)[d]\right| \leq \mathbf{r}[d] \;\; \forall t_s \leq t \leq t_e \;\; \forall \mathbf{g} \in node \;\; \forall d \tag{5}$$

where t_s and t_e indicate the time interval of interest. Thus $\mathbf{r}[d]$ is the furthest distance from the anchor in dimension d of any track owned by the node at any time in the range of interest. An illustration of these bounds is shown in Figure 7.

We construct the track-based ball-tree in a recursive top-down fashion. At each level we split the tracks into two sets with respect to the distance function by choosing two well separated tracks and dividing the remaining tracks based on their proximity to those two tracks. Formally, we term the two tracks used for splitting $\mathbf{g}_R$ and $\mathbf{g}_L$. We assign a track $\mathbf{g}_i$ to the right hand child if and only if its distance to $\mathbf{g}_R$ is smaller than its distance to $\mathbf{g}_L$. Since we want to form two tight bundles of tracks at the next level, we use a distance function similar to the one that will determine the bounds, the

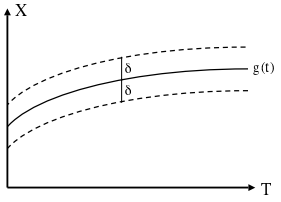

Figure 7: The track balltree nodes are defined in reference to an anchor track and radius. This radius indicates the maximum distance from the anchor track in and dimension.

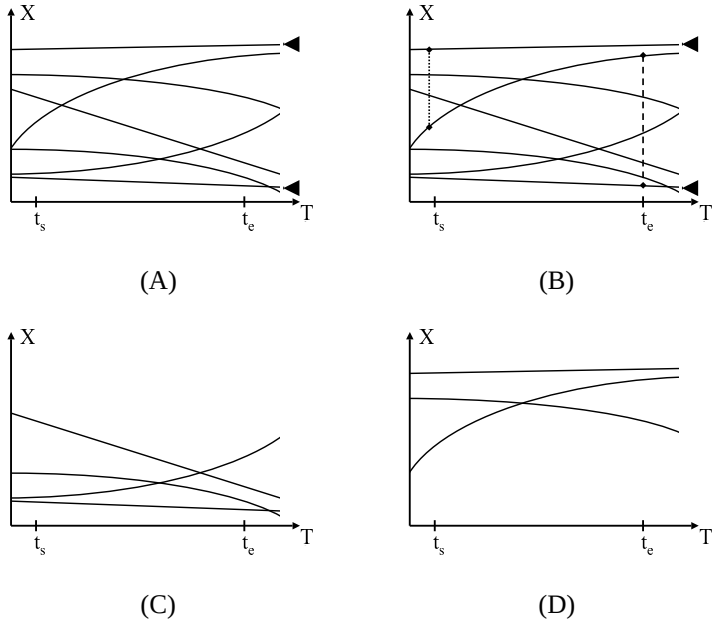

Figure 8: A set of tracks is split by choosing two temporary anchor tracks (A) and for each other track calculating which anchor is closer. The resulting partition is shown in (C) and (D).

maximum distance between two tracks on a given time interval $[t_s, t_e]$ with respect to some dimension d^*:

$$\Delta(\mathbf{g}_i, \mathbf{g}_j) = MAX_{t_s \leq t \leq t_e} \left| \mathbf{g}_i(t)[d^*] - \mathbf{g}_j(t)[d^*] \right| \tag{6}$$

The dimension d^* is choose to be the widest, and thus loosest, dimension. The splitting process is illustrated in Figure 8.

Again the search for track/observation pairs follows the same approach. Given a candidate observation $\mathbf{x}$ we do a depth first search of the tree. We prune any node if we can determine that no track owned by this node could be close to the observation:

$$\left|\mathbf{x}[d] - \mathbf{g}(t_{obs})[d]\right| > \delta[d] + \mathbf{r}[d] \tag{7}$$

4 Experiments

To test the performance of the algorithms we simulated data from quadratic tracks. The use of simulated data allows us to conduct controlled experiments on the effect of various parameters, such as the number of plates per viewing.

4.1 Tracking Algorithm

In the below experiments all tracking is done using Kalman filters with independent dimensions. Thus the track estimate consists of D separate 3-dimensional vectors (for position, velocity, and acceleration) and D separate 3×3 matrices. All tracks were started with the *correct* state estimate.

We used a very simple form of track oriented multiple hypothesis tracking. Specifically, we create a new set of tracks at each viewing period that consists of those tracks from the previous time steps that were extended with a new associated observation. The previous set is then discarded, making the extended tracks our current set of track estimates. To generate the new set, each track from the previous time step is associated with new observations. The criterion for a valid match is threshold distance of δ in each dimension:

$$\left|\mathbf{x}[d] - \mathbf{g}(t_{obs})[d]\right| \leq \delta[d] \tag{8}$$

Each time a track/observation pair produces a viable match, a new track is created from the pair and added to the new set of tracks. Thus one track can be associated with multiple observations and form multiple new tracks. Similarly, one observation may be associated with multiple tracks.

The above tracking algorithm results in very simple track maintenance logic. New tracks are never created from singleton observations. Rather tracks are always the result of extending one of the previous tracks. Further, tracks are pruned if they are not associated with *any* observation during the current viewing period. Thus a track is pruned if it does not see an observation where it expects during the viewing period. This pruning logic is a consequence of the fact that we create a new set of tracks each viewing period and populate it with those tracks from the previous step that have been extended by one observation.

While the above tracking algorithm is simplistic, it has several desirable qualities. Primarily, the algorithm returns *all* possible tracks that are compatible with the initial track estimates and the threshold. The lack of combinatorial data association removes any inherent pruning that might incorrectly remove a true track.

Finally, it should be noted that the data structures and algorithms were developed independently of the exact track logic used. While the track logic discussed above is

very simplistic, it is meant only as an illustrative example. The data structures and algorithms themselves are applicable to a wide range of tracking approaches.

4.2 Effect of Flattening

The first experiment was designed to demonstrate the potential effects of flattening on continuous time data. In many domains the amount of flattening may be small and thus the errors may be slight. However, it is interesting to look at cases where the flattening is significant. Such a case could occur in the asteroid tracking domain if each night was treated as a single time step and all observations for that night were flattened back to that time.

4.2.1 Experiment

The experiment used simulated data generated from 100 tracks:

- **c** $\sim$ uniform(-1.0, 1.0)
- **b** $\sim$ uniform(-0.1,0.1)
- **a** $\sim$ uniform(-0.01,0.01)

The observations were generated at 5 viewings, each of which contained a single plate. The midpoints of the plates were spaced 1 time unit apart and the width of the plate was varied from 0.0 to 0.8. Thus the data ranged from flat to nearly continuous. We tested two approaches: flattening the data and using the continuous data.

The performance of the algorithm was monitored by three different measures:

1. *Percent Correct* (P_C) - The percentage of returned tracks that exactly matched at least one true underlying track. This score measured the false positive rate.

2. *Percent Found* (P_F) - The percentage of true underlying tracks that exactly matched at least one of the returned tracks. This score measured the false negatives.

3. *Weighted Number Incorrect* (WI) - The weighted number of *incorrect* tracks (false positives) that have a better score than at least 10% of the true tracks. The weight of each track is determined by the number of true tracks the incorrect track's score exceeds. This score was calculated by sorting the returned tracks and traversing the list from best to worst according to their probability under the Kalman filter. At each step, the total score is incremented by the number of incorrect tracks seen so far. The counting stops after 90% of the true tracks have been seen. This calculation is illustrated in Figure 9. This score accounts for both the number and position of false positives.

These measures give a snapshot of algorithm performance for one setting of the gating threshold ($\delta[d] = 0.02 \ \ \forall d$). It is worth noting that we could trade off percentage correct and percentage found by varying that threshold.

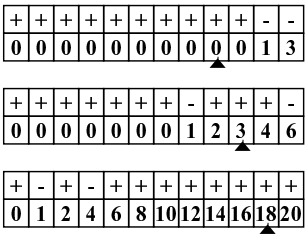

Figure 9: The calculation of "weighted number incorrect" for three different example results. The ordered correct (+) and incorrect (-) tracks are shown with the corresponding scores underneath. The triangles mark the 90% correct seen point and thus the final score. This score accounts for both the number and position of the false positives.

	Flattened			Non-Flattened		
Δt	P_C	P_F	WI	P_C	P_F	WI
0.0	0.009	0.992	1.402	0.009	0.992	1.402
0.2	0.009	0.992	1.361	0.009	0.992	1.405
0.4	0.009	0.989	1.608	0.009	0.992	1.412
0.8	0.009	0.975	3.821	0.009	0.990	1.478

Table 1: The performance of the multiple hypothesis tracker for both flattened and non-flattened data as the width of observations time increase. The columns indicate percent correct (P_C), percent found (P_F), and weighted number incorrect (WI).

4.2.2 Results and Discussion

The results, shown in Table 1, indicate the possible detriment to flattening the observations. At a plate width of $\Delta t = 0.0$ the data is already flat and both approaches produce the same results. As the plate width increases, over twice as many tracks are missed when flattening the data as compared to keeping continuous times. Further, the weighted error increases by over a factor of 2.

The above experiment paints an exaggerated, but possible, result. While in many cases the amount of flattening may be slight, it is important to consider whether systematic errors are being introduced.

4.3 Effectiveness of Tree Structures

The second experiment was designed to evaluate the computational performance of the various approaches and data structures. Even in cases where flattening may be a safe option, the use of continuous time data structures may still provide computational benefits.

4.3.1 Experiment

To test the efficiency of our proposed algorithms, we examined their relative performance as the number of plates was varied. The number of viewings was held constant and the number of plates K was varied from 5 to 5000. Each viewing covered a time range of 0.5, providing a gap of 0.5 between viewings. The plates were equally spaced over this time span and formed a disjoint partitioning of the sky. Each object appeared in exactly one plate per viewing. As illustrated by Figure 10 the plates provided spatial structure, only containing those observations that were in a given region of space at a given time.

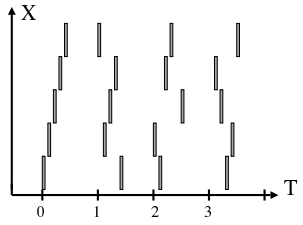

Figure 10: Each of the 4 viewings is divided into 5 plates that tile the observed space.

The actual observations were generated from either 5000 or 10000 random tracks:

- $\mathbf{c} \sim$ uniform(-100.0, 100.0)
- $\mathbf{b} \sim$ uniform(-5.0,5.0)
- $\mathbf{a} \sim$ uniform(-1.0,1.0)

The performance was measured by the computational cost of performing the multiple hypothesis tracking. Specifically, we looked at three measures:

1. The number of distance computations.
2. The number of pruning queries during tree traversal.
3. The wall-clock running time (including data structure construction).

The first two measures indicate processor independent cost, but do not account for time required to build data structures. In contrast, the wall-clock running time indicates the total computational cost, but is processor dependent and may be reduced by further optimizations, such as data reordering for cache friendliness.

4.3.2 Data Structures

We tested both data structures and an exhaustive search:

- *Exhaustive* - At each time step, iterate through all pairs of tracks and observations. This approach is denoted E in the results.

- *Observation-based kd-tree* - Construct a kd-tree of observations at each time step (either plate or viewing) and use this data structure to quickly find candidate observations for each query track. This approach is denoted T_O in the results.

- *Track-based Ball-tree* - Construct track-based ball-tree on the tracks at the beginning of *each viewing*. Use this tree to quickly find candidate tracks for each query observation. This approach is denoted T_B in the results.

In addition to the different approaches indicated above, we examined two different strategies for when to perform the data association:

- *Viewing Based* - Perform the data association after each viewing. Thus we wait until all plates at that viewing have arrived. This approach is denoted with a superscript V in the experiment.

- *Plate Based* - Perform the data association after each plate. This approach corresponds to a more "online" tracking approach where we process the new observations as they arrive. This approach is denoted with a superscript O in the experiment.

The two different strategies will have the most significant effect on the observation-based kd-tree. Under the viewing-based approach one kd-tree is built on all observations of each viewing. In contrast, under the plate-based approach, one kd-tree is built from the observations of each plate. Thus $K-1$ more trees are built.

Note that despite the use of two different sized time steps for data association, the track maintenance and filtering is only performed after each viewing. In other words, even with a plate-based strategy, the set of tracks given at the beginning of the viewing is used for all data associations even if new associations are found. This approach means that both strategies return the *same* result and can directly be compared. Further, since we are only updating our estimates of the track parameters at the end of each viewing, the track-based ball-tree only has to be constructed once for all plates in the viewing. Adaptations that allow multiple observations per time step are discussed in Section 5.

4.3.3 Results and Discussion

The results are shown in Tables 2, 3, and 4. Tables 2 shows the wall-clock running times (including tree construction) for each of the approaches. Table 3 shows the number of track/observation distance queries performed by each approach. Finally, Table 4 shows the number of pruning queries performed by each approach.

The results demonstrate the sheer intractability of an exhaustive approach to spatial data association. While this in itself is well established, when combined with continuous or semi-continuous time data it illustrates the potential need for algorithms that use the spatial structure of the data while preserving the continuous time aspect.

N	K	E^V	E^P	T_O^V	T_O^P	T_B^V	T_B^P
5000	5	39.1	22.5	2.9	3.2	3.7	3.6
5000	10	39.7	21.9	2.9	3.2	3.6	3.5
5000	50	40.3	22.4	2.9	4.9	3.4	3.3
5000	100	41.5	23.8	3.1	7.3	3.4	3.2
5000	500	47.7	36.3	3.3	26.2	3.5	3.1
10000	5	190.2	110.2	6.3	7.8	9.3	9.2
10000	10	193.1	89.4	6.8	7.9	8.7	8.8
10000	50	194.7	86.2	6.9	11.3	8.2	8.2
10000	100	197.9	88.7	7.0	16.5	8.0	7.9
10000	500	215.6	116.1	7.8	58.2	8.3	7.6

Table 2: The average running time (in seconds) for multiple hypothesis trackers over different numbers of observations and plates.

N	K	E^V	E^P	T_O^V	T_O^P	T_B^V	T_B^P
5000	5	127.32	127.32	0.116	0.227	0.135	0.135
5000	10	127.37	127.37	0.120	0.213	0.135	0.135
5000	50	127.35	127.35	0.149	0.201	0.135	0.135
5000	100	127.30	127.30	0.165	0.212	0.135	0.135
5000	500	127.30	127.30	0.208	0.337	0.135	0.135
10000	5	519.2	519.2	0.235	0.561	0.336	0.336
10000	10	518.8	518.8	0.243	0.521	0.335	0.335
10000	50	519.1	519.1	0.313	0.500	0.335	0.335
10000	100	519.0	519.0	0.358	0.532	0.334	0.334
10000	500	518.9	518.9	0.473	0.756	0.334	0.334

Table 3: The average number of distance computations (in millions) for multiple hypothesis trackers over different numbers of observations and plates.

N	K	E^V	E^P	T_O^V	T_O^P	T_B^V	T_B^P
5000	5	0.0	0.0	0.700	0.978	0.676	0.676
5000	10	0.0	0.0	0.689	0.992	0.676	0.676
5000	50	0.0	0.0	0.795	1.849	0.676	0.676
5000	100	0.0	0.0	0.856	3.086	0.675	0.675
5000	500	0.0	0.0	1.035	13.135	0.675	0.675
10000	5	0.0	0.0	1.590	2.540	1.633	1.633
10000	10	0.0	0.0	1.582	2.528	1.632	1.632
10000	50	0.0	0.0	1.886	4.252	1.632	1.632
10000	100	0.0	0.0	2.068	6.827	1.632	1.632
10000	500	0.0	0.0	2.494	27.604	1.631	1.631

Table 4: The average number of pruning computations (in millions) for multiple hypothesis trackers over different numbers of observations and plates.

The results also indicate several very important trends when dealing with data with a high number of plates. Conventional wisdom might suggest that we build one kd-tree on the observations from *each* plate. In general, this would require very little flattening and allow us to harness the spatial structure of the data. However if we are interested in performing the spatial data association only once per viewing, then we can achieve a significant computational advantage by treating the semi-continuous time data as fully continuous and building a single tree. Further, if we are interested in performing spatial data association at each plate (such as for an online tracking application), then it may be computationally advantageous to use a track-based structure. The track-based structures are then built on the larger data set and further do not need to be completely rebuilt for each new plate.

Finally, it should be noted that the above experiments provided an *optimistic* view of data on different plates. Because each plate only contained observations from the same region, these plates themselves contained a significant amount of spatial structure. When a single kd-tree was built on each plate, the trees had the advantage that a significant number of tracks could be pruned at the *top level*.

5 Multiple Track Observations per Viewing

One natural extension to this work is domains in which we may see *multiple* observations of the same target during single viewing. We can easily account for this possibility by returning a *sorted* list of observations that could feasibly match a given track. These observations are sorted in time. We can then perform exhaustive multiple hypothesis tracking on this reduced set of observations. Note that while this extension is simple for spatial data association, it may lead to significantly more complicated track filtering and maintenance logic.

6 Related Work

There have been several approaches suggested to performing spatial data association with continuous time data. Uhlmann proposes two different approaches for handling (semi-)continuous data consisting of batches of observations arriving from components of a distributed missile tracking system [Uhlmann, 2001]. In the first approach, viewings are grouped into batches covering a given time interval and the tracks are projected to the middle time of the batch. The tracks' gates are extended to account for the maximum potential movement during this time interval. Spatial data association is done by finding all observations in the batch that fall within the extended gate. In the second approach, the tracks' gates are again extended to account for motion, but this time accounting for the motion model. A tree can then be constructed on these gates and the observations used as queries. Since the gates will only be valid for a given time span, elements in the tree must be maintained. Neither of these approaches take full advantage of the temporal aspects of the data. They use a single gate over a range of time. In contrast, our approaches effectively use a gate that moves with the track's motion model and can grow/shrink with time to accommodate uncertainty in the track

estimate.

The use of tree-based structures on regions and points to accelerate trajectory based queries has also been used in other fields. For example, in the problem of ray-tracing, placing objects in tree structures is a common and successful approach [Glassner, 1984, Watt, 2000]. Our temporal kd-trees are similar to such approaches, but are extended to make use of the independent variable and work with nonlinear tracks.

The approach of building data structures on tracks has also been considered in a variety of domains. Arvo and Kirk proposed *ray classification*, a technique to accelerate computer ray tracing [Arvo and Kirk, 1987]. Rays are represented as points in 5-dimensional parameter space and partitioned into different groups. A similar technique has been presented in databases to answer queries about moving objects [Kollios *et al.*, 1999, Saltenis *et al.*, 2000, Papadopoulos *et al.*, 2002]. Again, the linear tracks are effectively treated as points in parameter space for the construction of a tree structure. By bounding the parameters, it is possible to create time parameterized bounds for the node in observation space. While this work has been restricted to linear models, it is important to note that the resulting trees are valid, but not optimal, for *all time*. In previous work we tested a similar method, extended to more complex track models, to provide comparison with the more efficient ball-tree based data structure [Kubica *et al.*, 2004]. Finally, Pfoser *et. al.* presented two tree models for querying piecewise linear tracks [Pfoser *et al.*, 2000]. These structures exploited the fact that 2-dimensional tracks could be broken into line segments and treated as a set of 3-dimensional objects.

7 Conclusions

Above we examined the problem of spatial data association on continuous time data. We examined the approach of flattening and showed that a "large" change in time can lead to significant error and thus reduce accuracy. Combined with the intractability of exhaustive methods in large domains, these results indicate the need to adapt spatial structure algorithms to continuous time data. Further, the use of continuous time methods can lead to an improvement in performance, even on some discrete time data.

We presented two algorithms designed to work in continuous time data: a track-based ball-tree and a observation-base kd-tree. We showed that both of these algorithms could lead to a significant computational savings on continuous time data without adding systematic noise. Finally, we showed that it may also be advantageous to use these approaches on discrete time data, such as observations with many plates. The continuous time data structures allow us to exploit the track's movement through time and thus can prune regions of space over entire spans of time, reducing computations from performing those prunings at each individual time.

Acknowledgements

Jeremy Kubica is supported by a grant from the Fannie and John Hertz Foundation. This project was supported by a grant from the National Science Foundation (Grant CCF-0121671).

References

[Arvo and Kirk, 1987] James Arvo and David Kirk. Fast ray tracing by ray classification. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 55–64. ACM Press, 1987.

[Bar-Shalom *et al.*, 2001] Yaakov Bar-Shalom, X. Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. Wiley-Interscience, 2001.

[Blackman and Popoli, 1999] Samuel Blackman and Robert Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.

[Ciaccia *et al.*, 1997] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB International Conference*, September 1997.

[Glassner, 1984] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, 1984.

[Kollios *et al.*, 1999] George Kollios, Dimitrios Gunopulos, and Vassilis Tsotras. On indexing mobile objects. In *Proc. of the 18th ACM Symp. on Principles of Database Systems*, 1999.

[Kubica *et al.*, 2004] Jeremy Kubica, Andrew Moore, Andrew Connolly, and Robert Jedicke. Spatial data structures for efficient trajectory-based queries. In *CMU Tech. Report 04-61*, 2004.

[Papadopoulos *et al.*, 2002] Dimitris Papadopoulos, George Kollios, Dimitrios Gunopulos, and Vassilis Tsotras. Indexing mobile objects on the plane. In *MDDS*, 2002.

[Pfoser *et al.*, 2000] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proc. 26th Int'l Conference on Very Large Databases*, September 2000.

[Saltenis *et al.*, 2000] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD Conference*, pages 331–342, 2000.

[Uhlmann, 1991] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[Uhlmann, 1992] J. K. Uhlmann. Algorithms for multiple-target tracking. *American Scientist*, 80(2):128–141, 1992.

[Uhlmann, 2001] J. K. Uhlmann. Introduction to the algorithmics of data association in multiple-target tracking. In David L. Hall and James Llinas, editors, *Handbook of Multisensor Data Fusion*, pages 3.1–3.18. CRC Press, 2001.

[Watt, 2000] Alan Watt. *3D Computer Graphics*. Addison-Wesley, 3rd edition, 2000.