2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

AdaPT: Real-time Adaptive Pedestrian Tracking for crowded scenes

Aniket Bera[1], Nico Galoppo[2], Dillon Sharlet[2], Adam Lake[2] & Dinesh Manocha[1]

http://gamma.cs.unc.edu/AdaPT

***Abstract*— We present a novel realtime algorithm to compute the trajectory of each pedestrian in a crowded scene. Our formulation is based on an adaptive scheme that uses a combination of deterministic and probabilistic trackers to achieve high accuracy and efficiency simultaneously. Furthermore, we integrate it with a multi-agent motion model and local interaction scheme to accurately compute the trajectory of each pedestrian. We highlight the performance and benefits of our algorithm on well-known datasets with tens of pedestrians.**

I. INTRODUCTION

Pedestrian tracking is a well-studied problem in robotics and related areas [37], [42]. Given a sequence of frames corresponding to moving crowds, the goal is to extract the trajectory of each pedestrian. As autonomous robots are increasingly used in the physical world inhabited by humans, it becomes more and more important to *detect*, *track*, and *predict* the actions of pedestrians [29], [47]. We need real-time pedestrian detection capabilities for collision-free navigation in dynamic environments. The pedestrian trajectories are used to predict future locations of people in order to compute appropriate routes for the robots, such as autonomous cars, mobile surveillance systems, wheelchairs, and museum guides.

The problem of tracking objects and pedestrians has been studied for almost two decades and remains a major challenge for crowded scenes [12]. Pedestrian tracking remains a challenge in part because pedestrians tend to change their speed to avoid collisions with obstacles and other pedestrians. Also, there is a large variability in their appearance and illumination, which makes it hard for color-based template tracking algorithms to consistently track them. Finally, in crowded scenes, the pairwise interactions between pedestrians can increase at a super-linear rate. Many approaches have been proposed for online pedestrian tracking [6], [16], [21], [22], but they can't provide realtime performance as the number and density of pedestrians in the scene increase. Many other fast trackers have been proposed [8], [32], but they only provide good accuracy in some scenes. It is also important to combine these trackers with good models for pairwise interactions and local collision-avoidance behavior to improve their performance in crowded scenes [7], [27], [43].

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

[1]Aniket Bera & Dinesh Manocha are with the Department of Computer Science, University of North Carolina at Chapel Hill. {ab, dm}@cs.unc.edu

[2]Nico Galoppo, Dillon Sharlet and Adam Lake are with Intel Corporation. {nico.galoppo, dillon.sharlet, adam.t.lake}@intel.com

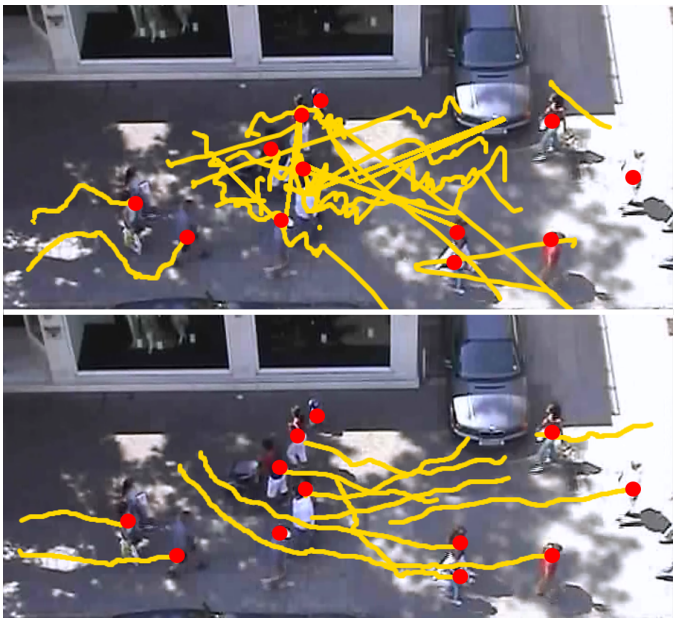

Fig. 1: *Comparing AdaPT (bottom) with the Mean-shift tracker [15] (top). AdaPT's frame rate in this scene is* 27 *fps, as compared to* 31 *fps for mean-shift tracker. However, our algorithm is able to accurately compute all the trajectories, shown as yellow lines. Red circles represent ground truth. (Dataset - zara01 [20])*

Main Results: We present a novel real-time pedestrian tracking algorithm (AdaPT) that is targeted for low to medium-density moving crowds. A key component of our approach is an adaptive tracking algorithm, which automatically combines two or more low-level tracking algorithms (e.g. deterministic and probabilistic trackers) to simultaneously achieve high accuracy and efficiency. To estimate its reliability, the algorithm keeps track of the state of each pedestrian and its interactions with nearby obstacles and other pedestrians. Furthermore, we integrate our adaptive tracking algorithm with a multi-agent trajectory-estimation and collision avoidance-algorithm (e.g. social forces). Our algorithm automatically learns the best parameters for each pedestrian over the first few frames and dynamically adapts them as the low-level trackers and the motion model work together to compute the trajectory of each pedestrian.

Our current implementation of AdaPT uses an adaptive combination of mean-shift and particle trackers; we evaluate its performance on well-known benchmarks, including

(a) Online Boosting (b) Mean-shift (c) AdaPT

Fig. 2: *Performance comparison of AdaPT with other algorithms on a crowded scene: (a) Online Boosting [4](8 fps); (b) MeanShift algorithm (31 fps); (c) AdaPT (27 fps). (Dataset - INRIA 879-38 I [31])*

"BIWI Walking pedestrians" dataset [27] and "Crowds by Example" dataset [20]. Our algorithm can track tens of pedestrians at realtime rates (more than 25fps) on a multi-core PC. Compared to prior realtime algorithms, our adaptive scheme exhibits higher accuracy (Figure 2) or improved performance (Figure 2).

The rest of the paper is organized as follows. Section II reviews related work in tracking and motion models. Section III gives a high-level overview of our approach. Section IV describes our current implementation, and we highlight its performance on different benchmarks in Section V.

II. Related Work

In this section, we briefly review some prior work on tracking methods and motion models. We refer the reader also to a few excellent surveys [10], [41], [44].

At a broad level, pedestrian tracking algorithms can be classified as either online or offline: online trackers use only the present or previous frames while offline trackers also use data from future frames. Zhang et al. [45] proposed an approach that uses non-adaptive random projections to model the structure of the image feature space of objects. Oron et al. [25] presented an algorithm to estimate the amount of local deformation in rigid or deformable objects. The color-based probabilistic tracking algorithm proposed by Perez et al. [28] is fast but prone to loss of trajectories from occlusion. Collins's algorithm [8] tracks blob via mean-shifts and is widely used. Jia et al. [14] proposed a method to track objects using a local sparse appearance model. Tyagi et al. [38] proposed a method to track people using multiple cameras.

Some tracking algorithms methods are offline. The algorithm proposed by Sharma et al. [34] performs offline pedestrian tracking using an unsupervised multiple-instance learning based solution. Rodriguez et al. [30] presented a novel method for tracking pedestrians offline by exploiting global information like crowd density and scene geometry. However, these methods, which require future-state information, are not useful for real-time applications.

Tracking algorithms can also be classified as deterministic or probabilistic trackers based on their underlying search mechanisms. Deterministic trackers iteratively attempt to search for the local maxima of a similarity measure between the target candidate (the location of the pedestrian in a frame) and the object model (the initial state of the pedestrian). The most commonly used deterministic trackers are the mean-shift algorithm and the Kanade-Lucas-Tomasi algorithm.

In probabilistic trackers, the movement of the object is modeled based on its underlying dynamics. Two well-known probabilistic trackers are the Kalman filter and the particle filter. Particle filters are more frequently used than Kalman filters in pedestrian tracking, since the particle filters are multi-modal and can represent any shape using a discrete probability distribution.

Many pedestrian-tracking algorithms have been used in robotics-related applications. Ess et al. [11] and Satake et al. [33] proposed pedestrian tracking algorithms using stereo cameras. Kobilarov et al. [17] described an approach to track and follow pedestrians using a mobile robot. Luber et al. [24] presented a novel method to track pedestrians using RGB-D data based on sets of boosted features.

Motion Models: Many pedestrian-tracking algorithms improve their accuracy by using motion models to predict pedestrian trajectories. In some algorithms' treatment of crowded scenes, different pedestrian trajectories are assumed to have a similar motion pattern. Song et al. [36] proposed an approach that clusters pedestrian trajectories based on the notion that "persons only appear/disappear at entry/exit." Ali et al. [2] presented a floor-field based method to determine the probability of motion in highly dense crowded scenes. Rodriguez et al. [31] used a large collection of public crowd videos to learn crowd motion patterns by extracting global video features. These methods are well suited only for modeling the motion of dense crowds that contain few distinct motion patterns. One of the most popular motion models is the Social Force model by Helbing et al. [13]. In this archetypal pedestrian-tracking model, pedestrians are modeled as particles that are affected by repulsive and attractive forces [43]. Other models include LTA (Linear Trajectory Avoidance) [7], LIN (constant speed model), SBCM (Spatial behavior cognition model) [27], or RVO (Reciprocal Velocity Obstacle) [39].

There have been hybrid approaches combining different motion models and/ or different trackers [3], [5], [18], [19], [23], [38], [46] but most of these methods are either

very compute intensive for realtime processing or have low accuracy.

III. Adaptive Tracking Algorithm

In this section we formally define the notation being used and give a high level overview of AdaPT.

A. Notation

We use the symbol LT_n to represent the n^{th} low-level tracking algorithm. $\text{TJ}_n^j(t)$ represents the trajectory of the j^{th} pedestrian computed using the n^{th} low-level tracking algorithm at time t. $S^j(t)$ is the state of the pedestrian at time t and corresponds to $S^j(t) = (S_p^j(t), S_v^j(t))$, where $S_p^j(t)$ and $S_v^j(t)$ are the *positional* and *velocity* component of the pedestrian state at time t, respectively.

B. Algorithm

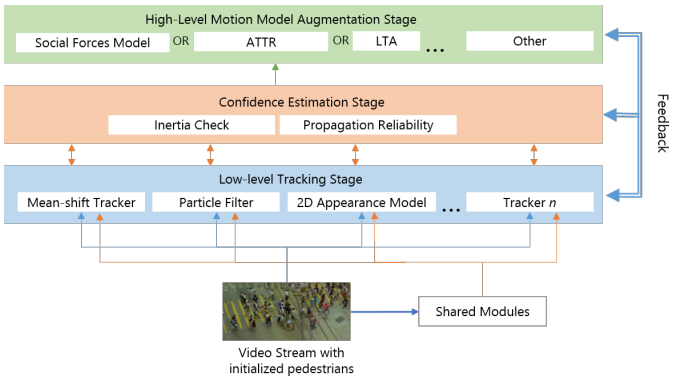

Fig. 3: *Our Adaptive Pedestrian Tracking Algorithm: Every video frame is processed in three stages. The low-level tracking stage (blue) computes the trajectories from known algorithms, such as deterministic or probabilistic trackers. A key component of the adaptive scheme is the confidence estimation stage (orange), which computes the most reliable trajectory for each pedestrian. The high-level motion model augmentation stage (green) uses a multi-agent approach to model the interactions between the pedestrians and estimate their trajectories. The shared modules are the modules used in common by different components.*

The main idea of this paper is to use an adaptive tracking approach that analyzes disparate pre-existing low-level tracking models and is guided by a high-level motion model. Our algorithm has three stages.

1) Low-Level Tracking Stage: During this stage, we run the individual low-level trackers, which share some of the common computational modules for reasons of efficiency. The output of this stage is a collection of trajectories for the pedestrians and a set of metrics that are used by other stages to calculate the confidence, or the reliability, of these individual trackers. We define $\text{Conf}_n(t)$ as the confidence of the n^{th} low-level tracker at time t, and $\text{Mex}_n(t)$ as the set of metrics for the n^{th} low-level tracking algorithm at time t:

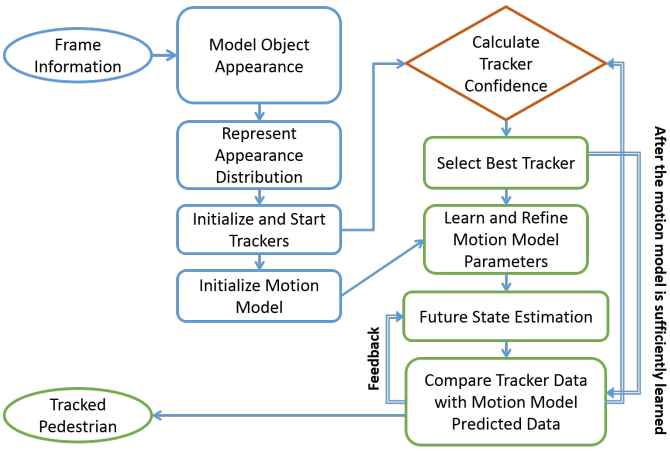

Fig. 4: *Algorithm Pipeline: We highlight the confidence-estimation algorithm that uses the information from low-level trackers and high-level motion model. The color of each module (blue, orange, and green) represents the corresponding stage in Figure (3).*

$$[\text{TJ}_n(t), \text{Mex}_n(t)]_{n=(0..m)} = f_{LLT}([\text{LT}_n(S(t))), \quad (1)$$

where f_{LLT} represents the low-level tracking stage of our algorithm and m is the total number of low-level tracking algorithms used.

2) Confidence Estimation Stage: In this stage, we analyze the output of different trackers and calculate the accuracy of each tracker relative to the others; the best one is produced as the output of this stage

$$\text{LT}_B(t) = f_{HTS}([\text{LT}_n(t), \text{Mex}_n(t)]_{n=0..m}), \quad (2)$$

where f_{HTS} represents the middle stage, m is the total number of low-level trackers and LT_B represents the best tracker for that frame. We use two different techniques to measure tracker confidence.

a) Temporal and Spatial Coherence Check: This method is applicable to all trackers. The main idea is that moving pedestrians do not abruptly change their velocities or locations. If there is a sudden change in the pedestrian state (*Velocity, Position*), we assume that change represents a drop in the confidence level. We treat this property as a hard check. If this check fails, the tracker automatically switches to the next best low-level tracker. We measure the change based on:

$$\Delta_v(t) = \frac{S_p^j(t) - S_p^j(t-1)}{S_p^j(t-1)}, \quad \Delta_p(t) = \frac{S_v^j(t) - S_v^j(t-1)}{S_v^j(t-1)}.$$

Intuitively, the tolerance values of Δ_v and Δ_p are different; people change speeds quickly and easily, but changing their position involves greater effort. So a tracker switch will occur when either Δ_v or Δ_p cross a certain threshold. More detail on these values are given in the implementation section.

b) Propagation Reliability: This is a tracker-specific check to establish how well the object model matches the target candidate at every frame. Many tracking algorithms use the *Bhattacharyya coefficient* to measure the distance

between the original template or the object model to the target candidate. In some tracking models, *Kullback-Leibler divergence* or the *Hellinger distance* are used to quantify the similarities between two probability distributions. We define the confidence of that tracker to be a function of this distance and the kernel size.

$$\text{Conf}_n(t) = \text{TD}(p, q, k), \tag{3}$$

where TD is the function of the similarity of two probability distributions that compare the target (p) with the object model (q) and the kernel size(k). In low- or medium-density crowds, the confidence estimate may drop and then rebound when the tracker has caught up with another template that also matches closely. It is quite possible that in this case a tracker has switched to tracking a different pedestrian, as most pedestrians have similar histograms due to the camera positioning. We can capture this sudden drop of confidence and switch automatically to another tracker.

For probabilistic trackers like the particle filter, we calculate the confidence differently. We maintain the probability distribution over the pedestrian state (position, velocity) of the object being tracked. The distribution is a set of particles (or weighted samples). The set of particles contains more weight at locations where the object being tracked is more likely to be. This distribution is propagated through time and the weights are iteratively updated.

When calculating confidence, we define it as the mean of the weights of the particles with the highest m weights.

$$\text{Conf}_n(t) = \frac{1}{m}\sum_{i=0}^{m} w_i, \tag{4}$$

where $[w_i]_{i=0..m}$ are the highest m weights of the particles.

After we have computed confidence measures for all trackers, we classify each measurement into three classes: high, medium, or low accuracy. Later in the implementation section, we present the confidence values for this classification based on our experiments.

The confidence values are sensitive to noise; therefore, we divide them into discreet classes before classifying the tracker. We aim to choose the tracker in the highest confidence class. If two or more trackers belong to the same highest class, the one with the least computational cost is chosen as our best tracker.

Window Reset: After computing the best tracker for a given frame, we reset and re-initialize the tracking template for all the other low-level trackers to the location of the best tracker and re-initialize the tracking parameters as:

$$[\text{LT}_n(t)]_{0..n} = \text{Best}([\text{LT}_{(i-1)}(t)]_{0..n}). \tag{5}$$

Finally, we analyze all the confidence measures$[\text{Conf}_i]_{0..n}$ and choose the best tracker amongst them. Switching between trackers can sometimes create a slight jitter or a sudden "jump" in pedestrian motion, as the different trackers might have slightly different trajectories. To reduce these artifacts, the tracker switch is only performed when the Euclidean between the trajectory of the best tracker and the next best tracker exceeds a user-defined threshold. This results in smoother trajectories and also reduces the number of window resets.

3) High-Level Motion Model Augmentation Stage: We integrate AdaPT with a pedestrian motion estimation model. Pedestrians are modeled as particles that are affected by repulsive and attractive forces [13]. Based on these social rules and interactions, the motion model is used to generate or estimate a trajectory for all pedestrians in the scene at every time step.

In Figure (5) we highlight how we use a motion model to compute the trajectory of each moving pedestrian and adaptively learn the simulation parameters based on the tracked data. The resulting motion for each agent is computed using statistical techniques, including Ensemble Kalman filters and maximum-likelihood estimation algorithms, to learn individual motion parameters from noisy data.

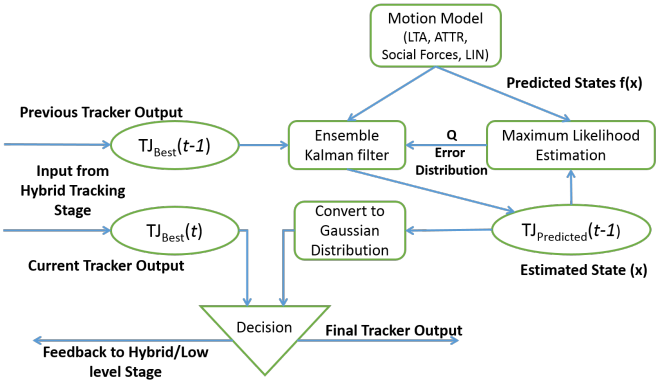

Fig. 5: *Overview of the High-Level Motion Model Augmentation Stage. This stage draws input from the Confidence Estimation stage, learns model parameters and improves tracking by providing feedback. The feedback is bidirectional and the model is re-trained after a fixed number of frames. The 'Decision' module is a parameter that can be modified depending on how much importance the motion model is given.*

The current pedestrian state is computed by using the output from our best tracker and recursively re-estimating the current state. The model combines the EM (Expectation-Maximization) algorithm with an ensemble Kalman Filtering approach to iteratively approximate the motion-model state of each agent at every timestep.

We perform a Bayesian learning for each pedestrian. Every pedestrian can be represented by a motion-model state vector $\mathbf{x}$. Given a pedestrian's state (position, velocity and preferred velocity), we use the motion model f to predict the pedestrian's next state $\mathbf{x}_{k+1}$. The motion model's error in predicting the state is denoted as $\mathbf{q}$; it follows a Gaussian distribution with covariance $\mathbf{E}$. Hence,

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k) + \mathbf{q}. \tag{6}$$

After receiving the predicted pedestrian state parameters from the motion model, we convert them into a Gaussian

probability distribution centered at the instantaneous position and perpendicular to the velocity vector. We select the tracker as follows. After converting the predicted state into a probability distribution (as determined by the high-level motion model augmentation stage), we examine the trackers that fall into the higher-probability band and select the one with the highest confidence (from the mid-level Confidence Estimation stage). The probability threshold can be modified based on how much weight is allocated to the motion model. If it falls below a certain limit, we return to the confidence-estimation algorithm stage and select the tracker with the second-highest confidence level, and so on. If the threshold limit is set very low, the motion-model results will not be used. On the other hand, if it is set too high, the tracking might fail if none of the low-level trackers meet the threshold. We denote the motion model's current prediction of the positional state as X_p; the current best tracker has output X_t at frame f. The guiding probability threshold used by the tracking algorithm would be

$$p_m(f) = e^{\frac{-1}{2}\|X_p(f) - X_t(f)\|^2}. \quad (7)$$

IV. Implementation

In this section we present details of our implementation, including various low-level trackers and the motion model. In our current implementation, we use the Social Force Model as the pedestrian motion model, along with the mean-shift tracking algorithm based on fuzzy color histograms [15]and the Sampling Importance Re-sampling (SIR) k-particle filter [35]. We use mean-shift as the most inexpensive tracker and then initialize the k-particle filter when the confidence in the mean-shift tracker fails (Equation (3)). We only use the particle filter when the reliability of the mean-shift tracker is low (see Figure (6)).

Fig. 6: *This graph demonstrates the adaptive use of different low-level trackers for different frames along the green line. When we notice that the confidence (blue) in the mean-shift tracker drops (e.g. frames 13, 38, 56, and 90), we re-initialize and re-start the particle filters with the prior state information corresponding to the high confidence level. Once the confidence in the mean-shift tracker increases (e.g. frames 20, 48, 65, and 90), we switch off the particle filter to reduce the computational cost.*

The mean-shift tracker works well for most scenarios and pedestrian locations, but its performance degrades with high pedestrian density, occlusion or clutter in the scene. In contrast, the k-particle filter works significantly better under such conditions, but has a high computational cost. In our experiments, we found that the SIR particle filter is at least 3 times to 4 times slower than the mean-shift tracker using fuzzy color histograms (for k=100, where k is the number of particles). In terms of the shared modules, we find that histogram computations and color-space conversion (BGR to HSV) are two modules that are used by both these trackers.

We use the following table to classify the confidence values (Equation (3)) into three categories. The values in the table were computed after plotting the change in confidence vs. percentage of error. These experiments were performed on 21 different video benchmarks (including [20], [27]), and the performance was compared with ground-truth results. In our experiments, we have used the tolerance value 0.3 for Δ_v and 0.6 for Δ_p. These numbers were computed based on Ounpuu 's prior work on pedestrian kinematics in Bio-mechanics [26].

Confidence Class	Change in confidence
High	<0.07
Medium	>0.07 & <0.4
Low	>0.4%

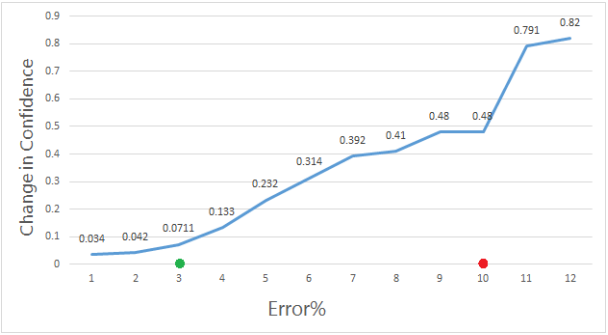

Fig. 7: *Change in confidence vs percentage of error. This was performed to evaluate the variation in the confidence function with respect to the tracking error. This classification is used to determine whether the tracker accuracy for a pedestrian is low, indicating that we need to switch to a more accurate tracker.*

A. Mean-shift using Fuzzy Color Histogram

The mean-shift tracker is a non-parametric feature-space analysis technique. It is an iterative kernel-based method that is used to locate the maxima of a density function given discrete data sampled from that function. The mean-shift tracking module is based on mean-shift iterations and eventually finds the most probable target location during each frame. The similarity match between the fuzzy color distribution (Histogram) of the target model and the target candidates is expressed by the *Bhattacharyya coefficient*. Histograms are widely used as a form of target representation because they are independent of rotation, scale changes and even resolution changes.

We define the target model based on the normalized fuzzy color histogram - $\vec{q} = \{q_u\}_{1..m}$, where m = number of bins in the histogram and $\sum_{u=1}^{m} q_u = 1$). The target candidate

centered at y) is $\vec{p} = \{p_u(y)\}_{1..m}$, where $\sum_{u=1}^{m} p_u = 1$). c_h is the *normalization factor* The probability of feature u in model is

$$q_u = C \sum_{b(x_i)=u} k(\|x_i\|^2) G_u(f(x_i)). \tag{8}$$

The probability of feature u in candidate is given as

$$p_u(y) = C_h \sum_{b(x_i)=u} k(\left\| \frac{y - x_i}{h} \right\|^2) G_u(f(x_i)), \tag{9}$$

where G_u is the output of the fuzzy cluster [15], $\{x_i\}_{i=1..n}$ are the target pixel locations, $k(x)$ is an isotropic, differentiable, convex, monotonically decreasing kernel profile with bandwidth h, and $b(x_i)$ associates the pixel x_i to the color histogram bin $(1..m)$.

We define the confidence ($\text{Conf}_n(t)$) of the tracker as

$$- \left\| \frac{\sum_{u=1}^{m} (\sqrt{p_u(y_{min}) q_u^c}) - \sum_{u=1}^{m} (\sqrt{p_u(y_{min}) q_u^i})(1 - K_s) + D_s}{\sum_{u=1}^{m} (\sqrt{p_u(y_{min}) q_u^i})} \right\|, \tag{10}$$

where K_s is the kernel size, D_s is the distance between the center of the pedestrian in two consecutive frames, q^i is the initial target model, q^c is the current target model, and $p(y_{min})$ is the target candidate corresponding to the maximum value of the Bhattacharyya coefficient.

B. Particle Filter

The particle filter is a parametric method that solves non-Gaussian and non-linear state estimation problems. Because it can recover from lost tracks and occlusions, the particle filter is frequently used in object tracking. However, the particle filter or the Kalman filter may not be fast enough for real-time pedestrian tracking for low- to medium-density crowds.

A particle-filter-based tracker maintains a probability distribution over the state of each pedestrian being tracked. This distribution is represented as a set of weighted samples (particles). Every particle is a guess or a possibility representing a state location of the object being tracked. These particles are weighted, and these weights are updated after every iteration. There is a greater likelihood of finding the object near the set of particles with higher weight. We can determine the trajectory of the tracked pedestrian by evaluating the particle with the highest weight during each time step. This weighted distribution is propagated through time according to the motion model.

A particle filter solves the tracking problem based on the following nonlinear state-space model: $x_t = f(x_{t-1}, u_t) + \delta_t$. The measurement equation is $y_t = g(x_t, v_t) + \eta_t$, where x_t is the state vector, f and g are the time-varying, nonlinear system equations, and u and v are independent but identically-distributed stochastic processes. The objective is to compute the $pdfp(x_t|y_{1:t})$ at every t.

We define the confidence for this tracker as

$$Conf_n(t) = \frac{\sum_{i=0}^{n/10} w^i}{n/10}, \tag{11}$$

where n is the total number of initial particles and w is the weights for the highest 10% of all the particles weights. We use 10% of all particles is that at every state the particles are refined, a narrower selection of particles increases the possibility of finding the pedestrian but if its too low, there is chance that we lose the pedestrian altogether.

The particle filterâĂŹs performance depends heavily upon the characteristics of the scenes and the object being tracked. Moreover, the number of particles needed to model the variations of the probability distribution function increase exponentially in higher-dimensional state spaces, which increases the computational load. When the state space is not densely sampled, the accuracy drops; the only way to overcome this is to increase the number of particles, which directly affects computation time.

C. Social Forces Model

The Social Force model is defined by the combination of three different forces: personal motivation force, social forces, and physical constraints.

Personal Motivation Force (F_i^M): This is the incentive to move at a certain preferred velocity v_i and in certain direction d_i.

Social Forces (F_i^S): These are the repulsive forces from other pedestrians P and obstacles O.

Physical Constraints (F_i^P): These are the hard constraints other than the environment and other pedestrians.

$$F_i^M = m_i \frac{v_i d_i - u_i}{\tau_i} \tag{12}$$

$$F_i^S = \sum_{j \in P\{i\}} f_{i,j}^S + \sum_{o \in O} f_{i,o}^S \tag{13}$$

$$F_i^P = \sum_{j \in P\{i\}} f_{i,j}^P + \sum_{o \in O} f_{i,o}^P \tag{14}$$

where u_i is present velocity, v_i is preferred velocity, and m_i is the mass of pedestrian p_i

The net force is $F_i^C = F_i^M + F_i^S + F_i^P$. (For a detailed explanation of this method, refer to [13]). Our model tries to estimate these individual forces and predict the future state of each pedestrian in forthcoming frames.

We represent the state of each pedestrian as a six dimensional vector:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{v}_{pref} \end{bmatrix}, \tag{15}$$

where $\mathbf{p}$ is the agent's position, $\mathbf{v}$ the velocity, and $\mathbf{v}_{pref}$ is the velocity, which typically points towards the pedestrian's goal. The crowd dynamics model f is:

$$f(\begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{v}_{pref} \end{bmatrix}) = \begin{bmatrix} \mathbf{p} + \mathbf{v}\Delta t \\ \frac{\int \mathbf{F_i^C dt}}{\mathbf{m_i}} \\ \mathbf{v}_{pref} \end{bmatrix}. \tag{16}$$

Once we have the pedestrian state, we estimate the future state via an iterative process, using the tracker data (which

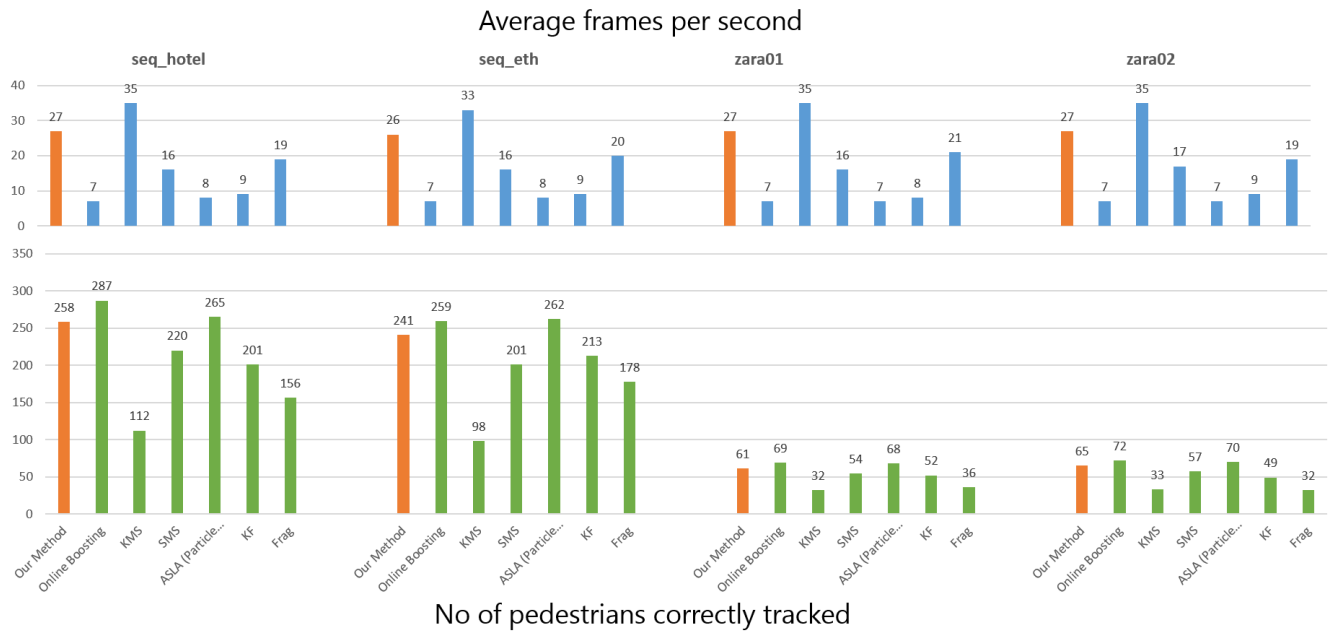

Fig. 8: *We compare AdaPT with other online trackers. The top graph represents the average fps achieved on our test machine. The bottom graph represents the tracking accuracy. (Algorithms: Online Boosting [4], KMS [9], SMS [8], ASLA [14], Frag [1], Kalman Filter [40]. Datasets: seq_hotel [27], seq_eth [27], zara01 [20], zara02 [20])*

is noisy), the Social Forces motion model, and the error distribution matrix E (Equation (6)). E is recomputed based on the difference between the tracked state and the prediction $f(x)$ and will be used to refine our current estimation of x. Once we have x, we convert this state to a Gaussian probability distribution (Equation (7)) and calculate how likely it is that our tracker follows our motion model.

V. Results

In this section, we highlight the performance of our algorithm on different benchmarks and compare the performance with these prior techniques: *Online Boosting* [4], *KMS* [9], *SMS* [8], *ASLA* [14], *Frag* [1], and *Kalman Filter* [40]. We compare the accuracy (in terms of pedestrians successfully tracked in the video sequence) and speed (frames per second) on the following 4 datasets: BIWI Walking Pedestrians dataset (*seq_hotel and seq_eth*) [27] and Crowds by example (*zara01, zara02*) [20]. A track is counted as "successful" when the estimated mean error between the tracking result and the ground-truth value is less than 0.8 meter in ground space.

We tested these algorithms on an Intel©Haswell, Core®i7-4771 Processor (4 Cores) with an 8MB Cache, 3.90 GHz and Intel©HD Graphics 4600. AdaPT is implemented in C++, and some components use OpenMP and OpenCL to exploit multiple cores.We adopted a 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).

TABLE I: *Crowd Scenes used as benchmarks. We highlight many attributes of these videos, along with the total number of pedestrians.*

seq_hotel	Illumination Variations, Occlusion	390
seq_eth	Background Variations, Illumination Changes	360
zara01	Background Variations, Illumination Changes, Occlusion	148
zara02	Background Variations, Illumination Changes, Occlusion	204

Fig. 9: *AdaPT's frame rate in this scene is* 27 *fps, tracking* 14 *pedestrians accurately in real-time. (Dataset CRW*116*: Web Stock Footage from Collection "Pedestrians from above")*

VI. Limitations, Conclusions, and Future Work

We present a realtime algorithm for pedestrian tracking in crowded scenes. Our adaptive scheme provides high accuracy and performance. We highlight its performance on well-known pedestrian datasets; it can track crowded scenes with

tens of pedestrians at interactive rates on a PC with a multi-core CPU.

Our approach has some limitations related to confidence-estimation computation. If our estimate is overly conservative, the performance of the adaptive tracker will be close to that of a probabilistic tracker. Errors in the motion model or propagation reliability can impact the accuracy. In practice, the performance of the algorithm can vary based on various attributes of a video stream.

There are many avenues for future work. We would like to evaluate our approach on other crowd scenarios corresponding to input and output environments with varying density and illumination conditions. The performance of our approach can further improve by exploiting the parallel capabilities of current systems to maximize data parallelism and also implement our tracker on mobile platforms and integrate with robots to perform autonomous navigation.

REFERENCES

[1] Amit Adam, Ehud Rivlin, and Ilan Shimshoni. Robust fragments-based tracking using the integral histogram. In *CVPR*, pages 798–805, 2006.

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

[3] Shai Avidan. Ensemble tracking. *PAMI*, pages 261–271, 2007.

[4] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *CVPR*, pages 983–990, 2009.

[5] Aniket Bera and Dinesh Manocha. Realtime multilevel crowd tracking using reciprocal velocity obstacles. 2014.

[6] Michael D Breitenstein, Fabian Reichlin, Bastian Leibe, Esther Koller-Meier, and Luc Van Gool. Online multiperson tracking-by-detection from a single, uncalibrated camera. *PAMI*, pages 1820–1833, 2011.

[7] Shu-Yun Chung and Han-Pang Huang. A mobile robot that understands pedestrian spatial behaviors. In *IROS*, pages 5861–5866, 2010.

[8] Robert T Collins. Mean-shift blob tracking through scale space. In *CVPR*, pages 661–675, 2003.

[9] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *PAMI*, pages 564–577, 2003.

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

[11] Andreas Ess, Bastian Leibe, Konrad Schindler, and Luc Van Gool. Robust multiperson tracking from a mobile platform. *PAMI*, pages 1831–1846, 2009.

[12] Tarak Gandhi and Mohan M Trivedi. Pedestrian protection systems: Issues, survey, and challenges. *ITS*, pages 413–430, 2007.

[13] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 1995.

[14] Xu Jia, Huchuan Lu, and Ming-Hsuan Yang. Visual tracking via adaptive structural local sparse appearance model. In *CPVR*, pages 1822–1829, 2012.

[15] Ming-Yi Ju, Chen-Sen Ouyang, and Hao-Shiu Chang. Mean shift tracking using fuzzy color histogram. In *ICMLC*, pages 2904–2908, 2010.

[16] Zia Khan, Tucker Balch, and Frank Dellaert. An mcmc-based particle filter for tracking multiple interacting targets. In *ECCV*, pages 279–290. 2004.

[17] Marin Kobilarov, Gaurav Sukhatme, Jeff Hyams, and Parag Batavia. People tracking and following with mobile robot using an omnidirectional camera and a laser. In *ICRA*, pages 557–562, 2006.

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

[19] Junseok Kwon and Kyoung Mu Lee. Tracking by sampling trackers. In *ICCV*, pages 1195–1202, 2011.

[20] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. In *Computer Graphics Forum*, pages 655–664, 2007.

[21] Yuan Li, Haizhou Ai, Takayoshi Yamashita, Shihong Lao, and Masato Kawade. Tracking in low frame rate video: A cascade particle filter with discriminative observers of different life spans. *PAMI*, pages 1728–1740, 2008.

[22] Z Li, QL Tang, and N Sang. Improved mean shift algorithm for occlusion pedestrian tracking. *Electronics Letters*, pages 622–623, 2008.

[23] Wenxi Liu, Antoni B. Chan, Rynson W. H. Lau, and Dinesh Manocha. Leveraging long-term predictions and online-learning in agent-based multiple person tracking. 2014.

[24] Matthias Luber, Luciano Spinello, and Kai O Arras. People tracking in rgb-d data with on-line boosted target models. In *IROS*, pages 3844–3849, 2011.

[25] Shaul Oron, Aharon Bar-Hillel, Dan Levi, and Shai Avidan. Locally orderless tracking. In *CVPR*, pages 1940–1947, 2012.

[26] Sylvia Ounpuu. The biomechanics of walking and running. *Clin Sports Med*, pages 843–863, 1994.

[27] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *ICCV*, pages 261–268, 2009.

[28] Patrick Pérez, Carine Hue, Jaco Vermaak, and Michel Gangnet. Color-based probabilistic tracking. In *ECCV*. 2002.

[29] Cédric Pradalier, Jorge Hermosillo, Carla Koike, Christophe Braillon, Pierre Bessière, and Christian Laugier. The cyca b: a car-like robot navigating autonomously and safely among pedestrians. *Robotics and Autonomous Systems*, pages 51–67, 2005.

[30] Mikel Rodriguez, Ivan Laptev, Josef Sivic, and J-Y Audibert. Density-aware person detection and tracking in crowds. In *ICCV*, pages 2423–2430, 2011.

[31] Mikel Rodriguez and Josef et al. Sivic. Data-driven crowd analysis in videos. In *ICCV*, pages 1235–1242, 2011.

[32] Juan C SanMiguel, Andrea Cavallaro, and José M Martínez. Standalone evaluation of deterministic video tracking. In *ICIP*, pages 1353–1356, 2012.

[33] Junji Satake and Jun Miura. Robust stereo-based person detection and tracking for a person following robot. In *ICRA Workshop on People Detection and Tracking*, 2009.

[34] Pramod Sharma, Chang Huang, and Ram Nevatia. Unsupervised incremental learning for improved object detection in a video. In *CVPR*, pages 3298–3305, 2012.

[35] Adrian FM Smith and Alan E Gelfand. Bayesian statistics without tears: a sampling–resampling perspective. *The American Statistician*, pages 84–88, 1992.

[36] Xuan Song, Xiaowei Shao, Quanshi Zhang, Ryosuke Shibasaki, Huijing Zhao, Jinshi Cui, and Hongbin Zha. A fully online and unsupervised system for large and high-density area surveillance: Tracking, semantic scene learning and abnormality detection. *TIST*, 2013.

[37] Pete Trautman, Jeremy Ma, Richard M Murray, and Andreas Krause. Robot navigation in dense human crowds: the case for cooperation.

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

[39] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *ICRA*, pages 1928–1935, 2008.

[40] Shiuh-Ku Weng, Chung-Ming Kuo, and Shu-Kang Tu. Video object tracking using adaptive kalman filter. *Journal of Visual Communication and Image Representation*, pages 1190–1208, 2006.

[41] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. pages 2411–2418, 2013.

[42] Fengliang Xu, Xia Liu, and Kikuo Fujimura. Pedestrian detection and tracking with night vision. *ITS*, pages 63–71, 2005.

[43] Kota Yamaguchi, Alexander C Berg, Luis E Ortiz, and Tamara L Berg. Who are you with and where are you going? In *CVPR*, pages 1345–1352, 2011.

[44] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm Computing Surveys (CSUR)*, 2006.

[45] Kaihua Zhang, Lei Zhang, and Ming-Hsuan Yang. Real-time compressive tracking. In *ECCV*, pages 864–877. 2012.

[46] Xuemei Zhao, Dian Gong, and Gérard Medioni. Tracking using motion patterns for very crowded scenes. In *ECCV*, pages 315–328. 2012.

[47] Brian D Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, James A Bagnell, Martial Hebert, Anind K Dey, and Siddhartha Srinivasa. Planning-based prediction for pedestrians. In *IROS*, pages 3931–3936, 2009.