Real-Time Certified Probabilistic Pedestrian Forecasting

Henry O. Jacobs, Owen Hughes, Matthew Johnson-Roberson, and Ram Vasudevan

***Abstract*— That autonomous vehicles will come to dominate our streets is imminent. This motivates the need for a real-time probabilistic forecasting algorithm for pedestrians, cyclists, and other agents, as it forms a necessary step in assessing the risk (and therefore the cost) we should expect to incur. In this paper, we present a novel approach to probabilistic forecasting for pedestrians based on weighted sums of ordinary differential equations. The resulting algorithm is embarrassingly parallel, and trained on historical trajectory information within a fixed scene. When compared with MDP-based methods, our algorithm appears to be superior from the standpoint of precision and recall.**

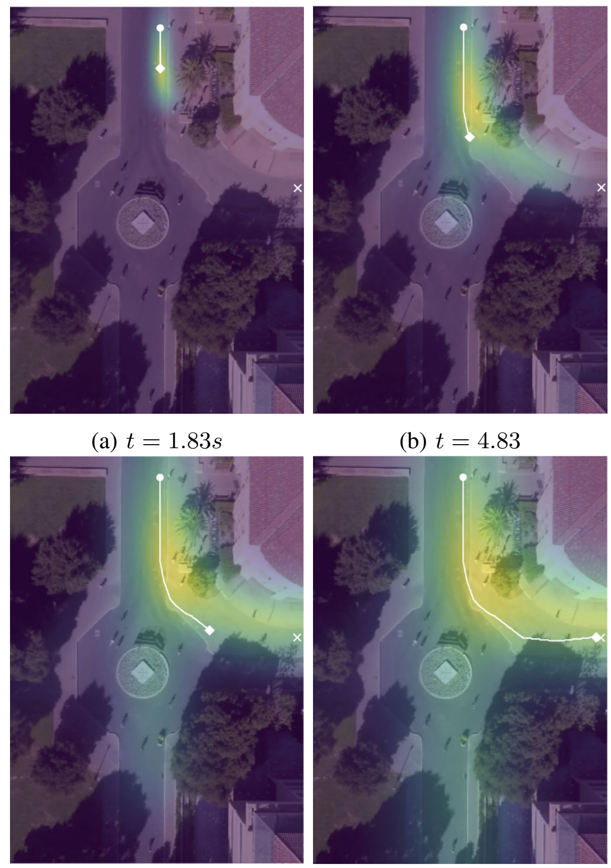
I. Introduction

Autonomous systems are increasingly being deployed in and around humans. The ability to accurately model and anticipate human behavior is critical to maximizing safety, confidence, and effectiveness of these systems in human-centric environments. The stochasticity of humans necessitates a probabilistic approach to capture the likelihood of an action over a set of possible behaviors. Since the set of plausible human behaviors is vast, this work focuses on anticipating the possible future locations of people within a bounded area. This problem is critical in many application domains such as enabling personal robots to navigate in crowded environments, managing pedestrian flow in smart cities, and synthesizing safe controllers for autonomous vehicles (AV).

With a particular focus on the AV application several additional design criteria become important. First, false negative rates for unoccupied regions must be minimized. The misclassification of space in this way has obvious safety issues. Second, speed is paramount. To effectively use human prediction within a vehicle control loop prediction rates must be commensurate with the speed at which humans change trajectories. The margins between humans and vehicles are small and both frequently operate at the boundary. Finally, long-time horizon forecasting is preferable since this improves robot predictability, reduces operation close to safety margins, prevents the need for overly conservative or aggressive controllers and makes high-level goal planning more feasible. This paper presents an algorithm for real-time, long-term prediction of pedestrian behavior which can subsequently be used by autonomous agents. As depicted in Figure 1, this method works quickly to generate predictions that are precise while reducing the likelihood of false negative detections.

H.O. Jacobs, O. Hughes, and R. Vasudevan are with the Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109 `{hojacobs,owhughes,ramv}@umich.edu`

M. Johnson-Roberson is with the Department of Naval and Marine Engineering, University of Michigan, Ann Arbor, MI 48109 `mattjr@umich.edu`



(a) $t = 1.83s$ (b) $t = 4.83$ (c) $t = 7.83s$ (d) $t = 11.5s$

Fig. 1: The performance of the presented algorithm captures the most probable routes that a pedestrian chooses. In this example which was captured at thirty frames per second, the presented algorithm took 0.0158s per frame in a Python implementation. In this figure, the dot is the start point of the test trajectory, the diamond is the position at time t, and the X is the end of the trajectory.

A. Background

Most forecasting algorithm are well characterized by the underlying evolution model they adopt. Such models come in a variety flavors, and are adapted to the task at hand (e.g. crowd modeling [1]). This paper is focused on the construction of useful motion models for pedestrians that can aide the task of real-time forecasting for autonomous agents. The simplest approach to forecasting with motion models forward integrates a Kalman filter [2] based upon the observed heading. Over short time scales this method may perform well, but the resulting distribution devolves into an imprecise Gaussian mass over longer time scales. This is a typical property of the extended Kalman filter and the Kalman filter with deterministic nonlinear drift [3]. In particular, such models are less useful for forecasts beyond two seconds, especially when a pedestrian turns. Nonetheless, these stochastic linear models serve as a reasonable default in the absence of any contextual knowledge. For example, [4] uses a motion model of this variety, parametrized by latent variables, such as the pedestrian's awareness of nearby vehicles, to predict when and how a pedestrian may cross a street without leveraging any environmental context.

More sophisticated models that attempt to leverage environmental data include inverse optimal control (IOC) based models [5], [6], [7], [8], [9]. These IOC models have the desirable property of attributing intention and goal-seeking behavior to the agents. For example, [7] extracts a Markov Decision Process (MDP) evolving on a finite 2D lattice by training on a small collection of features and trajectories. The resulting MDP is light-weight since it is parametrized by only a small collection of coefficients equal in number to that of the feature maps. Moreover, given an initial and final state, the probability distribution of the pedestrian at intermediate states is computable using matrix multiplication. The speed of this computation, makes the algorithm of [7] a reasonable baseline for comparison for the algorithm that is presented in this paper.

The approach presented in [7] has been generalized in a variety of ways. For example, time-dependent information, such as traffic signals, are incorporated in [9], by relaxing the Markov property and considering a switched Markov process. Other generalizations in [9] include replacing the finite-state space with a continuous one, and using a Markov jump process in the motion model. Unfortunately the desired posteriors are difficult to compute in closed form, and as a result a sampling based method: a Rao-Blackwellized particle filter [10] is employed. The resulting accuracy of such methods can only be known in a probabilistic sense in that the error bounds are themselves random variables. Moreover, accuracy can come at a large computational expensive which is prohibitive during real-time control.

One limitation of IOC models occurs in cases where there are many locally optimal solutions between a start and end goal. In these cases, IOC type methods can yield non-robust and imprecise behavior. This can occur, for example, when agents make sharp turns due to intermediate criteria on the way toward reaching their final destination. To address this, [11] adopt an empiricists approach, computing "turning maps" and attempting to infer how agents behave in a given patch based on its feature and the behavior of previous agents on similar patches. The motion model is a Markov jump process and the relevant posteriors were approximated using sample based techniques similar to [9]. The objective of [11] is not only prediction, but the development of a motion model learned on one scene that could then subsequently be transferred to other scenes. This requires representations of "objects" in the scene that do not depend rigidly on the finitely many labels an engineer managed to think of in a late-night brainstorming session. For example, such an algorithm should be able to infer how to behave near a roundabout by using prior knowledge of more fundamental building blocks like pavement, curbs, and asphalt.

Along similar lines to [11], [12] constructed an unsupervised approach towards forecasting. As before, the motion model was a Markov jump process, and the training set was a collection unlabeled videos. Unlike all the approaches mentioned thus far, the agents in [12] were not manually specified. They were learned by detecting which sort of patches of video were likely to move, and how. The resulting predictions outperformed [7] when comparing the most likely path with the ground truth using the mean Hausdorff distance. As in all methods mentioned thus far, computational speed and accuracy of any predicted posteriors were not a concern of [12], so no such results were reported. However, since the motion model was a Markov jump process which required the application of a sample based technique, we should expect the same painful trade-off between error and speed to occur as in [9] and [11].

B. Contributions

The primary contributions of this paper are:

- An accurate motion model for pedestrian forecasting.
- An expedient method for computing approximations of relevant posteriors generated by our motion model.
- Hard error bounds on the proposed approximations.

The method proposed by this paper is able to work three times faster than the existing state of the art while improving upon its performance over long time horizons.

For clarification, we should mention that there are a number of things that we do not do. For example, we do not concern ourselves with detection and tracking. Nor do we concern ourselves with updating our prediction as new data comes along [7]. We largely work in a 2D environment with a bird's eye view, operating under the assumption that the data has been appropriately transformed by a third party. This is in contrast to [9], which operates from first person video. While it would be a straight forward extension to consider such things, it would detract from the presentation.

The rest of the paper is organized as follows:

- §II describes our motion model as a Bayesian network.
- §III describes how to compute probability densities for an agent's position efficiently.
- §IV demonstrates the model by training and testing it on the Stanford drone dataset [13].

II. Model

This paper's goal is to generate a time-dependent probability density over $\mathbb{R}^2$, which predicts the true location of an agent in the future. The input to the algorithm at runtime is a noisy measurement of position and velocity, $\hat{x}_0, \hat{v}_0 \in \mathbb{R}^2$. If the (unknown) location of agent at time t is given by $x_t \in \mathbb{R}^2$, then the distribution we seek is the posterior $\rho_t(x_t) := \Pr(x_t \mid \hat{x}_0, \hat{v}_0)$.

To numerically compute ρ_t, we build a probabilistic graphical model. Our model assumes we have noisy information about agents, and each agent moves with some intention through the world in a way that is roughly approximated by a model. Our model can be divided into three parts:

1) Reality: This is parametrized by the true position for all time, x_t, and the initial velocity of the agent v_0.
2) Measurements: This is represented by our sensor readings $\hat{x}_0$ and $\hat{v}_0$ and are independent of all other variables given the true initial position and velocity, x_o, v_0.
3) Motion Model: This is represented by a trajectory $\check{x}_t$ and depends on a variety of other variables which are described below.

We now elaborate on these three components, and relate them to one another. Many of the choices we make are motivated by a balance between model quality and computational speed (see §III).

A. The Variables of the Model

The model concerns the position of an agent $x_t \in \mathbb{R}^2$ for $t \in [0, T]$ for some $T > 0$. We denote the position and velocity at time $t = 0$ by x_0 and v_0 respectively. At $t = 0$, we obtain a measurement of position and velocity, denotes by $\hat{x}_0$ and $\hat{v}_0$. Lastly, we have a variety of motion models, parametrized by a set $\mathcal{M}$ (described in the sequel). For each model $m \in \mathcal{M}$, a trajectory $\check{x}_t$ given the initial position and velocity x_0 and v_0. All these variables are probabilistically related to one another in a (sparse) Bayesian network, which we will describe next.

B. The Sensor Model

At time $t = 0$, we obtain a noisy reading of position, $\hat{x}_0 \in \mathbb{R}^2$. We assume that given the true position, $x_0 \in \mathbb{R}^2$, that the measurement $\hat{x}_0$ is independent of all other variables and the posterior $\Pr(\hat{x}_0 \mid x_0)$ is known. We assume a similar measurement model for the measured initial velocity $\hat{v}_0$.

C. The Agent Model

All agents are initialized within some rectangular region $D \subset \mathbb{R}^2$. We denote the true position of an agent by x_t. We should never expect to know x_t and the nature of its evolution precisely, and any model should account for its own (inevitable) imprecision. We do this by fitting a deterministic model to the data and then smoothing the results. Specifically, our motion model consists of a modeled trajectory $\check{x}_t$, which is probabilistically related to the true position by x_t via a known and easily computable posterior, $\Pr(x_t \mid \check{x}_t)$.

Once initialized, agents come in two flavors: linear and nonlinear. The linear agent model evolves according to the equation $\check{x}_t = x_0 + tv_0$ and so we have the posterior:

$$\Pr(\check{x}_t \in A \mid x_0, v_0, lin) = \int_A \delta(\check{x}_t - x_0 - tv_0) d\check{x}_t. \quad (1)$$

for all measurable sets $A \subset \mathbb{R}^2$, where $\delta(\cdot)$ denotes the multivariate Dirac-delta distribution. For the sake of convenience, from here on we drop the set A and the integral when defining such posteriors since this equation is true for all measurable sets A. We also assume the posteriors, $\Pr(x_0 \mid lin)$ and $\Pr(v_0 \mid lin, x_0)$ are known.

If the agent is of nonlinear type, then we assume the dynamics take the form:

$$n\frac{d\check{x}_t}{dt} = s \cdot X_k(\check{x}_t) \quad (2)$$

where X_k is a vector-field[1] drawn from a finite collection $\{X_1, \ldots, X_n\}$, and $s \in \mathbb{R}$. More specifically, we assume that each X_k has the property that $\|X_k(x)\| = 1$ for all $x \in D$. This property ensures that speed is constant in time. As we describe in §IV, the stationary vector-fields $X_1, \ldots, X_n$ are learned from the dataset.

It is assumed that k and s are both constant in time, so that $\check{x}_t$ is determined from the triple (x_0, k, s) by integrating (2) with the initial condition x_0. This insight allows us to use the motion model to generate the posterior for $\Pr(\check{x}_t \mid x_0, k, s)$. For each initial condition, x_0, we can solve (2) as an initial value problem, to obtain a point $\check{x}_t$ with initial condition $\check{x}_0 = x_0$. This process of solving the differential equation takes an initial condition, $\check{x}_0$, and outputs a final condition, $\check{x}_t$. This constitutes a map which is termed the *flow-map* [14, Ch 4], and which we denote by $\Phi^t_{k,s}$. Explicitly, we have the posterior:

$$\Pr(\check{x}_t \mid x_0, k, s) = \delta(\check{x}_t - \Phi^t_{k,s}(x_0)) d\check{x}_t \quad (3)$$

where $\Phi^t_{k,s}$ is the flow-map of the vector field $s\,X_k$ up to time t. Note that this flow-map can be evaluated for an initial condition efficiently by just integrating the vector field from that initial condition. Note the variables k, s and x_0 determine v_0. Thus we have the posterior:

$$\Pr(v_0 \mid k, s, x_0) = \delta(v_0 - sX_k(x_0)) dv_0. \quad (4)$$

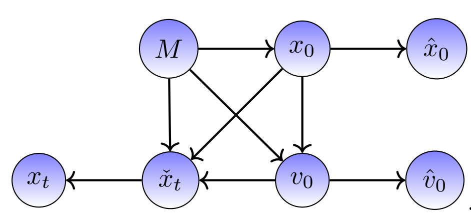
In summary, the agent models are parametrized by the set:

$$\mathcal{M} = \{lin\} \cup (\mathbb{R} \times \{1, \ldots, n\}) \quad (5)$$

and each flavor determines the type of motion we should expect from the agent model.

D. The Full Model

Concatenating the measurement model with our motion models yields the Bayesian network, where $M \in \mathcal{M}$ denotes the model of the agent:



. (6)

[1]A vector-field, generally speaking, is an assignment of a velocity to each position in some space. A vector-field on $\mathbb{R}^n$ is nothing but a map from $\mathbb{R}^n \to \mathbb{R}^n$.

We use this Bayesian network to compute ρ_t efficiently. In particular

$$\rho_t(x_t) := \Pr(x_t \mid \hat{x}_0, \hat{v}_0) \tag{7}$$
$$= \left(\sum_k \int \Pr(x_t, k, s \mid \hat{x}_0, \hat{v}_0) ds \right) + \Pr(x_t, lin \mid \hat{x}_0, \hat{v}_0). \tag{8}$$

The final term $\Pr(x_t, lin \mid \hat{x}_0, \hat{v}_0)$ is expressible in closed form when the posteriors $\Pr(x_0 \mid lin)$ and $\Pr(v_0 \mid lin, x_0)$ have known expressions (e.g. Gaussians or uniform distributions). In this instance, the numerical computation of $\Pr(x_t, lin \mid \hat{x}_0, \hat{v}_0)$ poses a negligible burden. Instead the primary computational burden derives from computing $\sum_k \int \Pr(x_t, k, s \mid \hat{x}_0, \hat{v}_0) ds$.

III. Efficient Probability Propagation

As mentioned, many of the modeling choices are born out of a balance between accuracy and real-time computability. One of the major modeling choices, that agents move approximately according to a small number of ODEs, is the most prominent such choice. This section details how this modeling choice can be leveraged to compute $\rho_t(x_t)$ quickly and accurately.

To begin, rather than focusing on computing $\rho_t(x_t)$, we describe how to compute the joint probability $\Pr(x_t, \hat{x}_0, \hat{v}_0)$. We can obtain $\rho_t(x_t)$ by normalizing $\Pr(x_t, \hat{x}_0, \hat{v}_0)$ with respect to integration over x_t. We can approximate the integration over s in (8), with a Riemann sum. Given a regular partition $\{s_0, s_1, \ldots, s_n\}$ of step-size $\Delta s > 0$ on the support of $\Pr(s)$, we can conclude that the integral term in (8) can be approximated by

$$\sum_k \int \Pr(x_t, k, s, \hat{x}_0, \hat{v}_0) ds = \underbrace{\Delta s \sum_j \sum_k \Pr(x_t, k, s_j, \hat{x}_0, \hat{v}_0)}_{\text{approximation}} + \underbrace{\varepsilon_s}_{\text{error}} \tag{9}$$

where the error is bounded by $\int |\varepsilon_s| ds \leq TV(x_t, \hat{x}_0, \hat{v}_0)\Delta s$ where $TV(x_t, \hat{x}_0, \hat{v}_0)$ is the sum, with respect to k, of the total variation of $\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0)$ with respect to s for fixed $x_t, \hat{x}_0, \hat{v}_0$. Since this error term can be controlled, the problem of solving $\rho_t(x_t)$ is reduced to that of efficiently computing $\Pr(x_t, k, s_j, \hat{x}_0, \hat{v}_0)$ for a fixed collection of s_j's.

$\hat{x}_0$ and $\hat{v}_0$ are measured and are assumed fixed for the remainder of this section. To begin, from (6) notice that:

$$\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0) = \tag{10}$$
$$= \int \Pr(x_t, \check{x}_t, \hat{x}_0, \hat{v}_0, k, s) d\check{x}_t \tag{11}$$
$$= \int \Pr(x_t \mid \check{x}_t) \Pr(\check{x}_t, \hat{x}_0, \hat{v}_0, k, s) d\check{x}_t \tag{12}$$

Observe that from the last line that $\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0)$ is a convolution of the joint distribution $\Pr(\check{x}_t, \hat{x}_0, \hat{v}_0, k, s)$. Assuming, for the moment, that such a convolution can be performed efficiently, we focus on computation of $\Pr(\check{x}_t, \hat{x}_0, \hat{v}_0, k, s)$. Again, (6) implies:

$$\Pr(\check{x}_t, \hat{x}_0, \hat{v}_0, k, s) = \tag{13}$$
$$= \int \Pr(\check{x}_t, x_0, \hat{x}_0, v_0, \hat{v}_0, k, s) dx_0\, dv_0 \tag{14}$$
$$= \int \Pr(\check{x}_t \mid x_0, k, s, v_0) \Pr(\hat{v}_0 \mid v_0) \cdot \cdot \Pr(v_0 \mid k, s, x_0) \Pr(\hat{x}_0, x_0, k, s) dx_0\, dv_0 \tag{15}$$
$$= \int \delta\left(\check{x}_t - \Phi^t_{k,s}(x_0)\right) \delta\left(v_0 - sX_k(x_0)\right) \cdot \cdot \Pr(\hat{v}_0 \mid v_0) \Pr(\hat{x}_0, x_0, k, s) dx_0\, dv_0, \tag{16}$$

where the last equality follows from substituting (3) and (4). Carrying out the integration over v_0 we observe:

$$\Pr(\check{x}_t, \hat{x}_0, \hat{v}_0, k, s) = \int \delta\left(\check{x}_t - \Phi^t_{k,s}(x_0)\right) \cdot \cdot \Pr(\hat{x}_0, x_0, k, s) \Psi(\hat{v}_0; k, s, x_0) dx_0, \tag{17}$$

where $\Psi(\hat{v}_0; k, s, x_0) := \Pr(\hat{v}_0 \mid v_0)|_{v_0 = sX_k(x_0)}$. We may approximate $\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0; k, s, x_0)$ as a sum of weighted Dirac-delta distributions supported on a regular grid, since $\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0; k, s, x_0)$ is of bounded variation in the variable x_0 (with all other variables held fixed).

To accomplish this, let $S_L(\hat{x}_0)$ denote the square of side length $L > 0$ centered around $\hat{x}_0$. Choose $L > 0$ to be such that $\int_{S_L(\hat{x}_0)} \Pr(x_0 \mid \hat{x}_0) dx_0 = 1 - \varepsilon_{tol}$ for some error tolerance $\varepsilon_{tol} > 0$. Then, for a given resolution $N_x \in \mathbb{N}$ define the regular grid on $S_L(\hat{x}_0)$ as $\Gamma_L(\hat{x}_0; N_x) := \left\{ x_0^{i,j} \mid i, j \in \{-N_x, \ldots, N_x\} \right\}$, where $x_0^{i,j} = \hat{x}_0 + \frac{L}{2N_x}(i,j)$. The grid spacing is given by $\Delta x = (\frac{L}{2N_x}, \frac{L}{2N_x})$. We approximate the smooth distribution $\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0; k, s, x_0)$ as a weighted sum of Dirac-deltas (in the variable x_0) supported on $\Gamma_L(\hat{x}_0; N)$:

$$\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0; k, s, x_0) = \underbrace{\left(\sum_{i,j=-N}^{N} W(k, s, i, j, \hat{x}_0) \delta(x_0 - x_0^{i,j}) \right)}_{\text{approximation}} + \underbrace{\varepsilon_0(x_0)}_{\text{error}} \tag{18}$$

where $W(k, s, i, j, \hat{x}_0)$ is the evaluation of $\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0; k, s, x_0)$ at the grid point $x_0 = x_0^{i,j} \in \Gamma(\hat{x}_0; N)$. More explicitly, this evaluation can be done for each grid point by using only the assumed posterior models in (6). For fixed k and s, the expression $\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0; k, s, x_0)$ is a density in x_0 and the error term in (18) has a magnitude of $\|\varepsilon_0\|_{L^1} \sim \mathcal{O}(|\Delta x| + \varepsilon_{tol})$ with respect to the L^1-norm in x_0.

Substitution of (18) into the final line of (17) yields:

$$\Pr(\check{x}_t, \hat{x}_0, \hat{v}_0, k, s) = \sum_{i,j} W(k, s, i, j, \hat{x}_0) \delta\left(\check{x}_t - \Phi_{k,s}(x_0^{i,j})\right) + \varepsilon_t(\check{x}_t) \tag{19}$$

where $\varepsilon_t(\check{x}_t) = \int \delta\left(\check{x}_t - \Phi^t_{k,s}(x_0)\right) \varepsilon_0(x_0) dx_0$. The first term of the right hand side of (19) is computable by flowing the points of the grid, $\Gamma_L(\hat{x}_0; N_x)$, along the vector field sX_k. The second term, ε_t, may be viewed as an error term. In fact, this method of approximating $\Pr(\check{x}_t, \hat{x}_0, \hat{v}_0, k, s)$ as a sum of Dirac-delta distributions is adaptive, in that the error term does not grow in total mass, which is remarkable since many methods for linear evolution equations accumulate error exponentially in time [15], [16]:

Theorem 1: The error term, $\varepsilon_t \sim \mathcal{O}(|\Delta x| + \varepsilon_{tol})$ in the L^1-norm, for fixed $k, s, \hat{x}_0$, and $\hat{v}_0$. Moreover, $\|\varepsilon_t\|_{L^1}$ is constant in time.

Proof: To declutter notation, let us temporarily denote $\Phi^t_{k,s}$ by Φ. We observe

$$\begin{aligned}\|\varepsilon_t\|_{L^1} &= \int \left| \int \delta(\check{x}_t - \Phi(x_0))\varepsilon_0(x_0)dx_0 \right| d\check{x}_t \\ &= \int \det(D\Phi|_{\Phi^{-1}(\check{x}_t)})|\varepsilon_0(\Phi^{-1}(\check{x}_t))|d\check{x}_t \\ &= \int |\varepsilon_0(u)|du = \|\varepsilon_0\|_{L^1}\end{aligned}$$

As ε_0 is of magnitude $\mathcal{O}(|\Delta x| + \varepsilon_{tol})$ the result follows. ■

While this allows us to compute posteriors over the output of our models, $\check{x}_t$, we ultimately care about densities over the true position. The following corollary of Theorem 1 addresses this:

Corollary 1: The density

$$\sum_{i,j} W(k, s, i, j, \hat{x}_0) \Pr(x_t \mid \check{x}_t)|_{\check{x}_t = \Phi^t_{k,s}(x_0^\alpha)} \tag{20}$$

is an approximation of $\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0)$ with a constant in time error bound of magnitude $\mathcal{O}(|\Delta x| + \varepsilon_{tol})$.

Proof: By (12)

$$\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0) = \int \Pr(x_t \mid \check{x}_t) \Pr(\check{x}_t, k, s, \hat{x}_0, \hat{v}_0) d\check{x}_t$$

Substitution of (18) yields

$$\begin{aligned}&\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0) \\ &= \sum_{i,j} W(k, s, i, j, \hat{x}_0) \Pr(x_t \mid \check{x}_t)|_{\check{x}_t = \Phi^t_{k,s}(x_0^{i,j})} + \tilde{\varepsilon}_t(x_t)\end{aligned}$$

where the error term is

$$\tilde{\varepsilon}_t(x_t) = \int \Pr(x_t \mid \check{x}_t)\varepsilon_t(\check{x}_t)d\check{x}_t \tag{21}$$

and ε_t is the error term of (18). We see that the L^1-norm of $\tilde{\varepsilon}_t$ is

$$\|\tilde{\varepsilon}_t\|_{L^1} = \int \left| \int \Pr(x_t \mid \check{x}_t)\varepsilon_t(\check{x}_t)d\check{x}_t \right| dx_t \tag{22}$$

$$\leq \int \Pr(x_t \mid \check{x}_t)|\varepsilon_t|(\check{x}_t)d\check{x}_t \, dx_t \tag{23}$$

Implementing the integration over x_t first yields:

$$\|\tilde{\varepsilon}_t\|_{L^1} \leq \int |\varepsilon_t|(\check{x}_t)d\check{x}_t =: \|\varepsilon_t\|_{L^1} \tag{24}$$

which is $\mathcal{O}(|\Delta x| + \varepsilon_{tol})$ by Theorem 1. ■

Corollary 1 justifies using (20) as an approximation of $\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0)$. In particular, this reduces the problem of computing $\rho_t(x_t)$ to the problem of computing the weights $W(k, s, i, j, \hat{x}_0)$ and the points $\Phi^t_{k,s}(x_0^{i,j})$ for all k, s and points $x_0^{i,j} \in \Gamma_L(\hat{x}_0; N_x)$. We can reduce this burden further by exploiting the following symmetry:

Algorithm 1 Algorithm to Compute ρ_t.

Require: $N_t \in \mathbb{N}, \{X_k\}_{k=1}^n, \hat{x}_0, \hat{v}_0, \Gamma_L(\hat{x}_0, N_x)$, and the posteriors of (6).
1: **for each** $m \in \{-\ell, \ldots, \ell\}$ **do**
2: Compute $W(k, s, i, j, \hat{x}_0)$ for $s = \bar{s}m/\ell$.
3: **end for**
4: Construct a $\mathcal{O}(\Delta x + \bar{s}/\ell + \varepsilon_{\text{tol}})$ approximation of $\Pr(\check{x}_t, \hat{x}_0, \hat{v}_0)$ at $t = \ell\Delta t$ via Theorem 3 .
5: Normalize with respect to $\check{x}_t$ to obtain an approximation of $\Pr(\check{x}_t \mid \hat{x}_0, \hat{v}_0)$.
6: Convolve with respect to $\Pr(x_t \mid \check{x}_t)$ to obtain $\rho_t(x_t)$ at $t = \ell \cdot \Delta t$

Theorem 2: $\Phi^t_{k,s} = \Phi^{st}_{k,1}$.

Proof: Say $x(t)$ satisfies the ordinary differential equation $x'(t) = sX_k(x(t))$ with the initial condition x_0. In other words, $x(t) = \Phi^t_{k,s}(x_0)$. Taking a derivative of $x(t/s)$, we see $\frac{d}{dt}(x(t/s)) = x'(t/s)/s = X_k(x(st))$. Therefore $x(t/s) = \Phi^t_{k,1}(x_0)$. Substitution of t with $\tau = t/s$ yields $x(\tau) = \Phi^{s\tau}_{k,1}(x_0)$. As $x(\tau) = \Phi^\tau_{k,s}(x_0)$ as well, the result follows. ■

This, allows us to compute $\Phi^t_{k,s}(x_0^\alpha)$ using computations of $\Phi^t_{k,1}(x_0^\alpha)$, which yields the following result:

Theorem 3: Let $\{s_1, \ldots, s_n\}$ be a regular partition on the support of $\Pr(s)$. Then the density

$$\begin{aligned}&\Delta s \sum_{i,j,k,\ell} W(k, s_\ell, i, j, \hat{x}_0) \Pr(x_t \mid \check{x}_t)|_{\check{x}_t = \Phi^{s_\ell t}_{k,1}(x_0^{i,j})} \\ &+ \Pr(x_t, lin, \hat{x}_0, \hat{v}_0)\end{aligned} \tag{25}$$

approximates $\Pr(x_t, \hat{x}_0, \hat{v}_0)$ with an error of size $\mathcal{O}(\Delta s + \Delta x + \varepsilon_{\text{tol}})$.

Proof: Substitute Theorem 2 into Corollary 1, to replace $\Phi^t_{k,s}$ with $\Phi^{st}_{k,1}$. This gives us an error term of size Δx, if we compute the integral over s exactly. Using (9), we can compute the integral over s approximately, with an error of magnitude Δs. ■

This is a powerful result, since if $\Pr(s)$ has a support contained within the domain $[-\bar{s}, \bar{s}]$, then to solve for $\Pr(x_t, \hat{x}_0, \hat{v}_0)$ (and thus $\rho_t(x_t)$) at times $0, \Delta t, \ldots, N_t\Delta t$, we only need to compute $\Phi^{\ell\Delta t\bar{s}}_{k,1}$ for each $\ell \in \{-N_t, \ldots, N_t\}$ once. This is because, once $\Phi^{\ell\Delta t\bar{s}}_{k,1}$ is computed for each $\ell \in \{-N_t, \ldots, N_t\}$ we can use the partition of $[-\bar{s}, \bar{s}]$ given by $\{\bar{s}\frac{m}{\ell}\}_{m=-\ell}^{\ell}$ to compute a $\Delta s = \bar{s}/\ell$ accurate approximation for each ℓ, courtesy of Theorem 3. The procedure to compute $\rho_t(x_t)$ is summarized in Algorithm 1.

For fixed k i and j, the computation of $\Phi^t_{k,1}(x_0^{i,j})$ at each $t = \{-N_t\bar{s}, \ldots, N_t\bar{s}\}$ takes $\mathcal{O}(N_t)$ time using an explicit finite difference scheme and can be done in parallel for each $k \in \{1, \ldots, n\}$ and $x_0^{i,j} \in \Gamma_L(\hat{x}_0; N_x)$. Similarly, computing $W(k, s, i, j, \hat{x}_0)$ constitutes a series of parallel function evaluations over tuples (k, s, i, j) of the posterior distributions described in (6), where the continuous variable s is only required at finitely many places in Algorithm 1. If the posteriors represented by the arrows in (6) are efficiently computably then the computation of $W(k, s, i, j, \hat{x}_0)$ is equally efficient.

IV. Implementation and Experimental results

Given this model, we describe a specific implementation and the process of fitting the model to a dataset. In particular, we must learn the vector fields $\{X_1, \ldots, X_n\}$, the posteriors $\Pr(x_0 \mid M)$ and the priors $\Pr(M)$ for $M \in \mathcal{M}$ from the data. For the purposes of demonstration, we use the Stanford Drone Dataset [13]. More generally, we assume that for a fixed scene we have a database of previously observed trajectories $\{\hat{x}^1, \ldots, \hat{x}^N\}$. From this data we tune the parameters of the model ($\{X_1, \ldots, X_n\}$, $\Pr(x_0 \mid M)$ and $\Pr(M)$) appropriately.

A. Learning the Vector Fields

We begin by identifying the number of possible vector-fields. To do this we use a clustering algorithm on the trajectories observed in a scene to categorize them into groups. In particular, we use the start and end point for each trajectory to obtain a point in $\mathbb{R}^4$. We then cluster in $\mathbb{R}^4$ using Affinity propagation [17]. This clustering of the end-points induces a clustering of the trajectories. Suppose we obtain clusters $S_1, \ldots, S_n$ consisting of trajectories from our data set, as well as a set of unclassified trajectories, S_0.

For each set S_k we learn a vector-field that is approximately compatible with that set. Since most trajectories appearing in the dataset have roughly constant speed, we chose a vector-field that has unit magnitude. That is, we assume the vector-field takes the form $X_k(x) = (\cos(\Theta_k(x)), \sin(\Theta_k(x)))$ for some scalar function $\Theta_k(x)$. Learning the vector-fields then boils down to learning the scalar function Θ_k. We assume Θ_k takes the form:

$$\Theta_k(x) = \sum_{\alpha} \theta_{k,\alpha} L_\alpha(x),$$

for some collection of coefficients, $\theta_{k,\alpha}$, and a fixed collection of basis functions, L_α. In our case, we choose L_α to be a set of low degree Legendre polynomials. Θ_k is learned by computing the velocities observed in the cluster, S_k. These velocities are obtained by a low order finite difference formula. Upon normalizing the velocities, we obtain a unit-length velocity vectors, $v_{i,k}$, anchored at each point, $x_{i,k}$, of S_k. We learn Θ_k by defining the cost-function:

$$C[\Theta_k] = \sum_i \langle v_{i,k}, (\cos(\Theta_k(x_{i,k}), \sin(\Theta_k(x_{i,k}))\rangle$$

which penalizes Θ_k for producing a misalignment with the observed velocities at the observed points of S_k. When Θ_k includes high order polynomials (e.g. beyond 5th order), we also include a regularization term to bias the cost towards smoother outputs. Using the H^1-norm times a fixed scalar suffices as a regularization term.

B. Learning $\Pr(x_0 \mid M)$ and $\Pr(M)$

We first considering the nonlinear models. We begin by assuming that x_0 is independent of s given k, i.e. $\Pr(x_0 \mid k, s) = \Pr(x_0 \mid k)$. Additionally, we assume that s and k are independent. This means that we only need to learn $\Pr(x_0 \mid k)$, $\Pr(k)$, and $\Pr(s)$.

We let $\Pr(k) = (n+1)^{-1}$ and $\Pr(s) \sim \mathcal{U}([-s_{\max}, s_{\max}])$ where $s_{\max} > 0$ is the largest observed speed in the dataset. This implies that $\Pr(lin) = (n+1)^{-1}$ as well.

For each k we assume $\Pr(x_0 \mid k) = \frac{1}{Z_k}\exp(-V_k(x_0))$ and V_k is a function whose constant term is 0 and is given by:

$$V_k(x_0; \mathbf{c}) := \sum_{|\alpha|<d} c_\alpha L_\alpha(x_0)$$

for a collection of basis functions, L_α and coefficients $\mathbf{c} = \{c_\alpha\}_{|\alpha|<d}$. We chose our basis functions to be the collection of tensor products of the first six Legendre polynomials, normalized to the size of the domain. Then, one may fit the coefficients c_α to the data by using a log-likelihood criterion. The resulting (convex) optimization problem takes the form:

$$\mathbf{c}^* = \inf_{|\mathbf{c}|} \sum_{x \in S_k} V_k(x_0; \mathbf{c})$$

Where the norm on $\mathbf{c}$ is a sup-norm. We bias this optimization towards more regular functions by adding a penalty to the cost function. Finally, we let $\Pr(x_0 \mid lin) \sim \mathcal{U}(D)$.

C. Learning the Measurement Model

We assume a Gaussian noise model. That is

$$\Pr(\hat{x}_0 \mid x_0) \sim \mathcal{N}(x_0, \sigma_x) \quad , \quad \Pr(\hat{v}_0 \mid v_0) \sim \mathcal{N}(v_0, \sigma_v).$$

Therefore, our model is parametrized by the standard deviations σ_x and σ_v. We assume that the true trajectory of an agent is smooth compared to the noisy output of our measurement device. This justifies smoothing the trajectories, and using the difference between the smoothed signals and the raw data to learn the variance σ_x. To obtain the results in this paper we have used a moving average of four time steps (this is 0.13 seconds in realtime). We set $\sigma_v = 2\sigma_x/\Delta t$ where $\Delta t > 0$ is the time-step size. This choice is justified from the our use of finite differences to estimate velocity. In particular, if velocity is approximated via finite differencing as $v(t) \approx (x(t+h) - x(t))\,\Delta t^{-1} + \mathcal{O}(h)$ and the measurements are corrupted by Gaussian noise, then the measurement $\hat{v}(t)$ is related to $v(t)$ by Gaussian noise with roughly the same standard deviation as $(x(t+h) - x(t))\,\Delta t^{-1}$.

D. Learning the Noise Model

Finally, we assume that the true position is related to the model by Gaussian noise with a growing variance. In particular, we assume $\Pr(x_t \mid \check{x}_t) \sim \mathcal{N}(\check{x}_t, \kappa t)$ for some constant $\kappa \geq 0$. The parameter, κ, must be learned. For each curve in S_k we create a synthetic curve using the initial position and speed and integrating the corresponding vector-field, $s\,X_k$. So for each curve, $x_i(t)$, of S_k, we have a

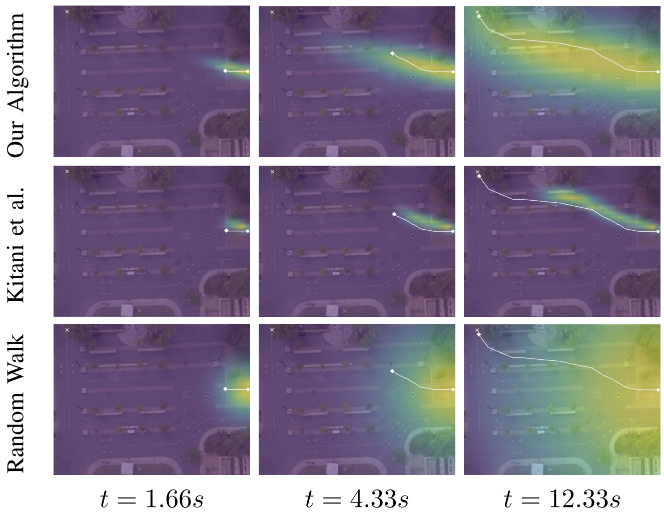


Fig. 2: An illustration of the predictions generated by the various algorithms. Notice that the Random Walk is imprecise, while the predictions generated by the algorithm in [7] suffer from the inability of their motion model to adequately match the speed of the agent.

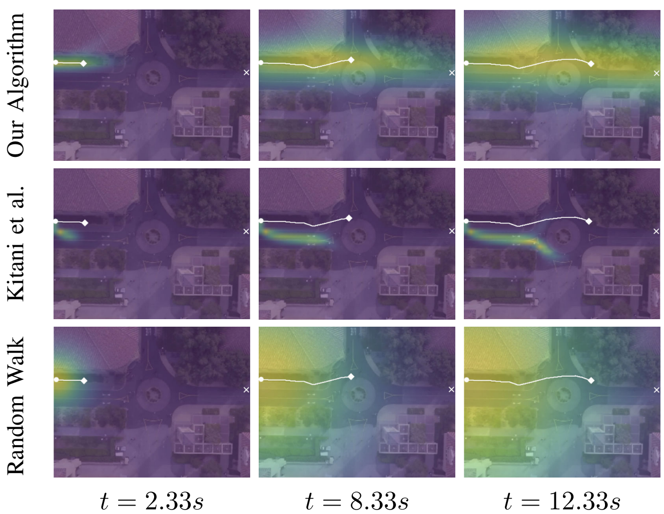


Fig. 3: An illustration of the predictions generated by the various algorithms. Notice that the Random Walk is imprecise, while the predictions generated by the algorithm in [7] are unable to match the speed of the agent and choose the wrong direction to follow the agent around the circle.

synthesized curve $x_{i,synth}(t)$ as well. We then measure the standard deviation of $(x_i(t) - x_{i,synth}(t))/t$ over i and at few time, $t \in \{0, 100, 200\}$ in order to obtain κ.

E. Evaluating Performance

This section establishes our methods performance and compares it to the model from [7] and a random walk. We implement our model as well as our evaluation code in Python 2.6. Our implementation is available online[2]. Our test data used the bicycle annotations in [13]. We did a 2-fold cross validation on 4 different scenes of varying complexity. Our analysis used the same partitions of the trajectories for all three predictors.

[2]https://github.com/OkayHughes/iros2017_pedestrian_forecasting

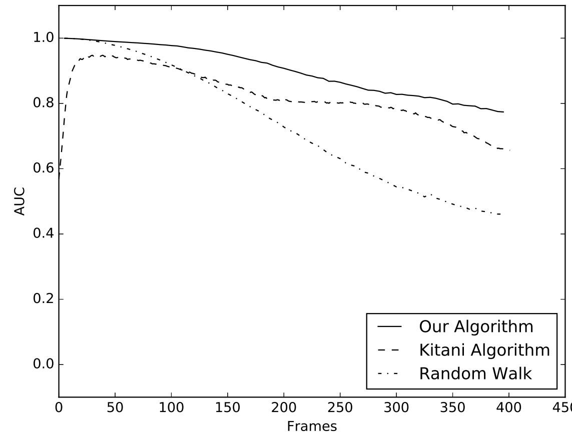


Fig. 4: A comparison of the AUC of the various algorithms. Note that the initial dip in the performance of [7] is due to their confidence in their initial estimate.

TABLE I: Comparison of runtimes of the various algorithms.

	Our Predictor	Random Walk	Kitani et al.
$\frac{\text{time}}{\text{frame}}$	0.0169s	2.1E-7s	0.0614s

Note that the implementation of the algorithm in [7] required the endpoint of each test trajectory. Without this information the implementation of the predictor provided by the authors devolved into a random walk. Neither our algorithm nor the random walk were provided this information.

The output distributions of the three algorithms were compared using their integrals over the cells of a regular grid over our domain. These integrals are used to visualize the distributions in Figures 3 and 2. Because our predictions all share the same scale, we were able to amalgamate all of the prediction and truth values for all of the simulated agents at a given time step and generate ROC curves. Figure 4 shows the Area Under the Curve of these ROC curves versus time. Our predictor behaves better than any of the other compared methods at moderately large t. This is despite providing the algorithm in [7] with the specific final location of each agent.

The run time per frame for each algorithm was generated using total run time for 400 frames, averaged across several agents and scenes, divided by the number of frames. This is shown in Table I. Our algorithm implementation leveraged its embarrassing parallelism using a relatively naive Python implementation, which split the computation of frames among 18 cores. The algorithm in [7] was timed with minimal modification using the source code provided by the authors.

V. Conclusion

Matt can you please take a crack at this .... Avenues of improvement: update predictions when given additional measurements

The conclusion goes here.

ACKNOWLEDGMENTS

Kris Kitani was critical in aiding the comparison to his algorithm.

REFERENCES

[1] D. Helbing, "A fluid-dynamic model for the movement of pedestrians," *Complex Systems*, vol. 6, pp. 391–415, 1992.

[2] R. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[3] N. Schneider and D. M. Gavrila, "Pedestrian path prediction with recursive bayesian filters: A comparative study," in *Proc. of the German Conference on Patter Recognition*, 2013.

[4] *Context-Based Pedestrian Path Prediction*, 2014.

[5] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI*, 2008, pp. 1433–1438.

[6] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *IROS*, 2009.

[7] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, *Activity Forecasting*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 201–214.

[8] D. Xie, S. Todorovic, and S. Zhu, "Inferring "dark energy" and "dark matter" from image and video," in *Proc. Int'l Conference on Computer Vision*, 2013.

[9] V. Karasev, A. Ayvaci, B. Heisele, and S. Soatto, "Intent-aware long-term prediction of pedestrian motion," *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2016.

[10] A. Doucet, N. d. Freitas, K. P. Murphy, and S. J. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, ser. UAI '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 176–183. [Online]. Available: http://dl.acm.org/citation.cfm?id=647234.720075

[11] L. Ballan, F. Castaldo, A. Alahi, F. Palmieri, and S. Savarese, "Knowledge transfer for scene-specific motion prediction," in *Proc. of European Conference on Computer Vision (ECCV)*, Amsterdam, Netherlands, October 2016. [Online]. Available: http://arxiv.org/abs/1603.06987

[12] J. Walker, A. Gupta, and M. Hebert, "Patch to the future: Unsupervised visual prediction," in *Computer Vision and Pattern Recognition*, 2014.

[13] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, *Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes*. Cham: Springer International Publishing, 2016, pp. 549–565. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46484-8_33

[14] R. Abraham, J. E. Marsden, and T. S. Ratiu, *Manifolds, Tensor Analysis, and Applications*, 3rd ed., ser. Applied Mathematical Sciences. Spinger, 2009, vol. 75.

[15] R. J. LeVeque, *Numerical methods for conservation laws*, 2nd ed., ser. Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 1992. [Online]. Available: http://dx.doi.org/10.1007/978-3-0348-8629-1

[16] D. Gottlieb and J. Hesthaven, "Spectral methods for hyperbolic problems," *Journal of Computational and Applied Mathematics*, vol. 128, no. 1–2, pp. 83 – 131, 2001, numerical Analysis 2000. Vol. VII: Partial Differential Equations. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042700005100

[17] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.