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 acting in human-centric environments should be able to anticipate the behavior of neighboring humans to ensure safe operation. Anticipating the future locations of individuals within a scene, for instance, is critical to synthesizing safe controllers for autonomous vehicles. To aid during safe control design for such systems, a prediction algorithm must satisfy several important criteria. First, since safety is paramount, the prediction algorithm should avoid misclassifying areas as unoccupied. Second, the predictions must be generated at faster than real-time rates to ensure that they can be used within a control loop. Finally, long-time horizon forecasts are preferable since they can make planning over long-time horizons feasible, which can reduce the conservativeness and aggressiveness of controllers [].

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 **??**, this approach relies upon ... The presented approach significantly outperforms...

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

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

Fig. 1: Our algorithm captures the most probable routes that a pedestrian can take and discerns which are the most probable in real time. In this example, our algorithm took 0.0158s per frame. 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.

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

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

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.

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:

- We describe our motion model as a Bayesian network in §II.
- We then describe how to compute probability densities for an agent's position efficiently in §III.
- Finally, we demonstrate the model by training and testing it on the Stanford drone dataset [13] in §IV.

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:

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

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:

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

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, and it has the further advantage of being the (local) generator of solutions to an optimal navigation problem (see Appendix A). 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, which takes $\check{x}_0$ and outputs $\check{x}_t$, 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:

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

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:

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

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

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

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:

$$\begin{tikzcd}
 & M \arrow[r] \arrow[d] \arrow[dr] & x_0 \arrow[r] \arrow[d] \arrow[dl] & \hat{x}_0 \\
x_t & \check{x}_t \arrow[l] & v_0 \arrow[l] \arrow[r] & \hat{v}_0
\end{tikzcd}. \tag{6}$$

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

$$\rho_t(x_t) := \Pr(x_t \mid \hat{x}_0, \hat{v}_0) \tag{7}$$
$$= \left(\sum_k \int \Pr(x_t, k, s \mid \hat{x}_0, \hat{v}_0) ds \right) \tag{8}$$
$$+ \Pr(x_t, lin \mid \hat{x}_0, \hat{v}_0). \tag{9}$$

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 were 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 ρ quickly and accurately. In particular, we illustrate how to compute

[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 $\mathbb{R}^n \to \mathbb{R}^n$.

an accurate approximation of the the posterior $\Pr(x_t, k, s \mid \hat{x}_0, \hat{v}_0)$ by integrating the vector-fields $X_1, \ldots, X_n$ on a fixed grid.

To begin, from (6) notice that

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

We see from the last line that $\Pr(x_t, k, s, \mid \hat{x}_0, \hat{v}_0)$ is proportional to 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:

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

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

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

where $\Psi(\hat{v}_0; k, s, x_0) := \Pr(\hat{v}_0 \mid v_0)|_{v_0 = sX_k(x_0)}$. Assuming the density $\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0; k, s, x_0)$ is of bounded variation in the variable x_0 (with all other variables held fixed), we may approximate it as a sum of weighted Dirac-delta distributions supported on a regular grid, $\Gamma := \cup_\alpha \{x_0^\alpha\}$, I think defining the indexing set of α would be useful especially in the algorithm with spacing given by Δx [15]. In other words:

$$\begin{aligned}
&\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0; k, s, x_0) = \\
&\left(\sum_\alpha c_{k,s,\alpha}\delta(x_0 - x_0^\alpha)\right) + \varepsilon_0(x_0)
\end{aligned} \tag{18}$$

for constants:

$$c_{k,s,\alpha} = [\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0 \mid k, s, x_0)|\Delta x|]|_{x_0 = x_0^\alpha} \tag{19}$$

this definition of constants is confusing to me, specifically the first term: $[\Pr(\hat{x}_0, x_0, k, s)\Psi(\hat{v}_0 \mid k, s, x_0)]\,|_{x_0 = x_0^\alpha}$ seems to be circularly defined and error of magnitude $\|\varepsilon_0\|_{L^1} \sim \mathcal{O}(|\Delta x|)$ with respect to the L^1-norm in x_0. The coefficients, $c_{k,s,\alpha}$, can be computed efficiently as product of the posteriors represented in our Bayesian network (6). Substitution of (18) into the final line of (17) yields:

$$\begin{aligned}
\Pr(\check{x}_t, \hat{x}_0, \hat{v}_0, k, s) = \sum_\alpha c_{k,s,\alpha}\delta\left(\check{x}_t - \Phi_{k,s}(x_0^\alpha)\right) + \\
+\varepsilon_t(\check{x}_t)
\end{aligned} \tag{20}$$

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 is computable by flowing the points of the grid, x_0^α, by the evolution of 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: it would be useful if we had a sentence here in non-technical terms to describe why this result is useful

Theorem 1: The error term, ε_t, is of size $\mathcal{O}(\Delta x)$ in the L^1-norm, for fixed $k, s, \hat{x}_0$, and $\hat{v}_0$. Moreover, the magnitude 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 $\mathcal{O}(\Delta x)$ 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

$$\sum_\alpha c_{k,s,\alpha} \Pr(x_t \mid \check{x}_t)|_{\check{x}_t = \Phi^t_{k,s}(x_0^\alpha)} \tag{21}$$

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

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

$$\begin{aligned}
&\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0) \\
&\quad = \sum_\alpha c_{k,s,\alpha} \Pr(x_t \mid \check{x}_t)|_{\check{x}_t = \Phi^t_{k,s}(x_0^\alpha)} + \tilde{\varepsilon}_t(x_t)
\end{aligned} \tag{22}$$

where the error term is

$$\tilde{\varepsilon}_t(x_t) = \int \Pr(x_t \mid \check{x}_t)\varepsilon_t(\check{x}_t)d\check{x}_t \tag{23}$$

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

$$\begin{aligned}
\|\tilde{\varepsilon}_t\|_{L^1} &= \int \left| \int \Pr(x_t \mid \check{x}_t)\varepsilon_t(\check{x}_t)d\check{x}_t \right| dx_t && (24)\\
&\leq \int \Pr(x_t \mid \check{x}_t)|\varepsilon_t|(\check{x}_t)d\check{x}_t\, dx_t && (25)
\end{aligned}$$

Algorithm 1 Algorithm to Compute ρ_t.

```
Require: what are the things that a user needs to specify? Like they need the vector fields and $\Gamma$?
1: for each $k \in \mathcal{M}$ having the indexing set over $\alpha$ would be useful here...also how are you discretizing the $s$ space? do
2:     Compute the constants $c_{k,s,\alpha}$ (19) .
3:     Compute $\Phi^t_{k,1}(x^\alpha_0)$ for each $k$ and each point $x^\alpha_0$ in the grid over a time-interval $[-T, T]$ .
4:     Construct $\Phi^t_{k,s}(x^\alpha_0)$ from $\Phi^t_{k,1}(x^\alpha_0)$ (Theorem 2).
5:     Construct an $\mathcal{O}(|\Delta x|)$ approximation of $\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0)$ (Corollary 1).
6: end for
7: Compute $\Pr(x_t, \hat{x}_0, \hat{v}_0)$ using (6) and (9).
8: Let $\rho_t(x_t) = \Pr(x_t, \hat{x}_0, \hat{v}_0) / \int \Pr(x_t, \hat{x}_0, \hat{v}_0) dx_t$.
```

Implementing the integration over x_t first yields:

$$\|\tilde{\varepsilon}_t\|_{L^1} \leq \int |\varepsilon_t|(\check{x}_t) d\check{x}_t =: \|\varepsilon_t\|_{L^1} \tag{26}$$

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

Corollary 1 justifies using (21) 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 coefficient $c_{k,s,\alpha}$ and the points $\Phi^t_{k,s}(x^\alpha_0)$ for all k, s and points $x^\alpha_0 \in \Gamma$. We can reduce this burden further by exploiting the following symmetry:

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. ■

Thus, computation of $\Phi^t_{k,s}(x^\alpha_0)$ for all s and t requires only computing $\Phi^t_{k,1}(x^\alpha_0)$ for all t.

Using these results, Algorithm 1 describes how to compute an $\mathcal{O}(\Delta x)$ approximation of $\rho_t(x_t)$. Steps 2 and 3 are where the bulk of the work occurs. Fortunately, these two steps are embarrassingly parallel and yield tolerable computational complexity for real-time computation. For fixed k and α, the computation of $\Phi^t_{k,1}(x^\alpha_0)$ on the interval $[-T, T]$ takes $\mathcal{O}(T)$ time using an explicit finite difference scheme and can be done in parallel for each $k \in \mathcal{M}$ and $\alpha \in$ define the indexing space here... Similarly, computation $c_{k,s,\alpha}$ constitutes a series of parallel function evaluations over triples (k, s, α) what is the indexing space of s. If the posteriors represented by the arrows in (6) are efficiently computably then the computation of all coefficients $c_{k,s,\alpha}$ is equally efficient. We summarize this in the following theorem:

Theorem 3: Assuming we have an accurate scheme for integrating ODEs in a computation time proportional to the integration time-interval[2]. Algorithm 1 yields an $\mathcal{O}(\Delta x)$ accurate approximation of ρ_t in $\mathcal{O}(Tn|\Gamma|/N)$ time given N processing units and n vector fields.

[2]For example, 4th order explicit Runge-Kutta

IV. Implementation and Experimental Results

Given this model, we describe a specific implementation and the process of fitting the model to a dataset lets be more explicit all you are fitting is the VF?. 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 appropriately again what parameters are being tuned explicitly.

A. Learning the Vector Fields

Before we begin to learn vector-fields, we must learn 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 citation. 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 are these the same α from earlier? If not please change to avoid confusion. In our case, we choose L_α to be a set of low degree Legendre polynomials. Θ_k can be learned by computing the velocities observed in the cluster, S_k. These velocities may be 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 begin by considering the nonlinear models first. We make the modeling assumption 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. Other reasonable choices (such as $\Pr(k) \propto |S_k|$) could work, but we have chosen to be conservative in this paper.

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 from 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 can also 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 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.

Our Algorithm
Kitani et. al.
Kitani et. al.
./figures/comparison/ours/comparison_2_t=70
./figures/comparison/kit/comparison_2_t=70
./figures/comparison/rand/comparison_2_t=70
./figures/comp

(a) $t = 70$ (b) $t = 250$ (c) $t = 370$

E. *Evaluating Performance*

In this section we establish our methods for evaluating the quality of our predictions, and comparing them against the model from [8] and a random walk. We implemented our model as well as the evaluation code in Python 2.6. Our implementation is available online [3].

	Our Predictor	Random Walk	Kitani et. al.
time frame	0.0169s	flop	0.0614s

We implement our model as well as the evaluation code in Python 2.6. Our implementation is available online [4] owen can you please post this to a github that is distinct from hoj's, which we've used for paper writing and everything.... Our data came from the Stanford Drone Dataset, in the form of hand annotated bounding boxes around agents (e.g. cars, pedestrians, bicycles). Our datasets used the bicycle data. We did a 10-fold cross validation on 4 different scenes of varying complexity. Our analysis used the same partitions of the trajectories for all three predictors. There .

Our data came from the Stanford Drone Dataset, in the form of hand annotated bounding boxes around agents (e.g. cars, pedestrians, bicycles). Our datasets used the bicycle data. We did a 2-fold cross validation on 4 different scenes

[3]https://github.com/hoj201/pedestrian_forecasting
[4]https://github.com/hoj201/pedestrian_forecasting

TABLE I: Runtimes

	Our Predictor	Random Walk	Kitani et. al.
$\frac{\text{time}}{\text{frame}}$	0.0169s	flop	0.0614s

of varying complexity. Our analysis used the same partitions of the trajectories for all three predictors.

The algorithm provided by Kitani et. al. was given additional information when making predictions that neither ours nor the random walk model were provided, namely the endpoint of the test trajectory. Without these data the implementation of the predictor provided by the authors devolves into a random walk.

figure All three algorithms side by side captionThe grid used to evaluate the predicted distributions. The circle denotes the start of a trajectory and the X denotes the end.

The output distributions of the three algorithms were compared using their integrals over the cells of a suitably fine regular grid over our domain. These integrals can be seen in figure n. In keeping with our measurement model, the ground truth compared against was a grid that was identically zero except on a bounding box around the measurement at time t. The dimensions of the bounding box were the average bounding box dimensions in the training data set.

figureplot

Disregarding small time scales where any algorithm will be reasonably accurate, our predictor behaves much better than any of the others at moderately large t, where we capture the behavior of the agent without suffering the misplaced confidence observed in Kitani et. al.'s predictions. Their encoding of velocity results in the prediction lagging behind the actual observed motion. Even given significantly better information, Kitani et. al. generates lower quality predictions than our algorithm at all time scales.

The running time per frame for each algorithm was generated using run times for 400 frames, averaged across several agents and scenes, shown in Table I. Our algorithm implementation leveraged its embarassing parallelism using a relatively naive parallelization scheme, which split the computation of frames among 18 cores. It's highly probable that our run times would significantly decrease using an implementation of our algorithm in C that uses a more clever scheme. Kitani et. al. was timed with minimal modification to the source code provided by the authors.

We see that under our current implementation, we are beating Kitani et. al.'s algorithm at the relevant time scales, and are running more than 3 times faster that their algorithm. More significantly, our algorithm runs at almost 60FPS. Figure IV-E shows that at all time scales, our algorithm has very good predictive capability, and Figure **??** shows the qualitative merit of our predictions on several scenes.

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

Thanks to Kris Kitani for being nice. No thanks to Stanford, for being dicks.

Appendix

A. Inverse Optimal Control

One of the valuable aspect of the motion model of [7] was that the trajectories were solutions of optimal control problems. This is a desirable property for a motion model of pedestrians/cyclists/vehicles in that it seems reasonable to assume that we all move through the world with an intent to get somewhere. This framework was dubbed *inverse optimal control* (IOC). Here we illustrate how solutions of (2) can fit within the IOC framework.

Theorem 4: Let X be a vector-field such that $\|X(x)\|_2 = 1$ for all $x \in \mathbb{R}^2$. Then, for each initial condition $x_0 \in \mathbb{R}^2$, there exists a neighborhood U, a Riemannian metric, g, and a cost function, f, such that solutions of (2) are also solutions of the optimal control problem

$$p^* = \inf_{\|v_t\|_g = s} \int_0^T f(x_t)dt \tag{27}$$

where $v_t = \frac{dx_t}{dt}$, $\|\cdot\|_g$ is the norm with respect to g and the infimum is taken over all differentiable curves in U which originate from x_0.

Proof: By rescaling time, we may let s be any positive real number. Therefore, without loss of generality, we will seek an f and g such that $s = 1$. In this case a solution of (27) is generated by the vector-field $X = \frac{\nabla f}{\|\nabla f\|}$ where ∇ is the Riemannian gradient with respect to some metric (possibly non-Euclidean). Therefore, our goal is to illustrate that there exists a metric g and a function f such that the given X is given by $\frac{\nabla f}{\|\nabla f\|}$.

Because $\|X\| = 1$ globally, it has no fixed points. From the flow-box theorem [14, Theorem 4.1.14] we can assert that for any open set, $U \subset \mathbb{R}^2$ there exists a local diffeomorphism which transforms X_k into a flat vector-field. Mathematically, this asserts the existence of a map $\Phi : U \to V \subset \mathbb{R}^2$ such that the push-forward of X, which we will denote by $\tilde{X}$, and given by

$$\tilde{X}(\tilde{x}) = \Phi_* X(\tilde{x}) := D\Phi|_{\Phi^{-1}(\tilde{x})} \cdot X(\Phi^{-1}(\tilde{x})),$$

is just the flat vector field $(1, 0)$ for all $x \in U$. We could view the coordinate function $\tilde{f}(x) = x^0$ as a cost function on V, and we may set g_V to be equal to the standard flat metric on V inherited from $\mathbb{R}^2$. In this case we observe that $\tilde{X}$ generates solutions to the optimization problem

$$\tilde{p}^* = \inf_{\|\tilde{v}_t\|_{g_V} \leq 1} = \int_0^T \tilde{f}(\tilde{x}_t)dt$$

By a change of variables, it follows that the original vector field, X, then solves the optimization problem (27) where $g = \Phi^* g_V$ is the pull-back metric, and $f = \Phi^* \tilde{f}$ is the pull-back of $\tilde{f}$. ■

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] W. Rudin, *Functional analysis. International series in pure and applied mathematics*. McGraw-Hill, Inc., New York, 1991.