Spatial Querying for Camera-Based Tracking Platforms

R. Onur Keskin
Brown University, Computer Science Department, Providence, RI 02912

Abstract

We propose a set of techniques for querying camera-based tracking data effectively. We build an infrastructure for efficient processing of different types of spatial queries, including trajectory-based similarity and pattern matching queries. To achieve this, we project the raw tracking data from all the cameras into a global coordinate system and reduce the projection noise through partitioning methods. We represent the trajectories with polynomial approximations. We also introduce a user interface where user can query the data intuitively by simply sketching trajectories in a given 2d scene model.

1 Introduction

One of the most important tasks for camera-based tracking platforms is querying. However, there is a lack of study on building a complete system which would allow us to make spatial queries on the raw tracking data.

In camera-based tracking platforms, the data is represented in camera coordinate system. This kind of data usually is not meaningful for spatial querying since it includes perspective information added by the position of the camera. For more accurate and useful information retrieval, the data from all cameras should be converted to the same coordinate system.

Camera-tracking based trajectory data is highly noisy. Even though, from a coarse-grained view, trajectories are consistent, from a fine-grained view they are highly noisy. Tracking data usually has a noise in the form of zigzag pattern. We can remove the noise in a more effective way by taking this pattern into consideration.

We mainly focused on similarity matching queries. There have been studies using MBRs(Minimum Bounding Rectangle) for such queries[4],[5]. However, these methods either suffer a loss in pruning power or require expensive preprocessing[2]. We choose to use polynomial approximations for representing the trajectories. Previous studies show that polynomial approximations are very effective for comparing two trajectories, thus allow building indexing structures easily[6].

The main contribution of our work is to provide a framework for querying camera-tracking based trajectory data effectively. We use plane projective transformation in order to combine the raw tracking data from different cameras into a global coordinate system. Considering the distinctive properties of the tracking noise, we propose a partitioning algorithm for noise removal. And, we approximate trajectories with polynomials in order to satisfy different types of queries, including similarity matching and pattern matching queries. We also introduce a user interface for querying. The interface allows users to query visually and use join operators with different types of queries.

The rest of the paper is organized as follows. In section 2, we introduce our data model and discuss the techniques for converting the raw tracking data to our data model. In section 3, we show how to represent a trajectory by using polynomials and we discuss similarity metrics used for trajectory comparison. Section 4 introduces our visual querying interface and describes the key components.

2 Trajectory Model

In camera-based tracking platforms, the tracking data is usually represented in camera coordinate systems. The lack of global location information limits querying capabilities. We project the raw tracking data from all the cameras into a global coordinate system while

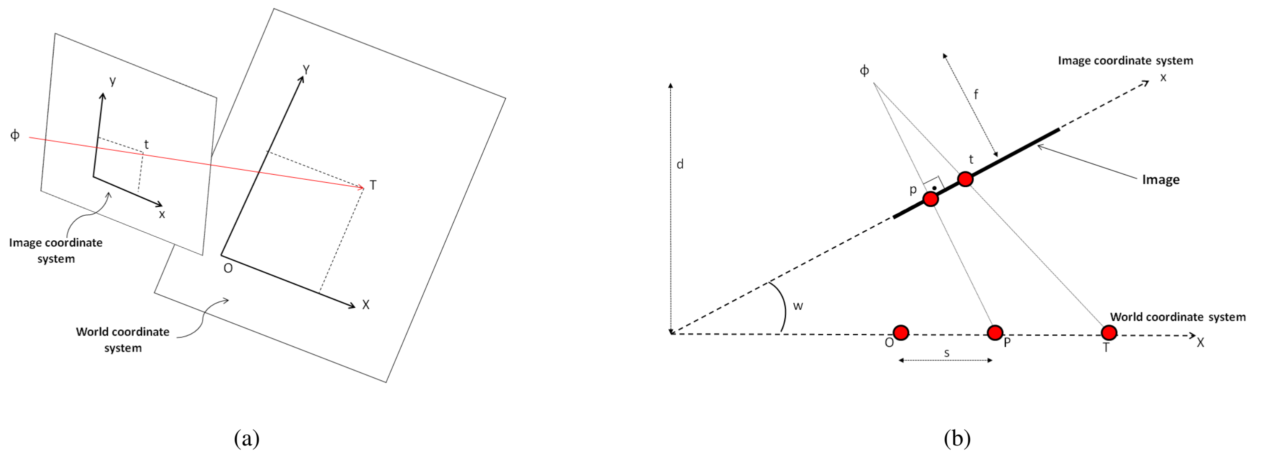


Fig. 1: (a) Perspective mapping: a point T on the world plane is imaged as t. ϕ is the camera center. (b) One-dimensional Camera Model: The camera center is a distance f (the focal length) from the image line. The ray at the principal point p is perpendicular to the image line, and intersects the world line at P, with world ordinate s. w is the angle between the world and image lines.

eliminating the noise with partitioning.

2.1 Plane Projective Transformation

The data represented in camera coordinate system includes the perspective information effected by the position of the camera. This kind of data usually is not meaningful enough for querying since, in this setup, every query needs to be done in a specific cameras coordinate system. However if we convert this data to the real world coordinate system, the querying can be done globally and more easily.

Points on the world plane are mapped to points on the image plane by a plane to plane homography, also known as a plane projective transformation(Fig. 1). A homography is described by a 3×3 matrix H(Eq. 1). Once this matrix is determined, the projection of an image point to a point on the world plane is straightforward.

$$X = Hx \tag{1}$$

The matrix can be computed from the relative positioning of the two planes and camera center(Eq. 2,3). However, it can also be computed directly from image to world point correspondences. Due to some advantages, which we will describe in future work section, we use the latter one.

$$\begin{bmatrix} X \\ 1 \end{bmatrix} = H_{2\times2} \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{2}$$

$$H_{2\times2} = \begin{bmatrix} \alpha & s \\ \mu & 1 \end{bmatrix} \tag{3}$$

where $H_{2\times2}$ is a homography matrix. For the geometry shown in Fig. 1b, the matrix is defined by $\alpha = \frac{d}{f\cos^2 w} - \frac{s}{f}\tan w$ and $\mu = -\frac{\tan w}{f}$.

In order to compute H directly from image to world point correspondence, we need to solve the following equation for H.

$$\begin{bmatrix} XW \\ YW \\ W \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4}$$

using above equation we can derive

$$\begin{bmatrix} x & 0 & .. & x_n & 0 \\ y & 0 & .. & y_n & 0 \\ 1 & 0 & .. & 1 & 0 \\ 0 & x & .. & 0 & x_n \\ 0 & y & .. & 0 & y_n \\ 0 & 1 & .. & 0 & 1 \\ -Xx & -Yx & .. & -X_n x_n & -Y_n x_n \\ -Xy & -Yy & .. & -X_n y_n & -Y_n y_n \end{bmatrix}^T \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} X \\ Y \\ . \\ . \\ X_n \\ Y_n \end{bmatrix} \tag{5}$$

The form of this equation is $A\lambda = B$ and we can determine λ by

$$\lambda = \left(A^T A\right)^{-1} A^T B \tag{6}$$

By using a number of known $\{(x,y),(X,Y)\}$ couples(Fig. 2), we solve this equation for λ. From now on, calculation of the image to world point correspondence for any point x is straightforward(Eq. 1).

2.2 Noise Reduction

Due to the nature of camera-based tracking, the data is highly noisy. Some of these are caused by misdetections and some are caused by unprecise tracking. Misdetections are usually easy to eliminate. They usually occur as extreme outliers and are easy to eliminate by using simple parameters like median step size of the trajectory.

Unprecise detection is an expected result in camera-based platforms. This unpreciseness is usually in the form of a zigzag pattern. In order to remove this kind of noise, we need to take this pattern into consideration. In the following section we propose an improved version of the trajectory partitioning algorithm presented in [1].

2.2.1 Trajectory Partitioning

Each trajectory is optimally partitioned and represented by a smaller set of points. We try to find the critical points where the behavior of the trajectory rapidly changes. For a trajectory, $TR = \{p_1, p_2, p_3...., p_{len_{TR}}\}$, we determine a set of critical points, $C_{TR} = \{p_{c1}, p_{c2}, p_{c3}...., p_{c_{len_{TR}}}\}$, where the behavior of the trajectory rapidly changes.

We use the minimum description length(MDL) principle in order to find the critical points. The MDL cost consists of two components: $L(H)$ and $L(D|H)$. Here, H means the hypothesis, and D the data. The two components are informally stated as follows [9]: "$L(H)$ is the length, in bits, of the description of the hypothesis; and $L(D|H)$ is the length, in bits, of the description of the data when encoded with the help of the hypothesis." The best hypothesis H to explain D is the one that minimizes the sum of $L(H)$ and $L(D|H)$. We formulize $L(H)$ and $L(D|H)$ as

$$L(H) = \sum_{j=1}^{len(C_{TR})} log_2\left(dist\left(p_{cj}, p_{cj+1}\right)\right) \tag{7}$$

$$\begin{aligned} L(D|H) = & \sum_{j=1}^{len(C_{TR})} \sum_{k=c_j}^{c_{j+1}-1} \{log_2\left(d_\perp\left(p_{cj}p_{cj+1}, p_{kj}p_{kj+1}\right)\right) \\ & + log_2\left(d_\theta\left(p_{cj}p_{cj+1}, p_{kj}p_{kj+1}\right)\right)\} \end{aligned} \tag{8}$$

Let $MDL_{par}(p_i, p_j)$ denote the MDL cost($= L(H) + L(D|H)$) of a trajectory between p_i and p_j when assuming that p_i and p_j are only the critical points - a trajectory which consists of just two points, p_i and p_j. Let $MDL_{nopar}(p_i, p_j)$ denote the MDL cost when assuming that there is no critical points between p_i and p_j - the original trajectory. For a trajectory TR, we try to find the longest subtrajectory $p_i p_j$ that satisfies $MDL_{par}(p_i, p_j) \leq MDL_{nopar}(p_i, p_j)$ for every i,j such that $i < j$ and $1 \leq i \leq len(TR)$, $1 \leq j \leq len(TR)$.

In camera-based tracking systems, a noise in the form of zigzags tend to be added. Knowing that the noise follows a pattern, we can improve the algorithm proposed in [1].

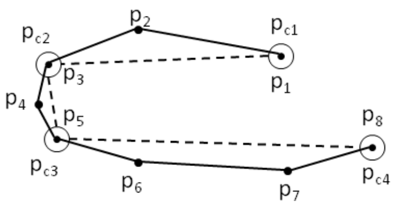
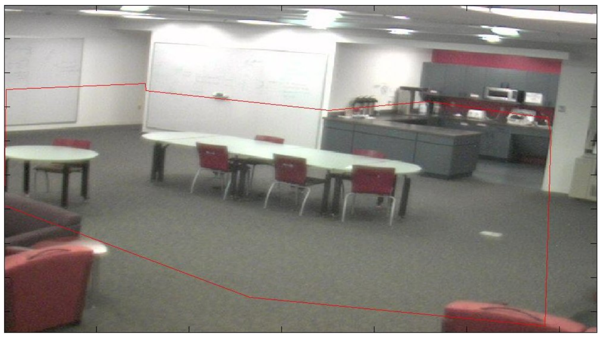


Fig. 3: Example of trajectory partitioning

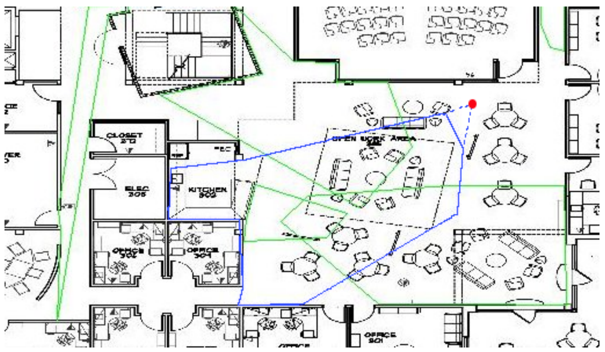
3 Approximating Trajectories with Polynomials

We approximate trajectories with polynomials. There are many possibilities for choosing the polynomial to be used, though we choose to use Chebyshev polynomials because of it's minimax approximation property[1]. [1],[2] proposed Chebyshev polynomial approximations and showed that they are easy to compute.

In many tracking systems, the data is obtained by periodic resampling and previous studies depend on



(a)



(b)

Fig. 2: We select the points from camera image(a) and floor plan(b) which correspond to each other. Using these (x,X) couples, we calculate the homography matrix, H. The red contours in the camera image are represented with blue contours in floor plan. The green contours are mappings of other cameras.

Algorithm 1 Modified Trajectory Partitioning

```
INPUT: A trajectory, $TR = \{p_1, p_2, p_3...., p_{len_{TR}}\}$
OUTPUT: A set of critical points, $CP$

Add $p_1$ into the set $CP$;
$startIndex := 1, length := 1, tolerance := 0$;
while $startIndex + length \leq len_{TR}$ do
  $currIndex := startIndex + length$;
  $cost_{par} := MDL_{par}(p_{startIndex}, p_{currIndex})$;
  $cost_{nopar} := MDL_{nopar}(p_{startIndex}, p_{currIndex})$;
  if $cost_{par} > cost_{nopar}$ then
    if $tolerance == 2$ then
      $currIndex := currIndex - tolerance$;
      $tolerance := 0$;
      Add $p_{currIndex-1}$ into the set $CP$;
      $startIndex := currIndex - 1$;
      $length := 1$;
    else
      $length := length + 1$;
      $tolerance := tolerance + 1$;
    end if
  else
    $length := length + 1$;
    $tolerance := 0$;
  end if
end while
Add $p_{len_{TR}}$ into the set $CP$;
```

this assumption. In our system, there are two sources of trajectory data. One is our camera-based tracking system and the other one is the visual querying interface. As in previous studies, our tracking system produces a periodicly resampled data. However, the data from the querying interface are created by user interaction, thus have different resampling rates depending on interaction speeds. These two different sources, causes different behaviors on time-series and leads to different polynomial approximations even though the spatial components are similar. For consistent approximations of trajectories from different sources, we propose a formula for calculating time points. For a trajectory, $TR = \{\langle p_1, t_1\rangle, \langle p_2, t_2\rangle, \langle p_3, t_3\rangle, ..., \langle p_{len_{TR}}, t_{len_{TR}}\rangle\}$, a time point t_i is calculated by the following formula:

$$T(i) = \begin{cases} 0 & i = 1 \\ dist(p_i, p_{i-1}) + T(i-1) & i > 1 \end{cases} \tag{9}$$

As in similarity-based matchings, we address the pattern-based matching problem by using polynomials. For each trajectory TR_i, we get the vector from the first point to the last point, $\vec{V_i} = \overrightarrow{P_0 P_{len_{TR_i}}}$ (Fig. 4). Using $\vec{V_i}$. we equalize the transformations of each trajectory. Thereafter the process is same with the similarity-based matchings; we approximate the trajectories with polynomials and compare them by using a similarity metric. In next section, we show how to approximate a trajectory with Chebyshev polyno-

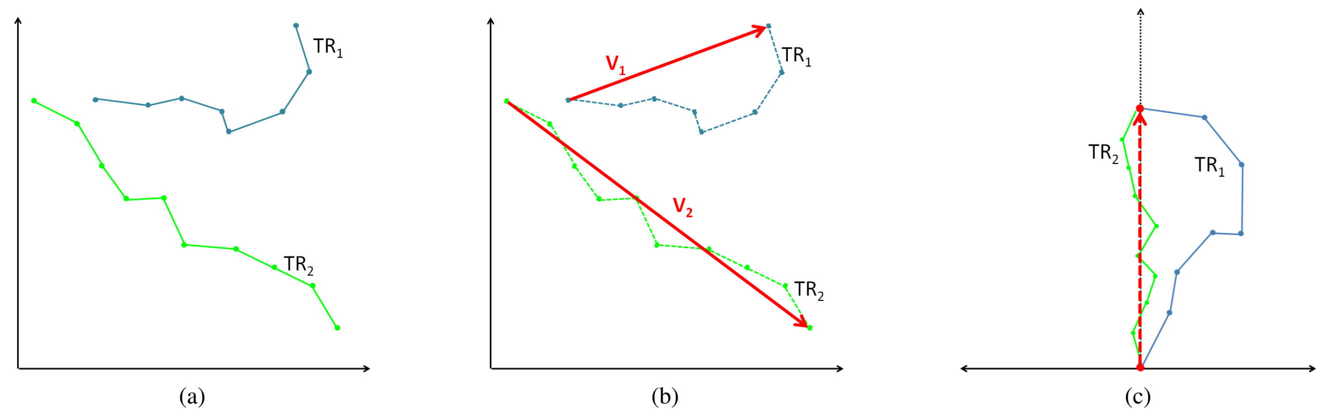


Fig. 4: (a) Trajectories on XY plane. (b) For each trajectory TR, we get $\vec{V} = \overrightarrow{P_0 P_{len_{TR}}}$ (c) We register each trajectory TR_i, by using $\vec{V_i}$

mials and introduce a similarity metric for comparing trajectories.

3.1 Chebyshev Polynomials

The Chebyshev polynomials of the first kind are defined by the recurrence relation

$$\begin{aligned} P_1(t) &= 1 \\ P_2(t) &= t \\ P_n(t) &= 2tP_{n-1}(t) - P_{n-2}(t) \end{aligned} \tag{10}$$

An arbitrary polynomial of degree M can be written in terms of the Chebyshev polynomials. Such a polynomial $f(t)$ is of the form

$$f(t) = \sum_{n=1}^{M} c_n P_n(t) \tag{11}$$

We can compute the coefficient c_n by the following formulas.

$$\begin{aligned} c_0 &= \frac{1}{M}\sum_{n=1}^{M} f(t_n) P_0(t_n) = \frac{1}{M}\sum_{n=1}^{M} f(t_n) \\ c_n &= \frac{2}{M}\sum_{n=1}^{M} f(t_n) P_n(t_n) \end{aligned} \tag{12}$$

where $t_n = \cos\frac{(n-0.5)\pi}{M}$. Note that the factors for c_0 and c_n are different. This is a consequence of orthogonality property of Chebyshev polynomials.

For a time series $\{\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, ..., \langle s_N, t_N\rangle\}$, we need to compute the coefficients. The above formulation is restricted to interval functions and in contrast, a trajectory is a discrete function. For a series, we normalize the t into the range [-1,1], we divide this interval into N disjoint intervals(Eq. 13) and we introduce the interval function $f(t)$(Eq. 14).

$$I_i = \begin{cases} \left[-1, \frac{t_1+t_2}{2}\right) & i = 1 \\ \left[\frac{t_{i-1}+t_i}{2}, \frac{t_i+t_{i+1}}{2}\right) & 1 < i < N \\ \left[\frac{t_{N-1}+t_N}{2}, 1\right] & i = N \end{cases} \tag{13}$$

$$f(t) = p_i \qquad if\ \ t \in I_i \tag{14}$$

With this interval function, we can use Eq.12 to compute the coefficients of the Chebyshev approximation.

3.2 Similarity Metric

Previous section shows how to approximate Chebyshev polynomials for a given time series. Our goal is to represent trajectories with Chebyshev approximations and define a similarity metric for comparing those.

Our trajectories are two dimensional and we approximate a polynomial for each dimension. We represent a trajectory $TR_i = \{\langle p_1, t_1\rangle, ..., \langle p_N, t_N\rangle\}$ by two time series; $TR_i^x = \{\langle x_1, t_1\rangle, ..., \langle x_N, t_N\rangle\}$ and $TR_i^y = \{\langle y_1, t_1\rangle, ..., \langle y_N, t_N\rangle\}$. Let C_i^x and C_i^y be Chebyshev coefficients for approximations of these two time series.

We formulate the similarity between two trajectories as

$$Dist_d\left(TR_i, TR_j\right) = \sqrt{\frac{\pi}{2}\sum_{k=0}^{M}\left(C_i^d\left(k\right) - C_j^d\left(k\right)\right)^2} \tag{15}$$

$$Dist\left(TR_i, TR_j\right) = \sqrt{Dist_x^2\left(TR_i, TR_j\right) + Dist_y^2\left(TR_i, TR_j\right)} \tag{16}$$

where d is either x or y, and M is the degree of the approximated polynomials.

4 Querying Interface

We introduce a user interface where user can query the data. User can simply pick a query method and sketch a trajectory on the 2d scene. Join operations also supported for these queries. Another type of querying is done by using existing trajectories. User can pick an existing trajectory and make a similarity-matching query.

4.1 Panels

An overview of the interface and brief descriptions can be found in Fig. 5. There are three main panels; 2d scene, trajectory tree and toolbar.

In querying mode, the 2d scene is the main interface for both input and output. Querying is done by simply picking a query method from the toolbar and sketching a trajectory on the scene as a query constraint. The constraint trajectory is painted with red on the scene. The results are also shown on the scene where they are painted with green with a transparency value depending on the query type and confidence value of the result. For example, in a similarity-matching query, a matching trajectory will be less transparent if it is more similar to the constraint trajectory(Fig. 6).

The trajectory tree, which is the panel on the right, provides navigation and some basic querying capabilities. If a group is picked, all the trajectories in this group is shown on the scene.

The toolbar icons are shown in Table 4.1.

4.2 Querying

We define two main querying constraints. The first one is "matching" and the second one is "intersecting"(Fig. 8a). Join operations for any combination of those constraints can also be done(Fig. 8b,8c,8d).

The "matching" constraint can be defined in three different ways; similarity-matching(Fig. 6), similarity-matching by omitting translation(Fig. 7a) and pattern-matching(Fig. 7b,7c,7d).

The "intersection" constraint is used for finding the trajectories which intersects a given line. For this specific query type, we use MBR-based indexing. This type of querying is done by sketching a line on the scene.

Navigation		Zoom In
		Zoom Out
		Reset Zoom
Pointer Behavior		Move Viewport
		Select Trajectory
		Matching Query
		Intersection Query
Matching Method	R	Similarity(Regular)
	T	Similarity(Omit Translation)
	P	Pattern
Join Operators	&&	And
	||	Or

Table 1: Toolbar icons

5 Conclusions

We have presented a set of techniques for querying camera-based tracking data and built a querying interface using our infrastructure. We showed how to combine the data from different cameras and external sources such as our user interface. We proposed representing trajectories with polynomial approximations for efficient comparison and indexing. We built an interface for querying spatial-data intuitively.

In future work, we will investigate clustering techniques for data streams in sensor networks. Our tracking system[3] is configured as a sensor network and it is critical to have a distributed clustering infrastructure for efficient indexing.

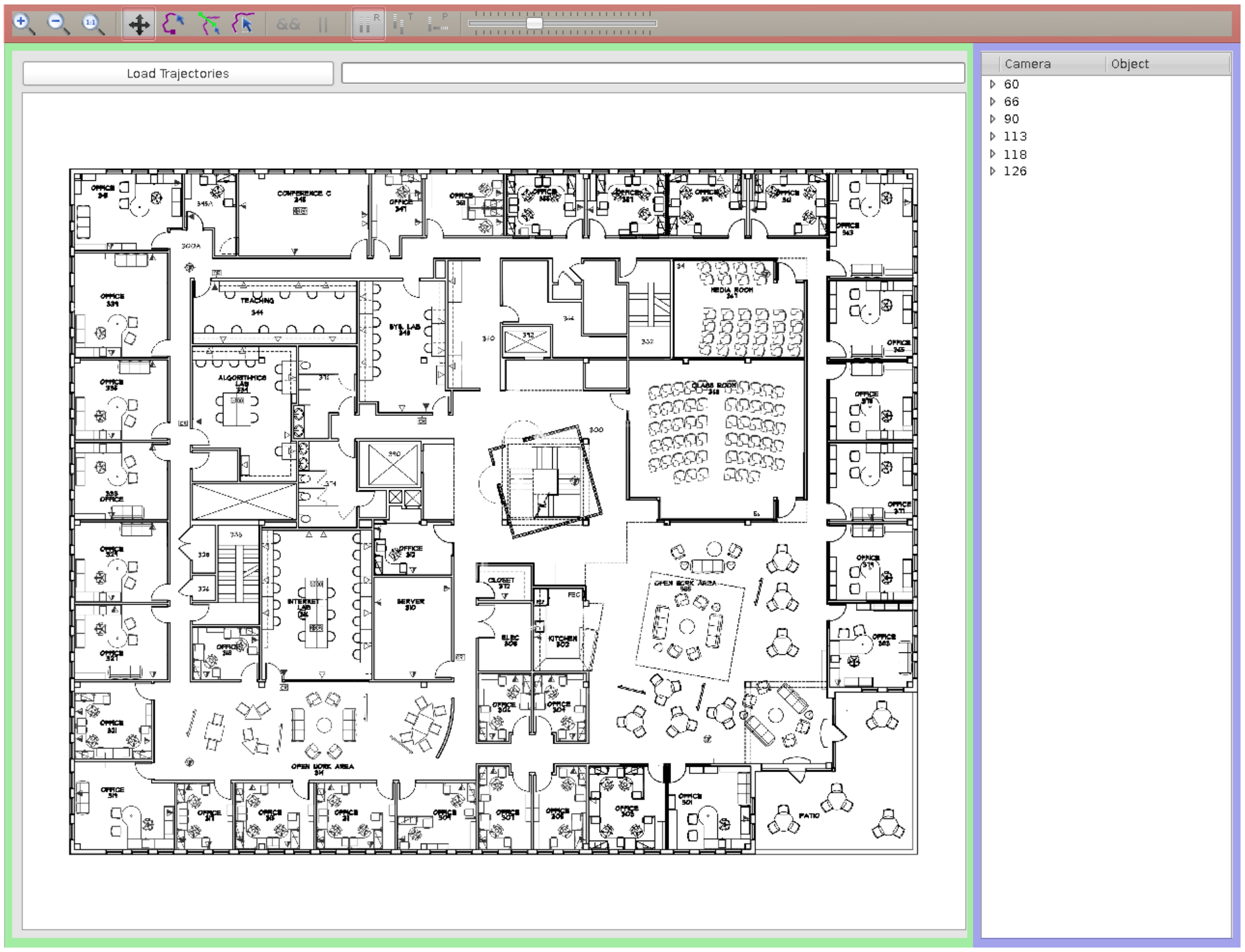


Fig. 5: *Red:* Toolbar; navigation and querying options. *Green:* 2d scene; main interface for both input and output, querying is done by sketching to this panel. *Blue:* Trajectory tree; allows navigation on trajectories. Similarity queries can also be done via this panel.

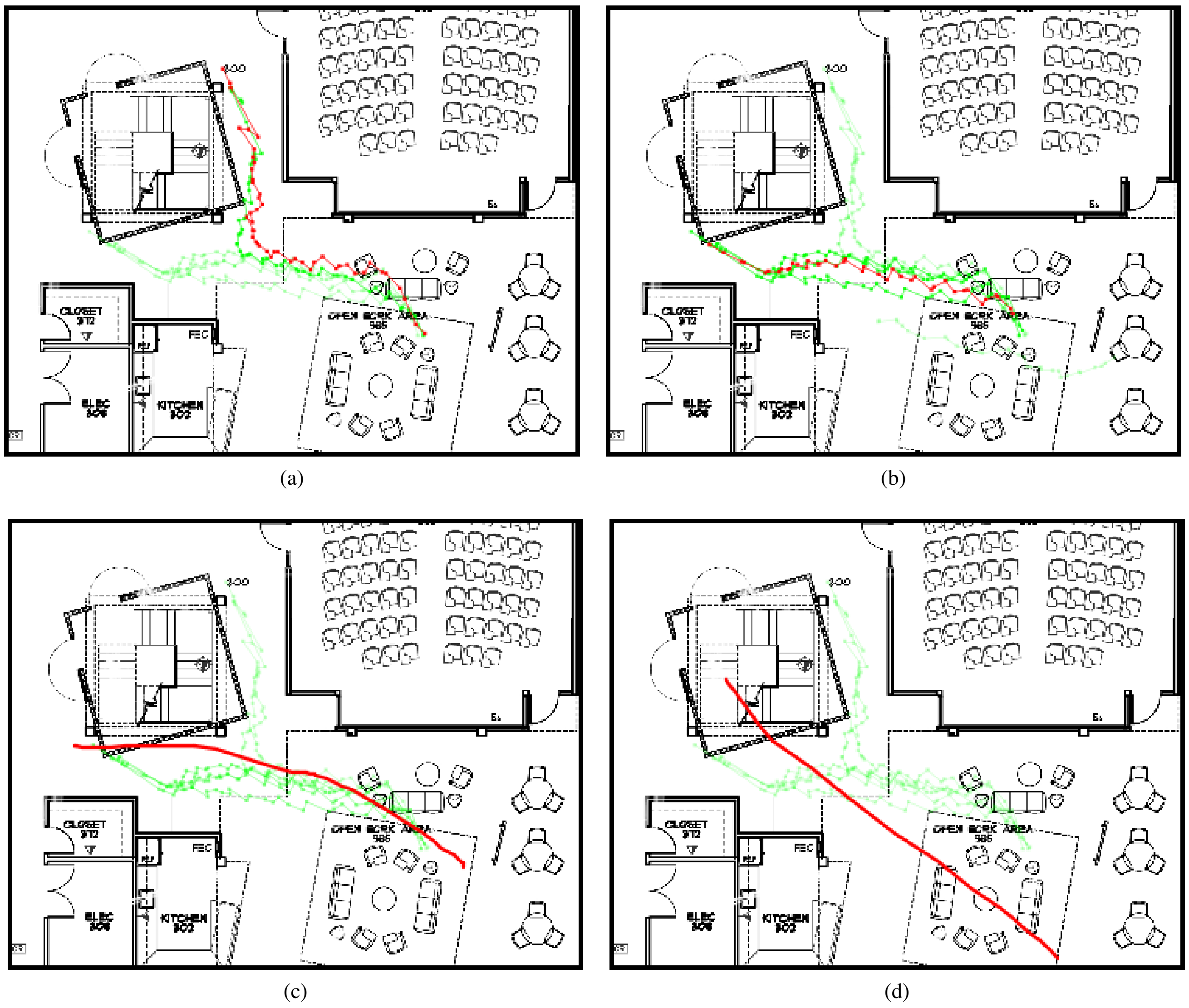


Fig. 6: (a)(b) Examples of similarity-matching queries by using an existing trajectory as constraint(by trajectory tree panel). The transparency represents the similarity distance. (c)(d) Examples of similarity-matching queries by sketching a trajectory on 2d scene.

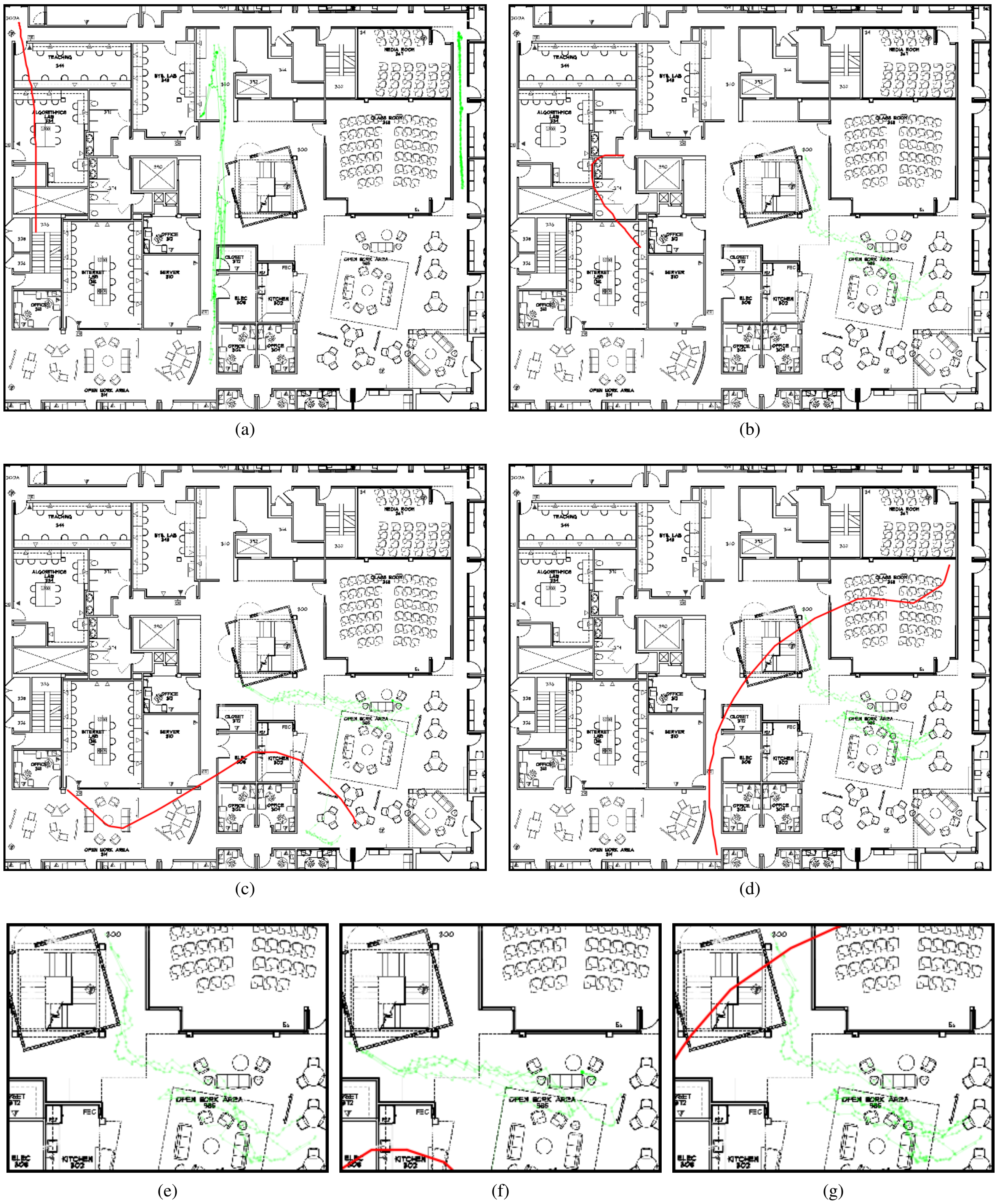
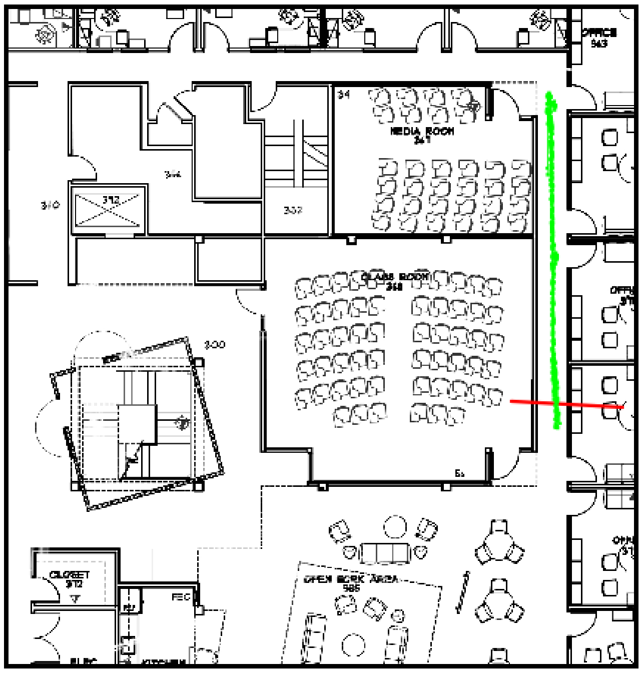
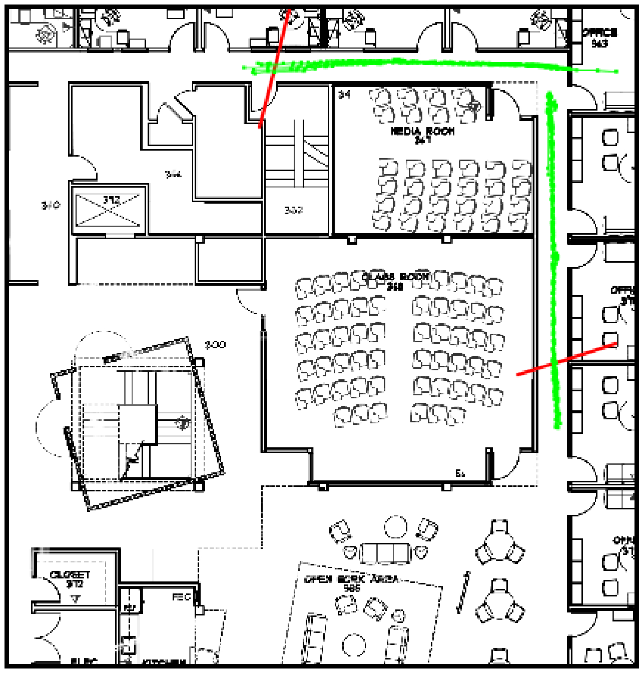
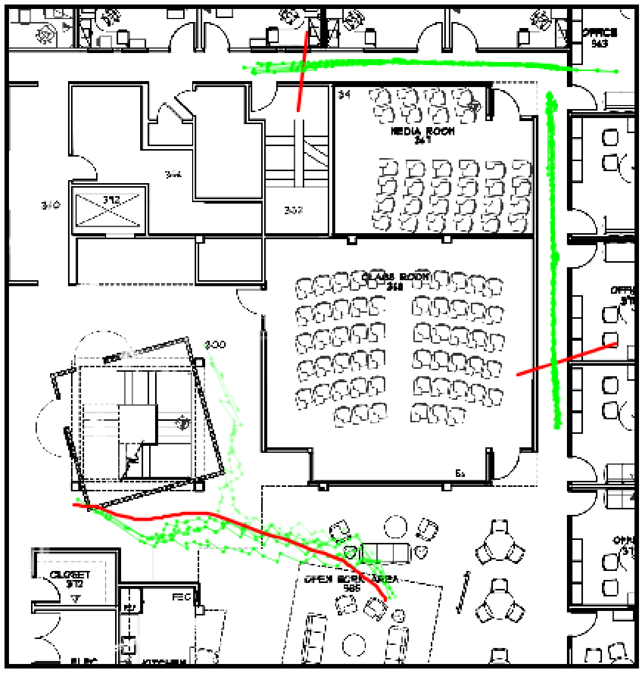


Fig. 7: (a) Omit-translation-similarity-matching query example. (b)(c)(d) Pattern-matching query examples. (e)(f)(g) Zoomed results of pattern-matching queries.


(a)


(b)


(c)

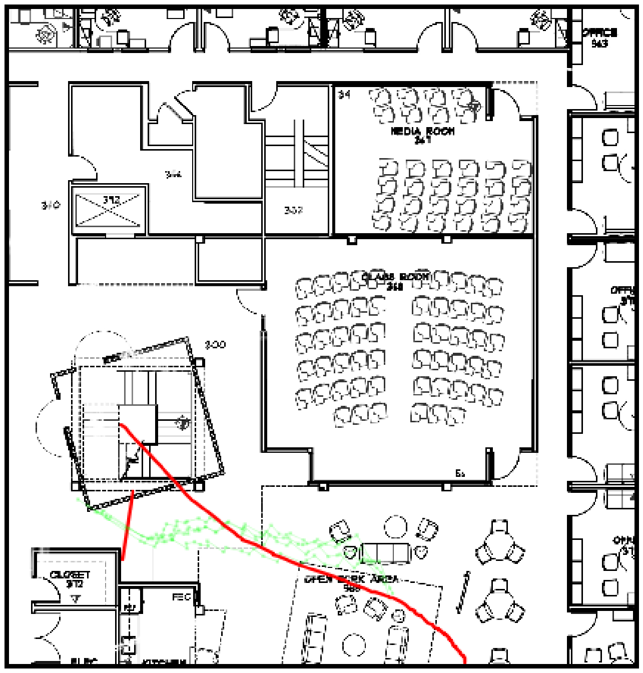

(d)

Fig. 8: (a) An example for intersection query. (b) Join operator(OR) example for two intersection queries. (c) Join operator(OR) example for two intersection queries and a similarity-matching query. (d) Join operator(AND) example for an intersection query and a similarity-matching query.

References

[1] K. Whang J. Lee, J. Han. Trajectory clustering: A partition-and-group framework. *SIGMOD*, 2007.

[2] C. Ravishankar J. Ni. Indexing spatio-temporal trajectories with efficient polynomial approximations. *IEEE Transactions on Knowledge and Data Engineering*, 19, 2007.

[3] D. E. Crispell J. Jannotti J. Mao G. Taubin M. Akdere, U. etintemel. Data-centric visual sensor networks for 3d sensing. *GSN*, pages 131–150, 2006.

[4] D. Gunopulos V.J. Tsotras M. Hadjieleftheriou, G. Kollios. Indexing spatio-temporal archives. *VLDB J.*, 15:143–164, 2006.

[5] V.J. Tsotras D. Gunopulos M. Hadjieleftheriou, G. Kollios. Efficient indexing of spatiotemporal objects. *Proc. Int'l Conf. Extending Database Technology(EDBT '02)*, pages 351–268, 2002.

[6] R. Ng Y. Cai. Indexing spatio-temporal trajectories with chebyshev polynomials. *SIGMOD*, 2004.