Efficient Trajectory Extraction and Parameter Learning for Data-Driven Crowd Simulation

Aniket Bera* Sujeong Kim† Dinesh Manocha‡

The University of North Carolina at Chapel Hill

ABSTRACT

We present a trajectory extraction and behavior-learning algorithm for data-driven crowd simulation. Our formulation is based on incrementally learning pedestrian motion models and behaviors from crowd videos. We combine this learned crowd-simulation model with an online tracker based on particle filtering to compute accurate, smooth pedestrian trajectories. We refine this motion model using an optimization technique to estimate the agents' simulation parameters. We highlight the benefits of our approach for improved data-driven crowd simulation, including crowd replication from videos and merging the behavior of pedestrians from multiple videos. We highlight our algorithm's performance in various test scenarios containing tens of human-like agents.

Keywords: crowd simulation, data-driven, multiple-people tracking

1 INTRODUCTION

Realistic simulation of crowd behavior has many real-world applications, and as a result has been the subject of extensive research. In computer games and animations, realistic crowd simulation enhances the user's experience and perception. In applications like safety analysis and surveillance, crowd simulation offers a model for better understanding of behavior in crowded situations, and allows researchers to model the effects of changes to the situation or environment on crowd behaviors. The overall goal is to model the dynamics and the variety of crowd behaviors: to represent how and when crowd behavior changes.

In general, it is challenging to simulate realistic crowd behaviors.While researchers in various fields like psychology and other social sciences have been studying and observing human behavior for decades, behavior is still not fully understood nor predictable. This is in part because human behaviors are governed by multiple factors. Often, designers or animators take account of the multiple behavioral factors manually, generating scene-specific behavior rules such as events, trajectories, or interactions by hand–a tedious and time-consuming process. In addition, the use of current simulation models (e.g. for local collision avoidance and navigation) involve considerable tweaking or variations of simulation parameters to generate the desired behaviors or trajectories. As the number of agents and number of scenarios increase, so do the diversity of behaviors and the number of interactions in crowds; this increase in complexity makes the hand-adjustment process practically not viable.

Instead of using an explicit crowd-simulation model and behavior rules, data-driven crowd simulation algorithms use examples generated using crowd videos or motion capture data [15, 14]. Once sufficient data on real-world crowd trajectories or behavior data has been collected (including data on various styles of gesture, crowd formations, and interactions between agents), these data-driven methods can handle various scenarios without excessive human intervention.

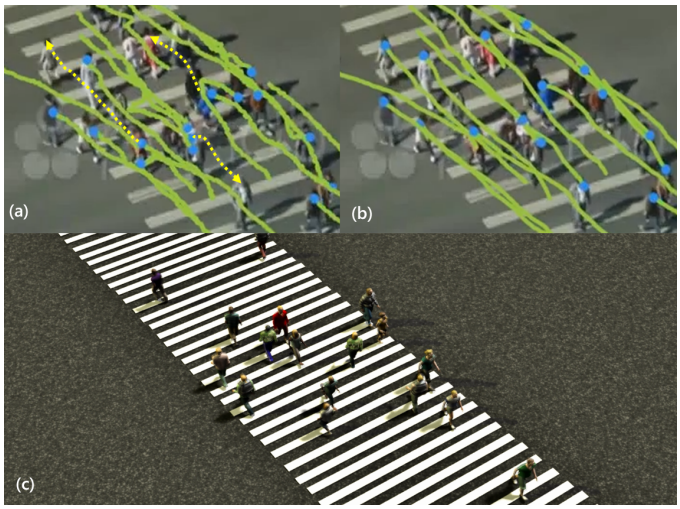


Figure 1: (a) Pedestrian tracking using a simple particle filter and motion model. Yellow trajectories indicate some of the problems with prior tracking results, e.g. missing or incorrect tracks (b) Tracking using our algorithms with improved accuracy and smooth trajectories. (c) Rendering of the pedestrian trajectories from our data-driven crowd simulation system

One of the challenges with data-driven methods is related to collecting the trajectory or behavior data from videos or other sensor data. Advances in real-world capturing technology have resulted in large databases of crowd videos (e.g. YouTube). However, most data-driven crowd simulation systems require manual tracking and/or annotation of environments (e.g. obstacles) and behaviors. These techniques are time consuming, limited to small group interactions, and not scalable to a large number of crowd videos. Furthermore, they are limited to simple scenarios in terms of number of agents or crowd density.

Main Results: We present a trajectory-extraction and behavior-learning algorithm for a data-driven crowd simulation system. The tracking and behavior results can be used to generate trajectories for one or more pedestrians in the simulated environment. Furthermore, we can combine the trajectories extracted from two or more different videos to generate *mixed* data-driven crowd simulation.Our algorithm uses the first few frames to automatically learn the best parameters for each pedestrian, then to dynamically compute a motion model to generate smooth and accurate trajectory for each pedestrian.

There are many benefits and possible applications of our approach. Our tracking algorithm results in improved accuracy; it reduces the number of *id switches* and *lost tracks* and and generates good trajectories that can be used for data-driven simulation. More-

*e-mail:ab@cs.unc.edu
†e-mail:sujeong@cs.unc.edu
‡e-mail:dm@cs.unc.edu

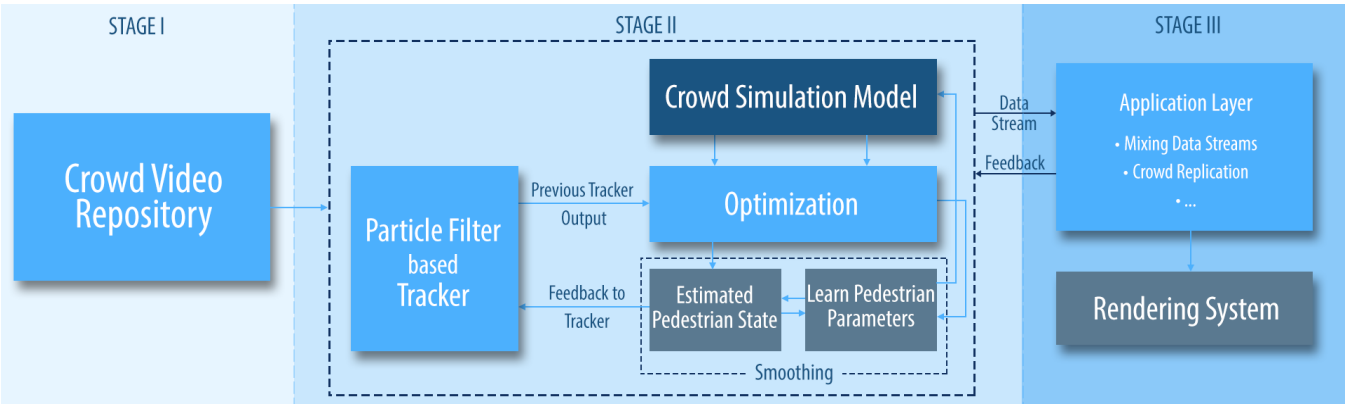


Figure 2: An overview of our Data-Driven Crowd Simulation System. We have a three-stage pipeline. We begin with a selection of crowd videos from the pool of crowd videos available. Next we feed this into our tracking pipeline. We iteratively learn the motion model parameters and use them to improve the tracking. The feedback is bidirectional; the simulation model is re-trained after a fixed number of frames. We iteratively compare prior pedestrian state history (pedestrian position, velocity) and prior tracker inputs to refine parameters and generate smooth trajectories. Lastly, our learned behavior and the model's resulting trajectories are used as input to our application layer, where we use them to replicate the crowd behaviors seen in individual source videos, or to mix agent trajectories from multiple crowd videos.

over, the trajectories our approach generates are smooth and can be directly used in simulated environments. Our method automatically replicates real-world crowd trajectories and behaviors, producing crowd motion similar to that observed in real-world videos. Additionally, our method allows for easy mixing of multiple tracking results, which allows for modeling of more complex scenarios. We demonstrate the performance of our approach using a database of multiple outdoor crowd videos with tens of agents.

The rest of the paper is organized as follows. Section 2 reviews related work in data-driven crowd simulation and tracking. Section 3 introduces our notation and terminology and gives an overview of our approach. We give a general overview of our tracking and learning algorithm in Section 4 and highlight some of its applications using real-world videos in Section 5.

2 Related Work

In this section, we give a brief overview of the prior work on data-driven crowd simulation and pedestrian tracking.

2.1 Data-driven Crowd Simulation

Many techniques have been proposed that attempt to fit motion-model parameters to a given real-world video, and that attempt to evaluate how closely they replicated the original data. Lerner et al. [16] use density-based measures, Guy et al. [9] use entropy-based measure for similarity, Berseth et al. [7] use performance criteria such as minimizing time and effort, Wolinski et al. [30] use optimization techniques to find good parameters for different motion models. Musse et al. [19] suggest a method to learn the intentions from tracked data. They build a desired velocity field extrapolated from low density tracking data. Patil et al. [20] use flow field extracted from an input video or from a sketch to direct the virtual crowds. Li et al. [17] propose a method to populate a virtual scene with crowds by copying and pasting small pieces of real-world data: extracted trajectories from motion-capture data of tens of people interacting with each other in a lab environment.

Although dealing with different applications, Zhang et al. [34] use Extended Kalman Filtering based tracking of real humans and overlay several simulated virtual agents in the video. Ren et al. [21] present a path-planning algorithm for virtual agents inserted in the real-world video by computing a density field from real crowd.

Instead of using real-world data to learn parameters for the motion models, another stream of data-driven crowd simulation techniques use collection of behavior examples, such as motion capture or manually extracted trajectories. These techniques usually require pre-processing in order to collect, analyze, and build the motion database. Lerner et al. [15] build a database of examples from manually extracted trajectories and query the trajectory segment of each pedestrian during the simulation time. Lee et al. [14] propose a method to simulate group behaviors from manually extracted trajectories and manually annotated behavior examples. Ju et al. [10] further extend the technique by building a formation model from tracked and synthesized trajectory data and using this model to synthesize virtual crowd motion.

More recently, methods have been introduced that target the simulation of large-scale crowds. Kapadia et al. [11] propose a data structure for databases that store sequences of motion. Sun et al. [26] propose learning main and background characters in a pre-processing stage; they use different simulation level of details based on the classification. Torrens et al. [27] present a machine learning scheme to train movement behavior using a combination of month-long observed movement data (manually drawn by a trained person) and synthetic data from agent-based simulation.

Our method can be used with these data-driven methods to replace the burden of manual tracking. Our method can also be used as a standalone application which directly uses real-world crowd data without building database of examples.

2.2 Pedestrian Tracking

In this section, we briefly review some prior work on the use of motion models in pedestrian tracking, which has been extensively studied in computer vision and image processing. We refer the reader to some excellent surveys [31, 8, 32].

At a broad level, pedestrian tracking algorithms can be classified as either online or offline trackers. Online trackers use only the present or previous frames for realtime tracking. Zhang et al. [33] propose an approach that uses non-adaptive random projections to model the structure of the image feature space of objects, and Tyagi et al. [28] describe a technique to track pedestrians using multiple cameras. Offline trackers, which use data from future frames as well as current and past data [24, 22], are not useful for interactive applications since they require future-state information.

Many tracking algorithms use different pedestrian motion features; these algorithms are especially useful in crowded scenes, where pedestrians often display similar motion patterns. Song et al. [25] propose an approach that clusters pedestrian trajectories based on the assumption that "persons only appear/disappear at en-

try/exit." Ali et al. [1] present a floor-field based method to determine the probability of motion in densely crowded scenes. Rodriguez et al. [23] learn about crowd motion patterns by extracting global video features from a large collection of public crowd videos. Kratz et al. [13] and Zhao et al. [36] track pedestrians using local motion patterns in dense-crowd videos. While here has been work on using motion models [5, 4, 3, 18] for tracking, the quality (especially the smoothness) of the trajectories is not well suited for data-driven crowd simulation. These methods, all well-suited for modeling motion in dense crowds with few distinct motion patterns, may not work in heterogeneous crowds.

3 Overview

We present a new data-driven technique that can be used to generate crowd simulations based on real-world videos. Our method is built on top of an improved multiple-people tracker. We integrate online smoothing technique with our tracking algorithm so that we can directly use the trajectories for interactive applications (e.g. games, virtual environments) or animations.

One of our goals is to design a pedestrian tracking algorithm that can be directly used for data-driven simulation of dense crowds. Manual tracking has traditionally been used to extract pedestrian trajectories, from which behavior models can be generated. But manual tracking becomes increasingly difficult as the scenario grows more complex (as in the case of large-scale crowds). Automatic tracking could be a good alternative solution, but has not so far been plausible, since the automatic trackers cannot meet the accuracy and quality demands of data-driven crowd simulation. Because of the probabilistic basis of most pedestrian tracking algorithms, human-agent detection is consistently noisy and hence the trajectories computed by current algorithms also tend to be noisy, inaccurate and can have issues with agent orientations (see Figure 4). This leads to incorrect or low-fidelity simulation or rendering of resulting agents in interactive applications. Even state-of-the-art multiple-people trackers can give low accuracy in crowded scenes; seemingly small problems, such as occlusion or changes in illumination, cause large problems in tracking, such as ID switches (when a tracker erroneously targets another pedestrian) or loss of the tracking target. Even though pedestrian tracking is well studied, there are many challenging issues that arise due to the following reasons: restricted visibility due to inter-pedestrian occlusion (one pedestrian blocking another), changes in lighting and pedestrian appearance, and the difficulty of modeling human behavior or the intent of each pedestrian.

We present an improved algorithm that uses recent work in crowd simulation models [29, 30] as the underlying motion prior. This results in improved accuracy and smoother trajectories for data-driven crowd simulation.

3.1 Crowd Simulation Models

In Figure 2, we highlight how the motion model is used to extract the trajectory of each moving pedestrian (Section 4.1). Furthermore, we use the results of prior tracked positions to adaptively learn and refine the simulation parameters (Section 4.2). Once we learn the motion model parameters, we refine our motion-model framework to better match the trajectory of each pedestrian. Using optimization techniques, our approach uses the tracker data to compute and then predict agent trajectories, and finally smooths the trajectory output.

3.2 Notation and Terminology

We use the following notation in the rest of the paper:

- S represents the state (position and velocity) of an arbitrary pedestrian as computed by the tracker.
- X represents the state (position and velocity) of an arbitrary pedestrian inside a crowd motion model.
- m represents the "configured" motion model, and is computed using our approach.
- **bold fonts** are used to represent values for all the pedestrians in the crowd; for example $\mathbf{S}$ represents the states (positions and velocities) of all pedestrians as computed by the tracker
- subscripts are used to indicate time; for example m_t represents the "configured" motion model at timestep t, and $\mathbf{S}_{t-k:t}$ represents all states of all agents for all successive timesteps between $t-k$ and t, as computed by the tracker.

The configured motion model can be specified as follows: $X_{t+1} = m_t(X_t)$ or $\mathbf{X}_{t+1} = m_t(\mathbf{X}_t)$ to compute the motion of one arbitrary pedestrian or all pedestrians, respectively.

Data Representation Our algorithm keeps track of the *state* (i.e. position and velocity) of each pedestrian for the last k timesteps or frames. These are referred to as the k-states of each pedestrian. These k-states are initialized by pre-computing the states from the first k timesteps. The k-states are updated at each timestep by removing the agents' state from the oldest frame and adding the latest tracker-estimated state.

Motion Model is one of several independent simulation models widely used for pedestrian modeling in crowds: In our current system, we choose RVO [29] which is a motion model for local collision avoidance and navigation. We use the motion model to compute the parameters that most closely resemble each agent's past frames. Based on an optimization approach, we first "configure" the motion model to best match the recent k-state data; we then use the configured motion model to make a prediction of the agent's next state.

The tracker is a particle-filter based tracker which uses the motion prior, obtained from the configured motion models, to estimate the agents' next state. This tracker also uses a confidence estimation stage [4] that dynamically computes the number of particles.

4 Multi-person tracking and learning

In this section, we give an overview of our tracking and parameter learning algorithms.

4.1 Tracking Algorithm

Though any online tracker which requires a motion-prior system can be used, we use particle filters as our underlying tracking algorithm. The particle filter is a parametric method which solves non-Gaussian and non-linear state estimation problems [2]. Particle filters are frequently used in object tracking, since they can recover from lost tracks and occlusions. The particle tracker's tracking uncertainty is represented in a Markovian manner by only considering information from present and past frames.

In our formulation, we use the configured motion model m_t, as well as the error Q_t, in the prediction that this motion model has generated. Additionally, the observations of our tracker can be represented by a function $h()$ that projects the state X_t to a previously computed state S_t. We denote the error between the observed states and the ground truth as R_t. We can now phrase them formally in terms of a standard particle filter as below:

$$S_{t+1} = m_t(X_t) + Q_t, \tag{1}$$

$$S_t = h(X_t) + R_t. \tag{2}$$

Particle filtering is a Monte Carlo approximation to the optimal Bayesian filter, which monitors the posterior probability of a first-order Markov process:

$$p(X_t|S_{t-k:t}) = \alpha p(S_t|X_t) \int_{X_{t-1}} p(X_t|X_{t-1}) p(X_{t-1}|S_{t-k:t-1}) dX_{t-1}, \tag{3}$$

where X_t is the process state at time t, S_t is the observation, $S_{t-k:t}$ is all of the observations through time t, $p(X_t|X_{t-1})$ is the process dynamical distribution, $p(S_t,X_t)$ is the observation likelihood distribution, and α is the normalization factor. Since the integral does not have a closed-form solution in most cases, particle filtering approximates the integration using a set of weighted samples $X_t^{(i)}, \pi_t^{(i)}{}_{i=1,\ldots,n}$, where $X_t^{(i)}$ is an instantiation of the process state (known as a particle), and $\pi_t^{(i)}$'s are the corresponding particle weights. With this representation, the Monte Carlo approximation to the Bayesian filtering equation is

$$p(X_t|S_{t-k:t}) \approx \alpha p(S_t|X_t) \sum_{i=1}^{n} \pi_{t-1}^{(i)} p(X_t^{(i)}) p(X_{t-1}^{(i)}), \tag{4}$$

where n refers to the number of particles.

In our formulation, we use the motion model, m_t, to infer dynamic transition, $p(X_t|X_{t-1})$, for particle filtering.

4.1.1 Motion Model

The simple motion models assume that agents will ignore any interactions with other pedestrians, and will instead follow "constant-speed" or "constant-acceleration" paths to their immediate destinations. However, the accuracy of this assumption decreases as crowd density in the environment increases (e.g. to 2-4 pedestrians per square meter). More sophisticated pedestrian motion models, such as RVO and Social Forces, can better model these interactions among pedestrians; they are formulated either in terms of attractive/repulsive forces or collision-avoidance constraints. In our approach we use RVO as the underlying motion model, as it can be more accurate for dense scenarios. Specifically, we use the ORCA algorithm [29] and the following parameters $\mathbf{P}$ and their initial values.

RVO Parameters	min	max	mean
comfort speed (m/s)	1	2	1.5
neighbor distance (m)	2	20	11
radius (m)	0.2	0.8	0.5
agent time horizon (s)	0.1	5	2
obstacle time horizon (s)	0.1	5	2

Given each agent's state at a certain timestep, RVO computes a collision-free state for the next timestep. Each agent is represented as a 2D circle in the plane, and the parameters (used for optimization) for each agent consist of the representative circle's radius, maximum speed, neighbor distance, and time horizon (only future collisions within this time horizon are considered for local interactions).

Let V_{pref} be the preferred velocity for a pedestrian that is based on the immediate goal location. The RVO formulation takes into account the position and velocity of each neighboring pedestrian to compute the new velocity. The velocity of the neighbors is used to formulate the ORCA constraints for local collision avoidance [29]. The computation of the new velocity is solved using linear programming for each pedestrian. If an agent's preferred velocity is not allowed due to the ORCA constraints, that agent chooses the closest velocity that lies in the feasible region:

$$V_{RVO} = \underset{V \notin ORCA}{\arg\max} \|V - V_{pref}\|. \tag{5}$$

More details and mathematical formulations of the ORCA constraints are given in [29].

Formally, at any timestep t, we define the agents' (k+1)-states (as computed by the tracker) $\mathbf{S}_{t-k:t}$:

$$\mathbf{S}_{t-k:t} = \bigcup_{i=t-k}^{t} \mathbf{S}_i. \tag{6}$$

Similarly, a motion model's corresponding computed agents' states $f(\mathbf{S}_{t-k:t}, \mathbf{P})$ can be defined as:

$$f(\mathbf{S}_{t-k:t}, \mathbf{P}) = \bigcup_{i=t-k}^{t} f(\mathbf{X}_i, \mathbf{G}, \mathbf{P}), \tag{7}$$

which are initialized with $\mathbf{X}_{t-k} = \mathbf{S}_{t-k}$ and $\mathbf{G} = \mathbf{S}_t$, where f returns the states obtained with the admissible velocity that is closest to the preferred velocity.

At timestep t, considering the agents' k-states $\mathbf{S}_{t-k:t}$, computed states $f(\mathbf{S}_{t-k:t}, \mathbf{P})$ and a user-defined error metric $error()$, our algorithm computes:

$$\mathbf{P}_t^{opt,f} = \underset{\mathbf{P}}{\operatorname{argmin}}\, error(f(\mathbf{S}_{t-k:t}, \mathbf{P}), \mathbf{S}_{t-k:t}), \tag{8}$$

where $\mathbf{P}_t^{opt,f}$ is the parameter set which, at timestep t, results in the closest match between the states computed by the motion algorithm f and the agents' k-states.

Consequently, the best (as per the error in the $error()$ metric itself) prediction for the agents' next state is obtained from the unconfigured motion model for timestep $t+1$ is:

$$\mathbf{X}_{t+1} = m(\mathbf{S}_t). \tag{9}$$

In our case, we have chosen an error metric that simply computes the average 2-norm between the observed agent positions and the tracker-computed positions. This metric is well adapted to our problem, as the number of considered past frames k is relatively small (around 10). We analyze the impact of larger values of k on the accuracy and running time in Section 4.1.1. This means that it is difficult for the simulated agents to stray too far from the observed trajectories, so this metric can be effectively used to compare the observed and computed trajectories. Formally, this metric is defined at timestep t as follows:

$$error = \sum_{i=t-k}^{t} \|\mathbf{S}_i - \mathbf{X}_i\|. \tag{10}$$

4.2 Incremental Parameter Learning

Motion models usually have several parameters that can be tuned in order to change the agents' behaviors. We assume that each parameter can have a different value for each pedestrian. By changing the value of these parameters, we get some variation in the resulting trajectory prediction algorithm. We use $\mathbf{P}$ to denote all the parameters of all the pedestrians. In our formulation, we denote the resulting parameterized motion model as

$$\mathbf{X}_{t+1} = f(\mathbf{X}_t, \mathbf{G}, \mathbf{P}). \tag{11}$$

Bera et al. [6] had done an extensive study on different optimization techniques but in this paper we use genetic algorithm as the underlying parameter-learning technique, as this algorithm offers the best compromise between learning results and speed.

```
pop ← initialize(P) while true do
    selection(pop) if termination(P, I) then
        stop
    end
    pop ← reproduction(pop)
end
```

- **initialize(P)**: RVO parameters **P** (refer table in Section 4.1.1) are randomly initialized in accordance with the mean distribution for each parameter.
- **selection()**: Pedestrians are matched using Equation (10) and divided into 3 groups (*Best*, *Middle* and *Worst*) depending on how well they match the input trajectory.
- **termination(P,** I**)**: The algorithm is terminated if there is very little or no improvement in the optimization value after I iterations.
- **reproduction()**: based on which group it belongs to, a parameter set is attributed three probabilities: α, β and γ. For each parameter of this individual, α decides if the value is changed or not, β decides if the value is changed by crossover or mutation and, finally, γ decides which type of mutation is done.
- **crossover**: a crossover is performed by copying a value from an individual belonging to the Best group.
- **mutation**: a mutation is performed by picking a new value at random based on either the base distribution or the current real distribution of an individual from the Best group (according to γ).

At each iteration, this algorithm evaluates and ranks all possible parameter sets (solutions). If there is no significant improvement after a certain number of iterations, the process is terminated.

4.3 Smooth Trajectories

Most pedestrian-tracking algorithms are probabilistic. As a result, they are inherently noisy, and can generate noisy trajectories (see Fig. 4). However, our method uses an improved crowd motion motion for local interaction, navigation and collision-free motion, and the underlying smoothness of RVO-based motion model ensures that the agents exhibit no oscillatory behaviors. And unlike prior RVO-based tracking approach [4], our method also uses an optimization scheme to incrementally learn the pedestrian parameters and use them to refine the trajectories, which also results in smoother trajectories than do prior methods. (see Fig. 4).

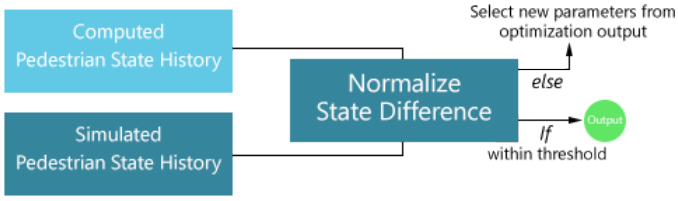


Figure 3: Our online trajectory-smoothing pipeline. We compare a short history of previous and the present pedestrian computed and simulated states; after normalizing the difference, we compare it with a threshold value. If the difference is high, we compute new parameters using the optimization algorithm, then re-compute the state.

5 Results

In this section we apply our tracking and parameter-learning algorithms for data-driven crowd simulation. We demonstrate their performance for crowd replication as well as a few applications.

5.1 Applications

We use our improved tracking algorithm for two applications: crowd replication and mixing crowd streams.

5.1.1 Crowd Replication

One important application of data-driven crowd simulation is crowd replication. The goal is to faithfully reproduce real human crowd trajectories or movements using virtual characters in a simulation.

Figure 6: Mixing Crowd Streams: The agents on the brown (left) and blue (central) floors exhibit varied behaviors, generated from different videos. The final mixed video (green floor) has behaviors combined from both the video streams. Pedestrians marked with brown are from the left video, and those marked blue are from the central video. The overall mixing algorithm uses the two sets of extracted trajectories and performs local collision avoidance between them.

Crowd replication is used extensively in movies and games, but in almost all cases, the pedestrian trajectories are either drawn manually or the tracked data is subject to extensive post-production work to make it usable.

Some of the problems which beset pedestrian-tracking algorithms include trajectory noise, incorrect pedestrian orientation (pedestrian orientation is computed using past states but due to noise, this computation can be incorrect leading to frequent orientation changes), and ID switches; our algorithm produces smooth trajectories (which also resolves orientation issues), and displays fewer ID switches and higher accuracy overall.

We feed the smooth trajectories from our pipeline to *Golaem*, a commercially available crowd-rendering platform.

5.1.2 Mixing Crowd Streams

Another application is to combine the pedestrian trajectories extracted from two or more different videos. Each video may only capture some aspects of crowd behavior, and we want to combine them for an application. In this case, our mixing algorithm takes as input the smooth trajectories extracted by our algorithm from two videos (those shown on the left and middle figures in Fig. 6). The mixing algorithm [35] uses the two sets of extracted trajectories and performs a simple local collision avoidance between them.

5.2 Analysis

We first show a quantitative and qualitative analysis of our method on four different real crowd videos (See Fig. 8). We highlight the application to crowd replication (See Fig. 5) using the extracted trajectories. Finally, we show mixing trajectories from multiple videos and creating more complex scenarios (See Fig. 7).

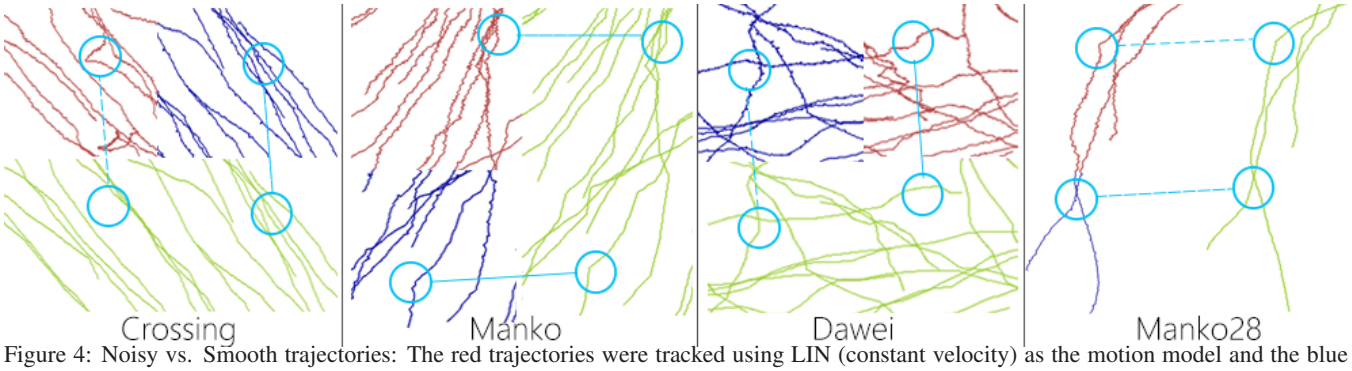


Figure 4: Noisy vs. Smooth trajectories: The red trajectories were tracked using LIN (constant velocity) as the motion model and the blue trajectories are results from prior tracking algorithm using RVO [4]. We display the improved trajectories extracted by our algorithm in green, which are smoother. Our motion model iteratively refines pedestrian behavior and produces smooth trajectories. The blue circles highlight the improvement using our method. (For clarity, these trajectories are just a cropped section of the entire scene.)

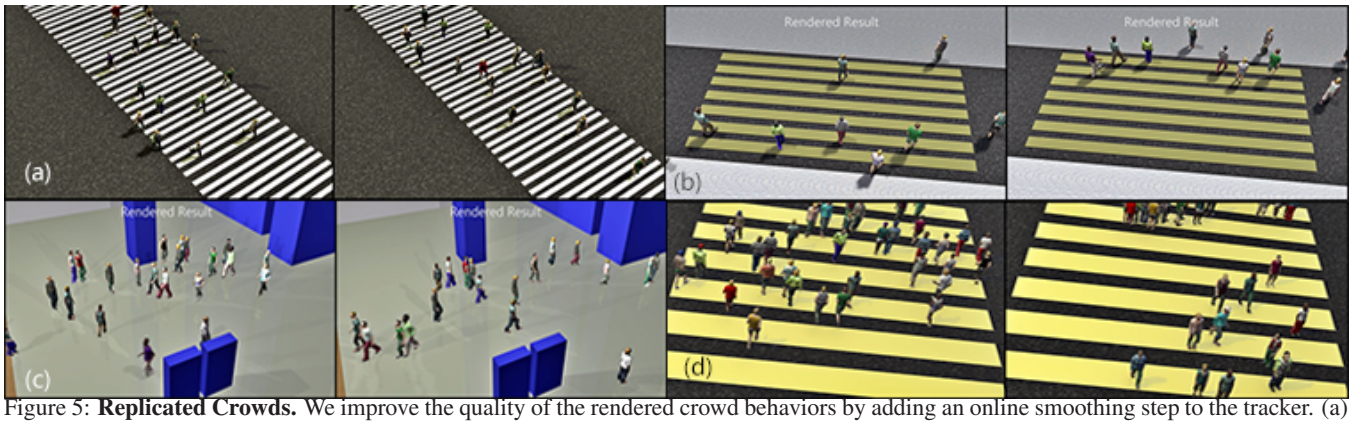
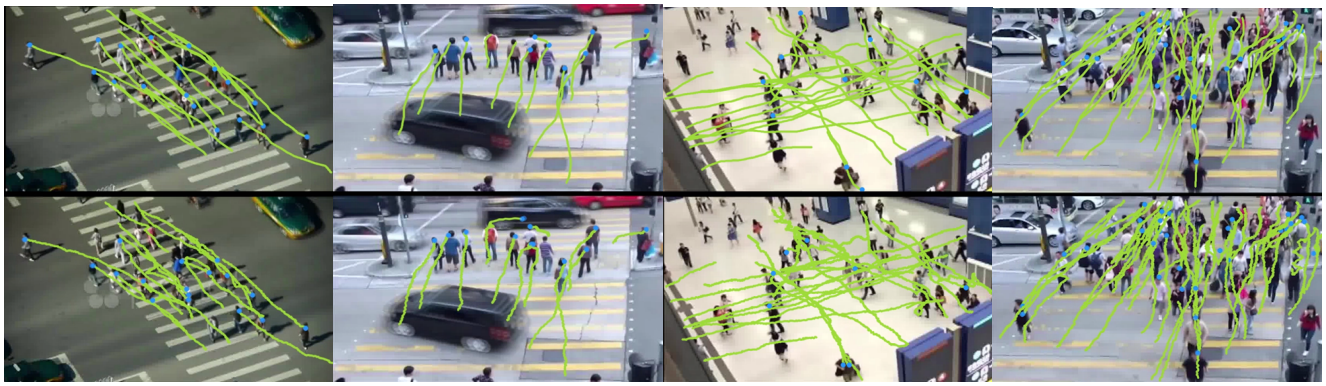


Figure 5: **Replicated Crowds.** We improve the quality of the rendered crowd behaviors by adding an online smoothing step to the tracker. (a) to (d) show the replicated crowds rendered directly using the tracking results without any pre-processing; these correspond to each benchmark video.

(a) Pedestrian trajectories extracted from two different videos

(b) Mixed trajectories

Figure 7: **Mixing trajectories from multiple input videos.** We can use multiple videos as inputs to a single, more complex scenario. In the left two videos of (a), pedestrians move in an image-space from bottom to top or from top to bottom. In the right two videos of (a), pedestrians move in a uniform direction: right to left in a slightly tilted way in the image-space. There are, therefore, three different main directions of pedestrian movements. (b) Using these trajectories with agent-based simulation methods, we can generate crowds with these three different flows, while still achieving local collision avoidance between the agents.



(a) Crossing (b) Manko28 (c) Dawei (d) Manko

Figure 8: **Comparison of improved tracking (our method, above) to prior methods based on particle filter + LIN (below).** We compare the quality of the extracted trajectories on four different real crowd videos. As compared to prior methods, our approach results in smoother trajectories and improved accuracy for each benchmark. Our method runs at interactive rates (24-26fps)

Dataset	Characteristics	Density	Agents
Crossing	BV, PO, IC	Medium	34
Dawei	BV, PO, IC, CO	Medium	72
Manko	BV, PO, IC, CO	High	81
Manko28	BV, PO, IC	Low	16

Table 1: We highlight the number of human-like agents and many other characteristics of crowds in these video datasets. In particular, handling partial or complete occlusion can be challenging for manual or automatic trajectory computation algorithms. We use the following abbreviations about the underlying scene: Background Variations(BV), Partial Occlusion(PO), Complete Occlusion(CO), Illumination Changes(IC)

		Crossing	Dawei	Manko	Manko28
LIN	MOTP	67.3%	70.1%	71.6%	68.9%
	MOTA	48.0%	39.8%	50.1%	47.2%
Our Approach	MOTP	71.9%	73.1%	77.9%	74.5%
	MOTA	51.9%	55.3%	59.0%	57.1%

Table 2: We compare the MOTA and MOTP for the different video datasets.

	***k* = 5**		***k* = 10**		***k* = 15**		***k* = 20**	
	ST%	FPS	**ST%**	**FPS**	ST%	FPS	ST%	FPS
Crossing	69.8	30	**71.9**	**27**	71.9	19	71.9	8
Dawei	71.8	31	**73.1**	**28**	73.2	20	73.2	7
Manko	76.1	29	**77.9**	**26**	77.9	18	78.0	7
Manko28	73.8	28	**74.5**	**26**	74.6	19	74.6	7

Table 3: We compare the percentage of successful tracks (ST) and average tracking frames per second (FPS) of our approach at different values of k. We find that the most optimal value for k while still maintaining reatime performance is 10

We tested these algorithms on an Intel©Haswell, Core®i7-4771 Processor (8 Cores) with an 8MB Cache, 3.90 GHz and Intel©HD Graphics 4600. Our algorithm is implemented in C++, and some components use OpenMP and OpenCL to exploit multiple cores. We adopted an agent-level parallelism: individual pedestrian computations are distributed across the CPU cores (except for the motion-model computations, where pedestrian behavior is interlinked and tasks are highly sequential).

5.2.1 Quantitative Comparison

We use the **CLEAR MOT** [12] evaluation metrics to analyze the performance of our crowd-tracking algorithms. We use the **MOTP** and the **MOTA** metrics. **MOTP** evaluates the alignment of tracks with the ground truth while **MOTA** produces a score based on the amount of false positives, missed detections, and identity switches. These metrics are considered standard for evaluation for detection and tracking algorithms in computer vision.

We begin by analyzing the performance of our tracking algorithm. We use four publicly available crowd videos: Crossing, Dawei, Manko and Manko28. These videos have medium-density crowd with varying pedestrian and crowd behaviors. Table 1 highlights some of the challenging characteristics with respect to these videos as well as crowd densities. We analyze the MOTA and MOTP metrics across the density groups and the different motion models (Table 2). We also analyze how varying k affects performance and accuracy. We see that (Refer Equation 8) as k increases, it adversely affects our runtime performance with a negligible gain in accuracy after $k = 10$.

5.2.2 Qualitative Comparison

We show that our smoothing algorithm results in improved trajectories for data-driven crowd simulation. In Fig. 8, we can see that the trajectories in the figures in the top row are much smoother than those denoted by the figures in the bottom row. By adding an online smoothing step to our tracker, we improve several common rendering problems: The noisy trajectories that produce jittery motions also tend to change the agent orientation, which causes problems for data-driven crowd simulation and rendering systems. Fig. 5 shows snapshots of the crowds replicated directly using the tracking results. In the supplementary video, each agent's smooth trajectory motion can be observed.

5.2.3 Complex Scenarios

We can use multiple videos as inputs to a single, more complex scenario. Fig. 7 shows snapshots of an example mixed-trajectory scenario. Fig. 7 (a) shows two video inputs from different parts of a shopping mall, each with a different pedestrian flow. In the left two video frames, shown in Fig. 8(a), pedestrians move bottom to top or vice versa. In the right two videos of (a), pedestrians move uniformly from right to left on a slight diagonal in the image-space. As results, there are three different main directions of pedestrian movements. By combining these trajectories with agent-based sim-

ulation methods, we can generate crowds that retain these three different flows and still achieve local collision avoidance between the agents (See Fig. 7).

6 Conclusion, Limitations and Future Work

We present an improved, more accurate pedestrian tracking algorithms for data-driven crowd simulation. Our approach improves on existing tracking algorithms by integrating with an RVO based multi-agent motion model; it iteratively learns crowd behavior using an optimization strategy, which improves the smoothness and accuracy of the resulting tracker and final simulations. We can thus simultaneously improve accuracy and produce smoother trajectories. We demonstrate the benefits of these improvements on real videos in two applications, crowd replication and crowd mixing.

Our approach has some limitations. For videos without camera information about perspective transforms, our method does not perform well, because those properties must be manually estimated, a process that is susceptible to errors. Errors in the motion model computation itself can impact accuracy. In practice, the performance of our algorithm can based on various attributes of the video stream like crowd density, video quality etc.

There are many avenues for future work. We would like to evaluate our approach on other crowd scenarios that have different conditions, such as varying density or illumination. Our ultimate goal is to develop a system that can automatically extract the trajectories or behaviors from crowd videos such as the free, publicly-available, and enormous YouTube database. The performance of our approach can be further improved by exploiting the parallel capabilities of current systems to maximize data parallelism. Finally, we would like to explore techniques that can combine data-driven crowd simulation algorithms with local navigation models.

7 Acknowledgements

This work was supported by NSF awards 1000579, 1117127, 1305286, Intel and a grant from The Boeing Company.

References

[1] S. Ali and M. Shah. Floor fields for tracking in high density crowd scenes. In *ECCV*, pages 1–14. 2008.

[2] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on*, pages 174–188, 2002.

[3] A. Bera, N. Galoppo, D. Sharlet, A. Lake, and D. Manocha. Adapt: real-time adaptive pedestrian tracking for crowded scenes. In *Proceedings of Conference on Robotics and Automation, Hong Kong*, 2014.

[4] A. Bera and D. Manocha. Realtime multilevel crowd tracking using reciprocal velocity obstacles. In *Proceedings of Conference on Pattern Recognition, Sweden*, 2014.

[5] A. Bera and D. Manocha. Reach - realtime crowd tracking using a hybrid motion model. In *Proceedings of Conference on Robotics and Automation, Seattle (To Appear)*, 2015.

[6] A. Bera, D. Wolinski, J. Pettré, and D. Manocha. Real-time crowd tracking using parameter optimized mixture of motion models. *arXiv preprint arXiv:1409.4481*, 2014.

[7] G. Berseth, M. Kapadia, B. Haworth, and P. Faloutsos. Steerfit: Automated parameter fitting for steering algorithms. In *Proceedings of the 2014 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2014.

[8] M. Enzweiler and D. M. Gavrila. Monocular pedestrian detection: Survey and experiments. *PAMI*, pages 2179–2195, 2009.

[9] S. J. Guy, J. van den Berg, W. Liu, R. Lau, M. C. Lin, and D. Manocha. A statistical similarity measure for aggregate crowd dynamics. *ACM Trans. Graph.*, 31(6):190:1–190:11, Nov. 2012.

[10] E. Ju, M. G. Choi, M. Park, J. Lee, K. H. Lee, and S. Takahashi. Morphable crowds. *ACM Trans. Graph.*, 29(6):140:1–140:10, Dec. 2010.

[11] M. Kapadia, I.-k. Chiang, T. Thomas, N. I. Badler, and J. T. Kider, Jr. Efficient motion retrieval in large motion databases. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 19–28, 2013.

[12] B. Keni and S. Rainer. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008, 2008.

[13] L. Kratz and K. Nishino. Going with the flow: pedestrian efficiency in crowded scenes. In *ECCV*, pages 558–572. 2012.

[14] K. H. Lee, M. G. Choi, Q. Hong, and J. Lee. Group behavior from video: A data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, pages 109–118, 2007.

[15] A. Lerner, Y. Chrysanthou, and D. Lischinski. Crowds by example. *Computer Graphics Forum*, 26(3):655–664, 2007.

[16] A. Lerner, Y. Chrysanthou, A. Shamir, and D. Cohen-Or. Data driven evaluation of crowds. In *MIG*, pages 75–83, 2009.

[17] Y. Li, M. Christie, O. Siret, R. Kulpa, and J. Pettré. Cloning crowd motions. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '12, pages 201–210, 2012.

[18] W. Liu, A. B. Chan, R. W. Lau, and D. Manocha. Leveraging long-term predictions and online-learning in agent-based multiple person tracking. *arXiv preprint arXiv:1402.2016*, 2014.

[19] S. R. Musse, C. R. Jung, J. C. S. Jacques, and A. Braun. Using computer vision to simulate the motion of virtual agents. *Computer Animation and Virtual Worlds*, 18(2):83–93, 2007.

[20] S. Patil, J. van den Berg, S. Curtis, M. Lin, and D. Manocha. Directing crowd simulations using navigation fields. *Visualization and Computer Graphics, IEEE Transactions on*, 17(2):244 –254, feb. 2011.

[21] Z. Ren, W. Gai, F. Zhong, J. Pettr, and Q. Peng. Inserting virtual pedestrians into pedestrian groups video with behavior consistency. *The Visual Computer*, 29(9):927–936, 2013.

[22] M. Rodriguez, I. Laptev, J. Sivic, and J.-Y. Audibert. Density-aware person detection and tracking in crowds. In *ICCV*, pages 2423–2430, 2011.

[23] M. Rodriguez and J. e. a. Sivic. Data-driven crowd analysis in videos. In *ICCV*, pages 1235–1242, 2011.

[24] P. Sharma, C. Huang, and R. Nevatia. Unsupervised incremental learning for improved object detection in a video. In *CVPR*, pages 3298–3305, 2012.

[25] X. Song, X. Shao, Q. Zhang, R. Shibasaki, H. Zhao, J. Cui, and H. Zha. A fully online and unsupervised system for large and high-density area surveillance: Tracking, semantic scene learning and abnormality detection. *TIST*, 2013.

[26] L. Sun, X. Li, and W. Qin. Simulating realistic crowd based on agent trajectories. *Computer Animation and Virtual Worlds*, 24(3-4):165–172, 2013.

[27] P. Torrens, X. Li, and W. A. Griffin. Building agent-based walking models by machine-learning on diverse databases of space-time trajectory samples. *Transactions in GIS*, 15:67–94, 2011.

[28] A. Tyagi and J. W. Davis. A context-based tracker switching framework. In *WMVC*, pages 1–8, 2008.

[29] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Robotics Research*. 2011.

[30] D. Wolinski, S. Guy, A.-H. Olivier, M. Lin, D. Manocha, and J. Pettré. Parameter estimation and comparative evaluation of crowd simulations. In *Computer Graphics Forum*, volume 33, pages 303–312, 2014.

[31] Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. pages 2411–2418, 2013.

[32] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *Acm Computing Surveys (CSUR)*, 2006.

[33] K. Zhang, L. Zhang, and M.-H. Yang. Real-time compressive tracking. In *ECCV*, pages 864–877. 2012.

[34] Y. Zhang, J. Pettr, J. Ondej, X. Qin, Q. Peng, and S. Donikian. Online inserting virtual characters into dynamic videoscenes. *Computer Animation and Virtual Worlds*, 22(6):499–510, 2011.

[35] Y. Zhang, X. Qin, J. Pettre, Q. Peng, et al. Real-time inserting virtual characters into dynamic video scene. In *Proceedings of the Chinese Conference on Computer Graphics (CHINAGRAPH 2010)*, 2010.

[36] X. Zhao, D. Gong, and G. Medioni. Tracking using motion patterns for very crowded scenes. In *ECCV*, pages 315–328. 2012.