

Real-Time Probabilistic Pedestrian Forecasting

Author Names Omitted for Anonymous Review. Paper-ID [add your ID here]

Abstract—The abstract goes here.

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 [1].

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 algorithms are well characterized by the underlying evolution model they adopt. Such models come in a variety of flavors, and are adapted to the task at hand. For example, see [3] for crowd modeling.

In this article we are primarily concerned with models that probabilistically predict the position of a single agent. The most simple approach to forecasting would be to forward integrate a Kalman filter [4] based upon the observed heading. Over short time scales this may perform quite well, but the resulting distribution will devolve into a large Gaussian mass over longer time scales. To mitigate the negative effects of this simplistic description of motion, slightly more sophisticated motion models should be considered. Such models include inverse optimal control (IOC) based models such as [7, 8, 6]. These IOC models have the often desirable property of attributing intention and goal-seeking behavior to the agents. For example Kitani et al. [6] extracted a very simple Markov decision process (MDP) evolving on a 2D lattice by training on a small collection of features and trajectories. The resulting MDP is very light weight, parametrized by only a small collection of coefficients equal in number to that of the feature maps. Secondly, given an initial and final state, the probability distribution of the pedestrian at intermediate states is easily computable because it basically comes down to matrix multiplication. Thirdly, the small collection of parameters

learned in one environment can, in principal, be used in a novel environment without retraining (i.e. “knowledge transfer”). Lastly, the IOC-MDP framework is very flexible and very easy to generalize. For example, time dependent information, such as traffic signals, are incorporated in [5], by relaxing the Markov property and considering a switched Markov process. Karasev et al. [5] generalizes [6] further by considering a discrete time Markov process over a continuous space which more closely approximates reality. However, this modelling choice resulted in posteriors which were difficult to compute with precision. They opted to use a sample based method: a Rao-Blackwellized particle filter. The resulting accuracy is known in a probabilistic sense in that the error bounds are themselves random variables. Moreover, high accuracy can come at a high computational cost prohibitive for real-time computation. It should be noted, however, that speed was not a goal of [5].

One difficulty which IOC models can suffer, is the tendency for the trajectories to become trapped in local optimal. This often occurs when agents make very sharp turns, and have intermediate goals to go one place before visiting another. To address this, Ballan et al. [2] adopted an empiricists approach, computing “turning maps” and attempting to infer how agents behave in a given patch based on its feature, solely based on how previous agents have behaved there (regardless of operating under an unobserved optimization criteria). Similar to [5], the motion model was a discrete time Markov process over a continuous space and so the relevant posteriors were approximated using sample based techniques. Again, the same criticisms in regards to speed and accuracy lodged at [5] can also be lodged at [2], and again, we should note that speed/high accuracy were not a priority there. However, the motion model of Ballan et al. [2] was merely an illustrative choice in the more ambitious pursuit of creating a flexible paradigm for pedestrian forecasting. In particular, a motion model on one scene ought to be transferable to other scenes, and the representation of the “objects” in the scene ought to be flexible in order for behavior to be robust. Said differently, the way a forecasting algorithm executes “knowledge transfer” should not depend too rigidly on the finitely many labels some engineer managed to think of in a late-night brainstorming session. The 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. In summary, the motion model used in [2] is merely a parameter of the more general framework they have proposed. We should note that the motion model we will present here can, in principal, be incorporated within their forecasting paradigm.

Finally,

B. Contributions

The primary contributions of this paper are:
The rest of the paper is organized as follows:

II. MODEL

ramv: I would include far fewer model details here just to illustrate that our focus is on efficient computation for general models.

Our 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 $\rho_t(x_t) := \Pr(x_t | \hat{x}_0, \hat{v}_0)$.

In order to derive an expression for ρ we will 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. This allows our model to 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) The 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_0, v_0 .
- 3) The motion model: This is represented by a trajectory \tilde{x}_t and depends on a variety of other variables which will be described shortly.

We will now elaborate on these three components, and relate them to one another.

A. The Sensor Model

At time $t = 0$, we obtain a noisy reading of position, $\hat{x}_0 \in \mathbb{R}^2$. We assume that our measurements come from Gaussian noise about the true location $x_0 \in \mathbb{R}^2$. In other words $\Pr(\hat{x}_0 | x_0) = G_{\sigma_x}(\hat{x}_0 - x_0)$ where G_σ denotes the bivariate Gaussian distribution. We use the same sensor model is assumed for velocity in that $\Pr(\hat{v}_0 | v_0) = G_{\sigma_v}(\hat{v}_0 - v_0)$. In principal, any measurement model could be used, and we may view this as a parameter of our method.

B. 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 blurring the result in order to acknowledge inevitable modeling errors.

Specifically, our motion model consists of a modeled trajectory \tilde{x}_t , which is probabilistically related to the true position by x_t via

$$\Pr(x_t | \tilde{x}_t) \sim \mathcal{N}(\tilde{x}_t, \sigma_{int}(t)),$$

where $\sigma_{int}(t) = \kappa t$ for some learned constant $\kappa > 0$. This insures the consistency condition $\tilde{x}_0 = x_0$, while acknowledging the existence of uncertainty for all $t > 0$.

Once initialized, agents come in two flavors: linear and nonlinear. Linear agents evolve in time according to the equation $\tilde{x}_t = x_0 + tv_0$ and so we have the posterior

$$\Pr(\tilde{x}_t | x_0, v_0, lin) = \delta(\tilde{x}_t - x_0 - tv_0).$$

We will assume linear agents also satisfy the posteriors

$$\Pr(x_0 | lin) \sim \mathcal{U}(D) \quad , \quad \Pr(v_0 | lin) \sim \mathcal{N}(0, \sigma_L).$$

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

$$\frac{d\tilde{x}_t}{dt} = s \cdot X_k(\tilde{x}_t) \quad (1)$$

where X_k is a vector-field from a finite collection $\{X_1, \dots, X_n\}$, 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). The vector-fields X_1, \dots, X_n are learned from the data-set (see §IV).

It is assumed that k and s are both constant in time, so that \tilde{x}_t is determined from the triple (x_0, k, s) by integrating (1) with the initial condition x_0 . In other words, we have the posterior

$$\Pr(\tilde{x}_t | x_0, k, s) = \delta(\tilde{x}_t - \Phi_{k,s}^t(x_0)) \quad (2)$$

where $\Phi_{k,s}^t$ is the flow-map of the vector field sX_k up to time t . It is notable, that the variables k, s and x_0 determine v_0 . Thus we also have the posterior

$$\Pr(v_0 | k, s, x_0) = \delta(v_0 - sX_k(x_0)). \quad (3)$$

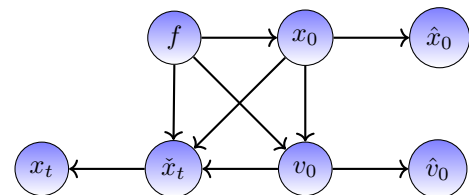
In summary, the agent flavors are parametrized by the set

$$F = \{lin\} \cup (\mathbb{R} \times \{1, \dots, n\}) \quad (4)$$

and each flavor determines the sort of motion we should expect from that agent.

C. The Full Model

Concatenating the measurement model with our motion model yields the Bayesian network, where f is an element of F from (4) and denotes the flavor of the agent:



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

$$\begin{aligned}\rho_t(x_t) &:= \Pr(x_t \mid \hat{x}_0, \hat{v}_0) \\ &= \left(\sum_k \int \Pr(x_t, k, s \mid \hat{x}_0, \hat{v}_0) ds \right) \\ &\quad + \Pr(x_t, \text{lin} \mid \hat{x}_0, \hat{v}_0).\end{aligned}$$

The final term $\Pr(x_t, \text{lin} \mid \hat{x}_0, \hat{v}_0)$ is expressible in terms of the error function, and would pose a negligible burden in terms of numerical computation. The primary computational burden derives from the sum $\sum_k \int \Pr(x_t, k, s \mid \hat{x}_0, \hat{v}_0) ds$.

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. In this section we give some details on how this modeling choice can be leveraged to compute ρ quickly, and accurately. In particular, we illustrate how to compute an accurate approximation of the the posterior $\Pr(x_t, k, s \mid \hat{x}_0, \hat{v}_0)$.

To begin, from the graphical model we can see that

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

As $\Pr(x_t \mid \tilde{x}_t) \sim \mathcal{N}(\tilde{x}_t; \kappa t)$, we see from the last line that $\Pr(x_t, k, s \mid \hat{x}_0, \hat{v}_0)$ is proportional to a Gaussian convolution of the joint distribution $\Pr(\tilde{x}_t, \hat{x}_0, \hat{v}_0, k, s)$. Assuming such a convolution can be performed efficiently (which we will address later), we can focus on computation of $\Pr(\tilde{x}_t, \hat{x}_0, \hat{v}_0, k, s)$.

Again, from the PGM we see that

$$\Pr(\tilde{x}_t, \hat{x}_0, \hat{v}_0, k, s) = \int \Pr(\tilde{x}_t, x_0, \hat{x}_0, v_0, \hat{v}_0, k, s) dx_0 dv_0 \quad (5)$$

$$= \int \Pr(\tilde{x}_t \mid x_0, k, s, v_0) \Pr(\hat{v}_0 \mid v_0) \Pr(v_0 \mid k, s, x_0) \quad (6)$$

$$\Pr(\hat{x}_0, x_0, k, s) dx_0 dv_0 \quad (7)$$

$$(8)$$

Substitution of (2) and (3) yeilds

$$= \int \delta(\tilde{x}_t - \Phi_{k,s}^t(x_0)) \delta(v_0 - sX_k(x_0)) \quad (9)$$

$$\Pr(\hat{v}_0 \mid v_0) \Pr(\hat{x}_0, x_0, k, s) dx_0 dv_0 \quad (10)$$

Carrying out the integration over v_0 we observe

$$\begin{aligned}\Pr(\tilde{x}_t, \hat{x}_0, \hat{v}_0, k, s) \\ = \int \delta(\tilde{x}_t - \Phi_{k,s}^t(x_0)) \Pr(\hat{x}_0, x_0, k, s) G_{\sigma_v}(\hat{v}_0 - sX_k(x_0)) dx_0\end{aligned} \quad (11)$$

As the density $\Pr(\hat{x}_0, x_0, k, s) G_{\sigma_v}(\hat{v}_0 - sX_k(x_0))$ is Riemann integrable in x_0 , we may approximate it as a sum of weighted Dirac delta's supported on a regular grid, $\Gamma = \cup_\alpha \{x_0^\alpha\}$ with spacing Δx .

Perhaps prove this claim in the appendix.

In other words

$$\begin{aligned}\Pr(\hat{x}_0, x_0, k, s) G_{\sigma_v}(\hat{v}_0 - sX_k(x_0)) = \\ \left(\sum_\alpha c_{k,s,\alpha} \delta(x_0 - x_0^\alpha) \right) + \varepsilon_0(x_0)\end{aligned} \quad (12)$$

for constants

$$c_{k,s,\alpha} = [\Pr(\hat{x}_0, x_0, k, s) G_{\sigma_v}(\hat{v}_0 - sX_k(x_0)) \Delta x^2] \big|_{x_0=x_0^\alpha} \quad (13)$$

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 appearing in the PGM. Substitution of (12) into the final line of (11) yields

$$\Pr(\tilde{x}_t, \hat{x}_0, \hat{v}_0, k, s) = \sum_\alpha c_{k,s,\alpha} \delta(\tilde{x}_t - \Phi_{k,s}(x_0^\alpha)) + \varepsilon_t(\tilde{x}_t)$$

where $\varepsilon_t(\tilde{x}_t) = \int \delta(\tilde{x}_t - \Phi_{k,s}^t(x_0)) \varepsilon_0(x_0) dx_0$. 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 $\Pr(\tilde{x}_t, \hat{x}_0, \hat{v}_0, k, s)$ as a sum of Dirac-deltas is adaptive, in that the error term does not grow in total mass.

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 $\Phi_{k,s}^t$ by Φ . We observe

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

As ε_0 is of magnitude $\mathcal{O}(\Delta x)$ the result follows. \square

As the Gaussian convolution of a Dirac-delta distribution is a Gaussian distribution, the following corollary is nearly immediate.

Corollary 1. *Let $G_\sigma(x - \mu)$ denote the probability density of a bivariate normal distribution of standard deviation σ and mean μ . Then the density*

$$\sum_\alpha c_{k,s,\alpha} G_{\kappa t}(x_t - \Phi_{k,s}^t(x_0^\alpha)) \quad (14)$$

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. Recall that $\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0)$ is related to $\Pr(\tilde{x}_t, k, s, \hat{x}_0, \hat{v}_0)$ by convolving the latter density in the variable \tilde{x}_t with a Gaussian kernel of variance κt . That is to say

$$\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0) = \int G_{\kappa t}(x_t - \tilde{x}_t) \Pr(\tilde{x}_t, k, s, \hat{x}_0, \hat{v}_0) d\tilde{x}_t$$

Substitution of (12) yields

$$\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0) = \sum_{\alpha} c_{\alpha} G_{\kappa t}(x_t - \Phi_{k,s}^t(x_0^{\alpha})) + (G_{\kappa t} * \varepsilon_t)$$

Gaussian convolution preserves the L^1 -norm of a density. Therefore $\|G_{\kappa t} * \varepsilon_t\|_{L^1} = \|\varepsilon_t\|_{L^1} \sim \mathcal{O}(\Delta x)$, which is also constant in time by Theorem 1. \square

Corollary 1 justifies using (14) as an approximation of $\Pr(x_t, k, s, \hat{x}_0, \hat{v}_0)$. So we've reduce the problem of computing $\rho_t(x_t)$ to the problem of computing the coefficient $c_{k,s,\alpha}$ and the points $\Phi_{k,s}^t(x_0^{\alpha})$ for all k, s and points $x_0^{\alpha} \in \Lambda$. We can reduce this burden further by exploiting the following symmetry.

Theorem 2. $\Phi_{k,s}^t = \Phi_{k,1}^{st}$.

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

Thus, computation of $\Phi_{k,s}^t(x_0^{\alpha})$ boils down to computing $\Phi_{k,1}^t(x_0^{\alpha})$.

At this point, we may summarize the ingredient to compute an approximation of $\Pr(x_t, k, s | \hat{x}_0, \hat{v}_0)$ in two parts

- 1) Compute the constants c_{α} from (13).
- 2) Compute $\Phi_{k,1}^t(x_0^{\alpha})$ for each k and each point x_0^{α} in the grid over a time-interval $[-T, T]$.

Each of these pieces may be computed efficiently, and the full computation is embarrassingly parallel. For example, for fixed k and α , the computation of $\Phi_{k,1}^t(x_0^{\alpha})$ on the interval $[-T, T]$ take $\mathcal{O}(T)$ time using an explicit finite difference scheme, and computation for each (k, α) may be performed in parallel with another such pair. Moreover, computation $c_{k,s,\alpha}$ is embarrassingly parallel in with respect to all triples (k, s, α) . If the posteriors of our PGM consist solely of elementary functions such as Gaussians and polynomials, then computation of all coefficients $c_{k,s,\alpha}$ can be made virtually instantaneous given enough processors, as would be the case with a sufficiently powerful graphics processing unit. More precisely, the full computation of our approximation of $\rho_t(x_t)$ would take $\mathcal{O}(TnA/N)$ time given N processing units where $A = |\cup_{\alpha} \{x_0^{\alpha}\}|$ is the size of the mesh and n is the number of vector fields.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

Now that the model has been described, we can discuss a way, but certainly not the only way, that one can fit parameters of the model to a data set. For the purpose of demonstration, we will use the Stanford Drone Dataset [?]. More generally, we will assume that for a fixed scene we have a database of previously observed trajectories $\{\hat{x}^1, \dots, \hat{x}^N\}$. From this data we will tune the parameters of the model appropriately. The parameters of the model which must be chosen or learned are

- the vector-fields $\{X_k\}_{k=1}^n$,
- the posteriors $\Pr(x | k, s)$ for $k = 1, \dots, m$ and $s \in \mathbb{R}$,
- the posterior $\Pr(x | \text{lin})$,
- the unconditioned probabilities $\Pr(f)$ for $f \in F$,
- the standard deviations of the measurement σ_x and σ_v ,
- and the blurring parameter κ .

A. Learning the vector fields

Before we can even begin to learn vector-fields, we must learn the number of such vector-fields we have. To do this we use a simple clustering algorithm on the trajectories, to categorize them into groups. In principal, any clustering algorithm could accomplish this. For our algorithm 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. This clustering of the end-points induces a clustering of the trajectories. So we obtain clusters S_1, \dots, 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 may learn a vector-field which is approximately compatible with that set. We'd like to only consider vector-fields who's vectors have unit magnitude because we observe many of the trajectories in the data appear to have a roughly constant speed. Therefore 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 will 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've chosen L_{α} to be a set of low degree Legendre polynomials. Learning Θ_k can be obtained by looking at all 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 can 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 should also include a regularization term to bias the minimizes towards smoother outputs. Using the H^1 -norm times a fixed scalar would suffice.

hoj: There is more to say here

B. Learning the posteriors

We'd like to compute $\Pr(x_0 \mid k, s)$ for each cluster $k = 1, \dots, n$ and each speed $s \in \mathbb{R}$. We will 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 will 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 will 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 $\Pr(\text{lin}) = (n + 1)^{-1}$ as well. Other reasonable choices (such as $\Pr(k) \propto |S_k|$) could work, but we've chosen to be more conservative in this paper.

In order to learn $\Pr(x_0 \mid k)$ we will assume that it's logarithm may be approximated by a low dimensional subspace of functions on \mathbb{R}^2 . That is to say, for each k we will 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 6 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 $\Pr(x_0 \mid \text{lin}) \sim \mathcal{U}(D)$.

C. Learning the variance parameters

We will assume that the true trajectory of an agent is fairly 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 4 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 difference's 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 $(x(t+h) - x(t)) \Delta t^{-1}$.

Finally, we must learn κ , the parameter which blurs the our motion model. We've taken an ad hoc approach in this paper, although more intelligent and theoretically sound alternatives likely exist. For each curve in S_k we create a synthetic curve using the initial position and speed and integrating the corresponding vector-field, sX_k . So for each curve, $x_i(t)$, of S_k , we have a synthesized curve $x_{i,\text{synth}}(t)$ as well. We then

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

D. Precision, accuracy, etc.

V. CONCLUSION

The conclusion goes here.

ACKNOWLEDGMENTS

Thanks to Kris Kitani

APPENDIX

A. Inverse Optimal Control

One of the valuable aspect of the motion model of [6] 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 where one determines an optimal control problem from trajectories (as opposed to solving for trajectories of an optimal control problem) was dubbed *inverse optimal control* (IOC). In this optional subsection, we illustrate how solutions of 1 can fit within the IOC framework.

Theorem 3. *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 (1) are also solutions of the optimal control problem*

$$p^* = \inf_{\|v_t\|_g = s} \int_0^T f(x_t) dt \quad (15)$$

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 (15) 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 [1, 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 $\Phi : U \rightarrow V \subset \mathbb{R}^2$ 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 (15) where $g = \Phi^* g_V$ is the pull-back metric, and $f = \Phi^* \tilde{f}$ is the pull-back of \tilde{f} . \square

REFERENCES

- [1] R Abraham, J E Marsden, and T S Ratiu. *Manifolds, Tensor Analysis, and Applications*, volume 75 of *Applied Mathematical Sciences*. Springer, 3rd edition, 2009.
- [2] Lamberto Ballan, Francesco Castaldo, Alexandre Alahi, Francesco Palmieri, and Silvio Savarese. Knowledge transfer for scene-specific motion prediction. In *Proc. of European Conference on Computer Vision (ECCV)*, Amsterdam, Netherlands, October 2016. URL <http://arxiv.org/abs/1603.06987>.
- [3] D Helbing. A fluid-dynamic model for the movement of pedestrians. *Complex Systems*, 6:391–415, 1992.
- [4] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82 (1):35–45, 1960.
- [5] 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.
- [6] Kris M. Kitani, Brian D. Ziebart, James Andrew Bagnell, and Martial Hebert. *Activity Forecasting*, pages 201–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-33765-9. doi: 10.1007/978-3-642-33765-9_15.
- [7] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proc. AAAI*, pages 1433–1438, 2008.
- [8] Brian D. Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, J. Andrew Bagnell, Martial Hebert, Anind K. Dey, and Siddhartha Srinivasa. Planning-based prediction for pedestrians. In *IROS*, 2009.