Realtime Probabilistic Pedestrian Forecasting

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

***Abstract*—The abstract goes here.**

I. Introduction

hoj: maybe ram should write the first paragraph.

A. Background

Papers to mention

- Kitani et al. [5] (Early MDP based paper)
- Karasev et al. [4] (Another motion model, discrete time)
- Ballan et al. [2] (The knowledge transfer paper)
- **?**] (Another MDP paper)
- **?**] (fluid based crowd model)

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

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 according to there own noisy perception. This allows our model to be divided into three parts:

1) Reality: This is parametrized by the true position and velocity of the observed agent x_t, v_0.
2) Our sensor model: This is represented by our sensor readings $\hat{x}_0$ and $\hat{v}_0$.
3) The agent model: This is represented by a trajectory $\check{x}_t$, which is the agent's own estimate of her position, as well as numerous other variables which will be introduced in upcoming subsections.

The measurements and the agent's internal state are dependent on reality. This couples all the variables in a Bayesian network, which we will describe in the following subsections. The resulting Bayesian network can be used to compute the desired posterior.

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 $x_0 \in \mathbb{R}^2$, so that $\Pr(\hat{x}_0 \mid x_0) = (2\pi\sigma_x^2)^{-1} \exp(-\|\hat{x}_0 - x_0\|^2/(2\sigma_x^2))$. Similarly, we have a velocity reading $\hat{v}_0 \in \mathbb{R}^2$ which is conditionally dependent on the true velocity $v_0 \in \mathbb{R}^2$ via the posterior $\Pr(\hat{v}_0 \mid v_0) = (2\pi\sigma_v^2)^{-1} \exp(-\|\hat{v}_0 - v_0\|^2/(2\sigma_v^2))$. We should note that $\hat{v}_0$ is typically derived from multiple measurements of position via a finite difference. This induces a linear inequality between σ_x and σ_v. For example, a first order finite difference would yield $\sigma_v \geq 2\sigma_x/\Delta t$ where $\Delta t > 0$ is the time-step size between consecutive measurements of position.

B. The Agent Model

All agents are initialized within some rectangular region $D \subset \mathbb{R}^2$. We denote the true position and velocity of an agent by x_t and v_t. While we may mention x_t and v_t, these are never available to us and are not quantities any model should seek to predict with certainty. Any proposed model should include something to account for the fact that it is almost certainly wrong. We do this by fitting a deterministic model to the data and blurring the result in order to acknowledge such modeling errors.

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

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

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

Once initialized, agents come in two flavors: linear and nonlinear. Linear agents imagine that they move in straight lines at constant velocity. Thus $\check{x}_t = \check{x}_0 + t\check{v}_0$ and so we have the posterior

$$\Pr(\check{x}_t \mid \check{x}_0, \check{v}_0, lin) = \delta(\check{x}_t - \check{x}_0 - t\check{v}_0).$$

We will assume linear agents also satisfy the posteriors

$$\Pr(\check{x}_0 \mid lin) \sim \mathcal{U}(D) \quad , \quad \Pr(\check{v}_0 \mid lin) \sim \mathcal{N}(0, \sigma_L).$$

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

$$\frac{d\check{x}_t}{dt} \equiv \check{v}_t = s \cdot X_k(\check{x}_t) \tag{1}$$

where X_k is a vector-field from a finite collection $\{X_1, \ldots, X_n\}$, and $s \in \mathbb{R}$. This collection is learned from the data-set. It is assumed that k and s are both constant in time,

so that the position x_t is determined from the tripe $(\check{x}_0, k, s)$ by integrating (1). In other words, we have the posterior

$$\Pr(\check{x}_t \mid \check{x}_0, k, s) = \delta(\check{x}_t - \Phi^t_{k,s}(\check{x}_0))$$

where $\Phi^t_{k,s}$ is the flow-map of the vector field $s \cdot X_k$ up to time t. It is notable, that the variables k, s and $\check{x}_0$ determine $\check{v}_0$. Thus we also have the posterior

$$\Pr(\check{v}_0 \mid k, s, \check{x}_0) = \delta(\check{v}_0 - sX_k(\check{x}_0)).$$

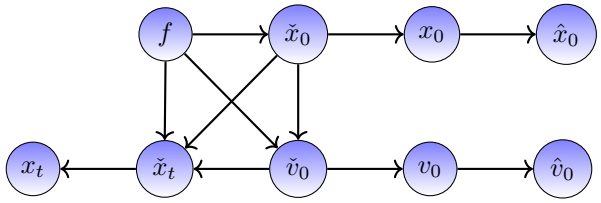
In summary, the agent flavors are parametrized by the set

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

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



Which allows us to derive ρ_t by appropriately marginalizing over the full joint probability over all 9 variables. [Reference]

The abundance of Dirac deltas and Gaussians allows the integrals involved in computing ρ_t to be solved in closed form as smooth functions over $\mathbb{R}^2$.

D. Inverse Optimal Control

ramv: I would just mention this as a property that is satisfied by our vector fields and leave the theorem for the appendix.

One of the valuable aspect of the motion model of [5] 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 1. *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*

$$p^* = \inf_{\|v_t\|_g = s} \int_0^T f(x_t)dt \tag{2}$$

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

III. Efficient Probability Propagation

ramv: I think this is the exciting part of the paper, since people struggle to advect things efficiently through nonlinear models. I'd definitely go into lots of detail here and include the theorem about the L^1 bound.

IV. Learning

ramv: I'd consider folding this section into the experiment section. This is basically how we learn all of the stuff in our model.

Now that the model has been described, we can discuss how one can fit the parameters of the model to a data set. In the results section, 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 $\Pr(x \mid k, s)$ for $k = 1, \dots, m$ and $s \in \mathbb{R}$,
- the posterior $\Pr(x \mid lin)$,
- 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 $S_1, \ldots, S_n$ consisting of trajectories from our data set, as well as a set of unclassified trajectories, S_0.

hoj:There is more to say here

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.

B. Learning the posteriors

We'd like to compute $\Pr(x_0 \mid k, s)$ for each cluster $k = 1, \ldots, n$ 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 $\Pr(lin) = (n+1)^{-1}$ 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 who's 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 $\Pr(x_0 \mid lin) \sim \mathcal{U}(D)$.

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, $s\,X_k$. 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.

V. Results

ramv: Id try to fold the learning section as a subsection in this section....

Precision, accuracy, etc.

VI. Conclusion

The conclusion goes here.

ACKNOWLEDGMENTS

REFERENCES

[1] R Abraham, J E Marsden, and T S Ratiu. *Manifolds, Tensor Analysis, and Applications*, volume 75 of *Applied Mathematical Sciences*. Spinger, 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] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82 (1):35–45, 1960.

[4] 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.

[5] 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.

[6] Tad McGeer. Passive Dynamic Walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990. doi: 10.1177/027836499000900206. URL http://ijr.sagepub.com/content/9/2/62.abstract.