2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Learning to Navigate Through Crowded Environments

Peter Henry[1], Christian Vollmer[1,2], Brian Ferris[1], and Dieter Fox[1]

[1]*University of Washington, Department of Computer Science & Engineering, Seattle, WA*
[2]*Neuroinformatics and Cognitive Robotics Lab, Ilmenau University of Technology, Germany*

***Abstract*— The goal of this research is to enable mobile robots to navigate through crowded environments such as indoor shopping malls, airports, or downtown side walks. The key research question addressed in this paper is how to learn planners that generate human-like motion behavior. Our approach uses inverse reinforcement learning (IRL) to learn human-like navigation behavior based on example paths. Since robots have only limited sensing, we extend existing IRL methods to the case of partially observable environments. We demonstrate the capabilities of our approach using a realistic crowd flow simulator in which we modeled multiple scenarios in crowded environments. We show that our planner learned to guide the robot along the flow of people when the environment is crowded, and along the shortest path if no people are around.**

I. Introduction

The goal of this research is to enable mobile robots to navigate through crowded environments such as indoor shopping malls, airports, or downtown side walks. Consider a person trying to get to a certain store in a shopping mall. The initially planned path to that store is based on the person's *apriori* knowledge about the layout of the mall and the expected crowd motion patterns in different areas of the mall. As the person moves with the flow through the mall, typically tending to move on the right, she updates her estimates of the people density and flow based on her observations. Using these updated estimates, she continuously re-assesses the appropriateness of the planned path and decides to follow a different route if necessary. She also continuously trades off crowd following behavior with the desire to reach the target, moving against the crowd flow if necessary.

Ideally, we would like robots navigating through such environments to move along paths similar to those of the people. By moving with the flow of the people, for instance, a robot could navigate more efficiently. An additional benefit of such imitative behavior would be better predictability of robot motion, thereby enabling more natural and socially acceptable interactions between people and robots [12]. While existing robot control systems deployed in crowded environments such as museums or railway stations have the capability to plan paths, update world models on the fly, and re-plan based on new information, these planning techniques solely aim at reaching a goal as efficiently as possible [3], [15], [18], [19].

The key research question addressed in this paper is how one can learn planners that generate human-like motion behavior. Our approach to this problem is to learn a motion planner from example traces of people moving through such environments. We use these example traces to learn how people trade-off different factors such as "desire to move with the flow", "avoidance of high-density areas", "preference for walking on the right/left side", and "desire to reach the goal quickly". Our learning technique is based on the following model: First, people plan paths through an environment that are optimal w.r.t. a cost function, even if they are not doing so explicitly or even consciously. Second, this cost function is a weighted combination of various environmental factors, including factors that change over time and are only partially observable (*e.g.*, crowd density). Third, a person continuously updates the values of these factors based on local perceptions and, fourth, updates the path to the goal if necessary.

To learn a path planner from example traces, we build on inverse reinforcement learning (IRL), a novel framework for learning from demonstrations [14]. IRL has been applied successfully to different problems, including learning to decide lane changing in a simulated highway scenario [2], learning to navigate a vehicle through a parking lot [1] and learning to plan routes through a street network [22]. However, these approaches assume that an agent has full access to all factors that influence its decision making. In our case, unfortunately, neither a person nor a robot have complete knowledge of the density and flow of people in the environment. Instead, these values must be estimated on the fly. In this paper, we show how to perform such an estimation using Gaussian processes [16] and we extend maximum entropy inverse reinforcement learning [22] to handle this more difficult scenario.

We evaluate our approach using the crowd motion simulator developed by Treuille and colleagues [20] (see Fig. 1). This simulator has been shown to generate realistic motion patterns based on parameterizable path cost functions. This simulator enables us to perform controlled experiments that are realistic enough to be transferable to real world scenarios. The experiments demonstrate that our approach is able to successfully learn navigation paths from demonstrations.

This paper is organized as follows. After discussing related work, we present our approach to learning navigation behavior from demonstrations. Then, in Section IV, we describe how Gaussian process regression can be used to estimate dynamic environmental features. Experimental results are presented in Section V, followed by a discussion.

978-1-4244-5040-4/10/$26.00 ©2010 IEEE

II. Related Work

Over the last decade, several mobile robots have been deployed successfully in crowded environments such as museums [3] railway stations [15], and exhibits [18], [19]. Typically, these systems solved the problem of navigating through crowded environments by computing the shortest path to the next goal and then following this path using a local collision avoidance system [3]. None of these systems attempt to make the robot move human-like or follow the natural flow of people through the environment. In contrast, our goal is to enable scenarios in which mobile robots are not the focus of attention but become every day devices that move through environments with only little special attention.

More recently, Kirby and colleagues [7] performed studies to evaluate different navigation strategies for a robot moving along with a single person. They found that people clearly preferred moving with a robot that shows more human-like navigation behavior. Mueller *et. al.* [13] developed a technique that aims at efficiently navigating through crowded spaces by following people. Their approach continuously tracks people in the robot's vicinity and chooses to follow people that move in the direction of the goal. While such a technique might result in more efficient navigation, the approach relies on manually tuned heuristics and has no explicit criterion for generating human-like behavior.

The graphics community has developed path planning techniques for efficiently generating naturally looking animations of densely crowded scenes [20], [21], [10], [11]. While these techniques result in very natural motion patterns, they require manual parameter tuning and are not readily applicable to robot path planning. In some cases [20], [21] this is due to the fact that the planning techniques assume *global* knowledge about properties such as the motion direction and density of all people. Other work [10], [11] is focused on mimicking group crowd behavior by matching into a database of observed examples to generate plausible overall animations. In contrast to these techniques, our approach takes limited sensing into account and learns the parameters underlying the planning technique. In this paper, we rely on a crowd simulator [20] to generate scenarios that allow us to evaluate our approach.

Over the last years, several researchers have developed techniques for learning Markov Decision Process (MDP) models by observing human experts performing a task. The key idea underlying this family of techniques, which includes inverse reinforcement learning [14], [22], maximum margin planning [17], and apprenticeship learning [2], [1], is to learn a reward (or cost) function that best explains the expert decisions. Reward functions are represented by log-linear functions of features describing a task environment. While these techniques have been shown to work extremely well in several applications, they assume that all feature values are known and static during each demonstrated planning cycle. In contrast, our scenario requires learning from example paths that are the result of a (simulated) person updating feature value estimates and re-planning on the fly.

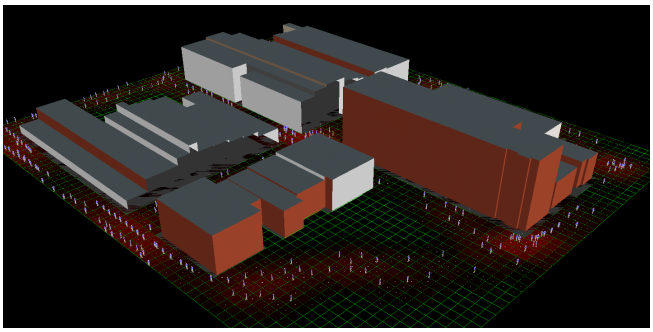

Fig. 1. An overview of the test environment and crowd simulator.

Gaussian processes (GPs) have been applied successfully by the spatial statistics community [4] and, more recently, by the robotics [5] community to model environmental properties. Here, we show how GPs can be used to estimate models of the density and flow of people in a crowded environment.

III. Inverse Reinforcement Learning with Partially Observable Features

In order for a robot to learn how to navigate a crowded space as humans do, we employ techniques based on maximum entropy inverse reinforcement learning (MaxEnt IRL) [22]. As in [23], we assume that a path, or trace, τ through states s_i and actions $a_{i,j}$ has a cost that is a linear combination of real-valued features $\mathbf{f}_\tau = \sum_{a_{i,j}\in\tau} \mathbf{f}_{a_{i,j}}$ observed along the path. In our context, states s_i correspond to discrete positions in the environment, actions $a_{i,j}$ are transitions between positions s_i and s_j, and vectors of real-valued features for each action model information such as the density of people in the state reached through that action. The cost of a path τ is parameterized by feature weights θ:

$$\text{cost}(\tau) = \theta \cdot \mathbf{f}_\tau = \sum_{a_{i,j}\in\tau} \theta \cdot \mathbf{f}_{a_{i,j}} \tag{1}$$

The original MaxEnt IRL model assumes that the agent has full knowledge of the features in the action space, and that these features remain constant for the duration of the path. For a fixed start and end state, this results in a maximum entropy distribution over paths parameterized by θ.

$$P(\tau|\theta) = \frac{1}{Z(\theta)} e^{-\theta\cdot\mathbf{f}_\tau} = \frac{1}{Z(\theta)} e^{\sum_{a_{i,j}\in\tau} -\theta\cdot\mathbf{f}_{a_{i,j}}} \tag{2}$$

Observe that increasing the linear combination of weights and features causes the probability to decrease exponentially.

To match observed behavior, IRL learns cost weights for the features such that the resulting planned paths are similar to the provided example paths. To achieve this, it is necessary and sufficient for the expected feature counts of the robot to match the observed feature counts of the humans when the cost is linear in those features [2].

We wish to find the parameters that maximize the likelihood of the observed traces $\mathcal{T}$:

$$\theta^* = \underset{\theta}{\text{argmax}} \sum_{\tau\in\mathcal{T}} \log P(\tau|\theta) \tag{3}$$

It can be shown that the gradient for a single path is the difference between observed and expected feature counts [2]. We denote by $\tilde{\mathbf{f}}$ the observed feature counts, which are simply the sum of features for all actions taken on an example trace τ. The expected feature counts can be obtained by multiplying the probability of each action by the features for that action, summed over all actions. Let $D_{a_{i,j}}$ be the expected frequency of action $a_{i,j}$, conditioned on θ, for all paths τ_m between the start and goal state. Then the gradient can be expressed as follows.

$$\nabla F = \tilde{\mathbf{f}} - \sum_{a_{i,j}} D_{a_{i,j}} \mathbf{f}_{a_{i,j}} \tag{4}$$

Using online exponentiated gradient descent, the weights θ are updated using the following formula.

$$\theta_{n+1} = \theta_n e^{-\gamma \nabla F} \tag{5}$$

Performing these updates over all paths while gradually reducing γ causes the weights to converge to θ^*. This is the weight update technique used in [22] and a comparison of exponentiated gradient descent with other techniques can be found in [8]. Unfortunately, the number of paths is exponential in the number of states. A tractable forward/backward algorithm is given in [23], similar to forward/backward inference algorithms for Hidden Markov Models and Conditional Random Fields [9]. This algorithm is given in Table I.

Forward/Backward Algorithm:

Input: start and goal states s_{start} and s_{goal}

$\forall s_i : Z'_{s_i} \leftarrow 0$

For N iterations (backward):

1: $Z'_{s_{goal}} \leftarrow Z'_{s_{goal}} + 1$

2: $\forall a_{i,j} : Z'_{a_{i,j}} \leftarrow e^{(-\theta \cdot \mathbf{f}_{a_{i,j}})} Z'_{s_j}$

3: $\forall s_i : Z'_{s_i} \leftarrow \sum_j Z'_{a_{i,j}}$

$\forall s_i : Z_{s_i} \leftarrow 0$

For N iterations (forward):

1: $Z_{s_{start}} \leftarrow Z_{s_{start}} + 1$

2: $\forall a_{i,j} : Z_{a_{i,j}} \leftarrow Z_{s_i} e^{(-\theta \cdot \mathbf{f}_{a_{i,j}})}$

3: $\forall s_j : Z_{s_j} \leftarrow \sum_i Z_{a_{i,j}}$

Result: probability mass Z'_s and Z_s for all states s.

TABLE I

After executing the forward/backward algorithm, we now possess values Z_{s_i} and Z'_{s_i} for every state, where Z_{s_i} is the accumulated probability mass of all paths from s_{start} to s_i, and Z'_{s_i} is the probability mass of all paths from s_i to s_{goal}. We can utilize these probabilities to compute $D_{a_{i,j}}$, which is the expected number of times action $a_{i,j}$ will be executed over all paths from s_{start} to s_{goal} subject to the current weights θ:

$$D_{a_{i,j}} = \frac{Z_{s_i} e^{(-\theta \cdot \mathbf{f}_{a_{i,j}})} Z'_{s_j}}{Z'_{s_{start}}} \tag{6}$$

Intuitively, this computes the expectation for action $a_{i,j}$ by multiplying the probability of all paths from s_{start} to s_i, then multiplying by the unnormalized probability $e^{(-\theta \cdot \mathbf{f}_{a_{i,j}})}$ of taking action $a_{i,j}$, and finally multiplying by the probability of all paths from s_j to s_{goal}. The result is normalized by dividing by the mass of *all* paths from s_{start} to s_{goal}, which is $Z'_{s_{start}}$ (or equivalently $Z_{s_{goal}}$). More details of the MaxEnt IRL formulation can be found in [22], [23].

A. Partially observable, dynamic features

Our scenario differs from the original MaxEnt IRL approach in two significant ways. First, the features that will allow a robot to move with people are as dynamic as the people themselves. The density and velocity of crowds change over time, both between traces *and* during the execution of a single trace. During training, however, we only have the final demonstrated traces, and not intermediate plans that were formulated but not executed. For instance, the event of changing a planned path based on an unforeseen blockage is not labeled as such in the training data. The second distinguishing aspect of our scenario is that the robot has no global knowledge of dynamic features.

In order to extend the MaxEnt IRL framework to this scenario, we assume that a person updates his estimate of the environment features at every time step and performs re-planning every $H \geq 1$ time steps. These assumptions allow us to compute the gradients of the example paths using a slightly modified training procedure. In a nutshell, we update for each time step the dynamic features within a small window around the person's current location, thereby simulating limited perceptual range (dynamic features outside this window are set to prior estimates). Each update gives us the information available to the person at that point in time. By assuming that the person plans a path based on this information and sticks to this path for the next H time steps, the gradient can be computed based on the difference between observed and expected feature over the next H steps only, instead of the complete path. However, since this update occurs at every time step, we need to compute this H-step difference for every time step in the path.

Specifically, we define the dynamic features for path τ and action $a_{i,j}$ at time t to be $\mathbf{f}^t_\tau$ and $\mathbf{f}^t_{a_{i,j}}$, respectively. The probability of a path depends on the cost of the path, which is obtained by adding the weighted features of the partial paths from each time horizon. Letting a^t represent the action of τ at timestep t, the probability of a path τ is now given as:

$$P(\tau|\theta) = \frac{1}{Z(\theta)} e^{\sum_t \sum_{0 \leq h < H} -\theta \cdot \mathbf{f}^t_{a^{t+h}}} \tag{7}$$

Because the features are dynamic, we compute a gradient at every timestep of an example path and only H timesteps into the future, as compared with (4) in the original IRL formulation, which computes a single gradient for the entire path. Our local gradient at timestep t is defined as:

$$\nabla F^t = \widetilde{\mathbf{f}^t} - \sum_{a_{i,j} \in H} D^t_{a_{i,j}} \mathbf{f}^t_{a_{i,j}} \tag{8}$$

where all terms are now indexed by the timestep t. In particular, $\widetilde{\mathbf{f}^t}$ is the observed features for the portion of the

example trace between timesteps t and $t+H$, and $D^t_{a_{i,j}}$ is the expectation of $a_{i,j}$ between t and $t+H$ conditioned on the current weights θ. We only compute expected features for actions reachable from the current location within H steps, hence the restricted summation in (8). However, note that we must perform the full forward/backward algorithm in Table I to compute even this subset of action expectation values. The gradient ∇F^t is used to update the weights as in equation (5), but now the weights θ are updated t times for each training path. A summary of our learning algorithm can be found in Table II.

Learning Algorithm:

For all training paths τ:

For all timesteps t in path τ:

1: Update estimates of locally observable features

2: Compute Z_s and Z'_s for all states using the algorithm in Table I

3: For actions $a_{i,j} \in H$ compute $D^t_{a_{i,j}}$ with equation (6)

4: Compute the gradient ∇F^t with equation (8)

5: Update the weights θ with equation (5)

TABLE II

B. Features and representation

In our current implementation, we use a grid representation of a two dimensional space. Additionally, our states include a discrete approximation of orientation at 45 degree intervals. Actions describe valid motion between these grid cell states. We found it most natural to express features (and hence costs) as corresponding to actions. Thus, features of the grid cells are represented as features on all actions leading into that grid cell's states. Our features were chosen such that they could be extracted from real robot sensors such as laser range finders and cameras.

We conjecture that humans optimize the cost of paths based on density and velocity of other nearby people (dynamic flow features), balanced against the distance traveled. For density features, we divide the real-valued density into four bins. Each action will have a value of 1 in exactly one of these bins, and zero in the others. This technique allows the learned weights to form a discrete approximation of a non-linear cost function over density. To represent velocity features, the average direction and magnitude of velocities for people in nearby cells are measured. As each state represents both a position and orientation, we use relative velocity features, again discretized into four bins evenly dividing the range of direction difference between 0 and 180 degrees. Each of these directional bins can have various magnitudes, which we represent with three bins per direction. In other words, relative direction and velocity magnitude form a "cross-product" feature for each action consisting of 12 bins total. We want to learn how humans balance these dynamic features against distance traveled, so each action, or transition between grid cells, has a static feature for the distance between the cells.

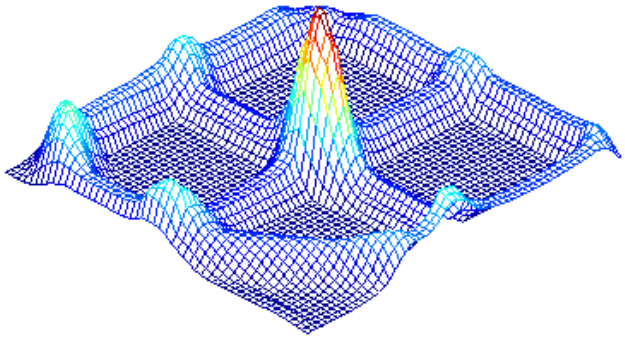

Fig. 2. Gaussian process mean model of pedestrian density over our simulated environment shown in Fig. 1.

For training data, we extract demonstration traces from the crowd simulator, where the traces consist of grid cells, directions, and local observations of dynamic flow features. These traces were collected from a simulation of normal pedestrian traffic. It is worth noting that similar example traces could be collected from real world pedestrians, using either hand labeling or automatic extraction. These simulations also provide our mean density and velocity models, which we can use as initial estimates of dynamic feature values in unobserved areas of the grid for both training and planning.

We found that our training procedure along with these features generated intuitively appealing results for weight values. For instance, low cost weights were learned for relative flow features moving in the same direction as the agent, and much higher costs were obtained for flow features moving in other directions.

IV. Gaussian Processes for Environmental Feature Estimation

Even under the best of circumstances, a robot will only be able to observe a small portion of the surrounding environment. We wish to integrate a robot's local observations of density and flow direction in a sound way with priors over the entire environment to produce a joint distribution of expected environment state. We perform this integration using Gaussian processes (GP) [16], a non-parametric model that estimates Gaussian distributions over functions based on training data.

Specifically, our environment is defined as a two-dimensional coordinate plane over which the robot travels. We wish to model features of density and direction of flow for traffic in this environment. To do so, we consider (α, β), where α is the (x, y) position in the coordinate plane and β is a vector of three values representing traffic density, traffic flow in the x component of the coordinate plane, and traffic flow in the y component of the coordinate plane, respectively.

Given a set of features $\{(\alpha_1, \beta_1) \cdots (\alpha_n, \beta_n)\}$, we learn GPs that map from input locations, α, to density and flow directions, β. A separate GP models each of the three components of β. For each Gaussian process, we use a

standard Gaussian kernel, whose parameters are trained by minimizing the log-likelihood of a GP over a training set. We can then evaluate the GP over the entire environment and the model will produce estimates of mean and variance, nicely integrating areas of dense and sparse data coverage. One such mean model of traffic density over our test environment can be seen in Fig. 2.

To update this model based on new feature values, we proceed as follows. Given a set of observations $\{(\alpha_1, \beta_1) \cdots (\alpha_n, \beta_n)\}$ at time t, we evaluate the mean model at locations $\{\alpha_1 \cdots \alpha_n\}$ and subtract the predicted mean from the actual observations to produce a set of deviation-from-the-prior observations $\{(\alpha_1, \overline{\beta}_1) \cdots (\alpha_n, \overline{\beta}_n)\}$. We construct a new GP with these updated observations, again using a basic Gaussian kernel, and evaluate the GP over the surrounding environment to get a smooth estimate of the observed deviation from the mean model prior. We add these estimated deviations to the mean model prior to get the updated feature model. The resulting technique correctly blends the new feature observations with the prior model. Note that a simple addition of the new observations to the prior GP would not result in a correct estimate, since such an approach would not consider that the new observations provide far more information about the current situation than the (outdated) prior model.

V. Experimental Results

Our primary goal is to generate robot paths similar to those of humans in crowded environments. We use the crowd motion simulator of [20] to obtain training data. So-called "swarms" of people were given starting locations and goal regions instigating natural crowd flows through a simulated 3D space. The density and velocity features are extracted from this simulator, then smoothed and discretized to fit our particular grid-based model. We argue that equivalent local features are available with real world robotic sensors, such as laser scanners and cameras. Additionally, as our framework is quite general, novel features could be incorporated as well.

We also use this same crowd simulator to provide a testbed for our planner. In order for our results to be realistic and transferable to real robots, the planner and path following systems are written as node processes in the Robot Operating System (ROS) by Willow Garage [6]. We wrote a wrapper for the crowd motion simulator in ROS to facilitate testing. The robot is represented in the crowd simulator as a single person, which allows the simulated crowds to react naturally to the presence and motion of the robot.

The path planner uses cost weights learned previously through IRL to define cost weights for all actions in the map. Given best estimates for dynamic features, the linear combination of weights and features gives a real valued cost to each edge. The planner then uses A* search to compute the best route to the goal for the current dynamic features. When new sensor information is available, the dynamic features are updated and a new planned path is generated from the current location.

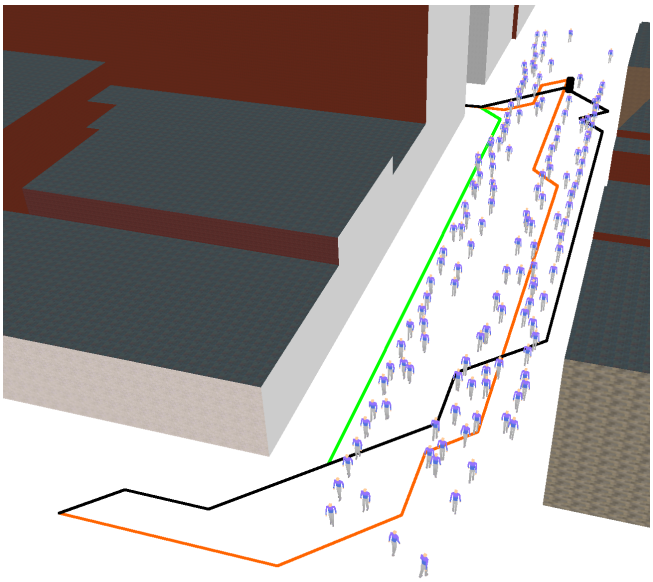

Fig. 3. A view of the robot's path (black), a human path (orange/dark gray), and the shortest path (green/light gray). There are two crowd flows, each moving along the right side of the walkway. Note that the robot moves with the flow on the right side and then crosses the oncoming crowd flow in a manner similar to the human path at the end of its journey.

	Shortest Path	Learned Path
Mean Distance	1.4	**0.9**
Maximum Distance	3.3	**2.3**

TABLE III

As a quantitative evaluation we selected various start and goal points in the lane formation scenario shown in Fig. 4. We consider the path taken by the crowd simulator from start to goal as the ground truth path. We then compare this path to the path generated by our approach (*Learned* path). As a baseline, we also compare the shortest Euclidean distance path (*Shortest* path) to the ground truth. To compare two paths, we compute for each position on the path the shortest distance to any position on the other path. From this we can compute the mean distance and maximum distance for a pair of paths. The results, averaged over six runs, are shown in Table III. From these results we conclude that our learned cost function provides a more natural path through crowded scenarios than a simple Euclidean distance based planner.

Fig. 3 shows an example of the path followed by our robot (black), the path of a human from the crowd simulator (orange/dark gray), and the simple shortest path (green/light gray). As can be seen, the path generated with our system matches nicely with the path of the simulated human. The reason this occurs is that our robot senses the crowd flow on the right side of the walkway and has learned low cost weights for walking with flows. Contrast this with the clearly disruptive and inefficient simple shortest path, which moves against the flow of the crowd.

One of the properties of the original global crowd simulator is lane formation. In the lane formation scenario shown in Fig. 4, there are three crowd flows, with the top and bottom flow moving right, and the middle flow moving left (against the robot's desired direction of travel). In the upper panel in

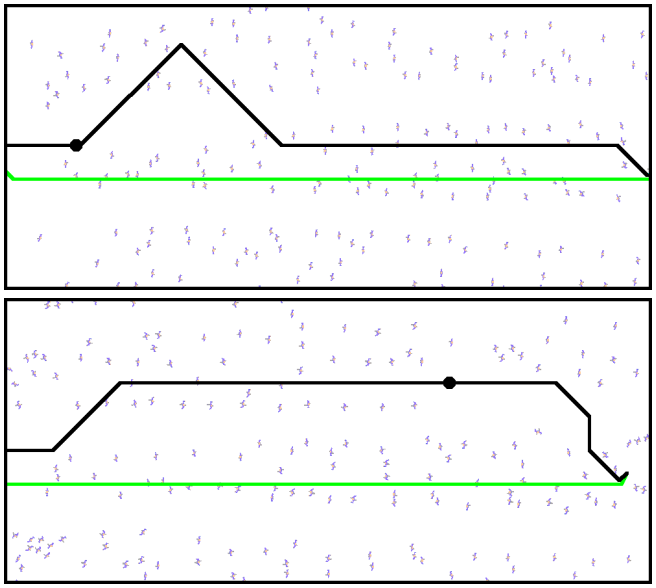

Fig. 4. The robot plans a path in the lane formation scenario. The robot is shown as a black circle, the path taken so far (and planned) by our robot is black, and the shortest path is green (light gray). The portion of the robot's path to its right is its planned path when it is at the position shown. There are three crowd flows: the top and bottom flows are moving right, and the middle flow moves left, against the robot. Upper panel: The robot can observe some of the oncoming flow, and plans a path to avoid the oncoming crowd. Note that beyond the locally observable features, the robot's planned future path takes it back to the middle. Lower panel: The conclusion of the lane formation scenario, showing that the robot has moved with the top crowd flow which was also moving right.

Fig. 4, the shortest path clearly moves against the flow. The planned path moves into a correct flow, but then reverts to the shortest path outside the sensor range. In the lower panel, we can see that the robot has continued to move with the correct, rightward moving flow, as it has obtained additional dynamic feature information as it moves forward. Our learned cost function planner smoothly reverts to following the shortest path in the absence of crowd flows. Note that the system was not trained on the lane formation scenario, yet exhibits natural, efficient behavior.

These properties of the algorithm can be seen in an accompanying video. Longer videos can be found at `http://www.cs.washington.edu/robotics/crowd`.

VI. Conclusion

We showed how to extend inverse reinforcement learning to deal with dynamic and partially observed features. The resulting system is able to learn to imitate human pedestrian behavior in crowded environments. With only a local sensor model, we use Gaussian processes to extend local information beyond the sensor range in a principled manner.

Our evaluation was performed within a realistic crowd simulator, and produced natural paths that integrated with existing crowd flows. We conjecture that our work produces more socially acceptable motion that will allow robots to perform tasks seamlessly in crowded environments.

In future work we hope to learn weights based on real world crowds, and implement the system on an actual robot. In a scenario with multiple robots distributed throughout the environment, the Gaussian process model will allow their local sensor readings to be combined across the map to allow more accurate density and flow information, enabling better path planning for each robot individually.

Acknowledgements

This research has been supported by ONR MURI grant N00014-09-1-1052 and by the NSF under grant numbers IIS-0705898 and IIS-0812671.

References

[1] P. Abbeel, D. Dolgov, A. Ng, and S. Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[2] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proc. of the International Conference on Machine Learning*, 2004.

[3] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2):3–55, 1999.

[4] N. Cressie. *Statistics for spatial data, revised edition*. Wiley, 1993.

[5] B. Ferris, D. Hähnel, and D. Fox. Gaussian processes for signal strength-based location estimation. In *Proc. of Robotics: Science and Systems (RSS)*, 2006.

[6] Willow Garage. Robot operating system (ros). http://www.willowgarage.com/pages/software/ros-platform.

[7] R. Kirby, J. Forlizzi, and R. Simmons. Natural person-following behavior for social robots. In *Proc. of Human-Robot Interaction*, 2007.

[8] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 1997.

[9] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of the International Conference on Machine Learning*, 2001.

[10] K.H. Lee, M. G. Choi, Q. Hong, and J. Lee. Group behavior from video: A data-driven approach to crowd simulation. In *Proceedings of the Eurographics Symposium on Computer Animation*, 2007.

[11] A. Lerner, Y. Chrysanthou, and D. Lischinski. Crowds by example. In *Eurographics*, 2007.

[12] M. Michalowski, S. Sabanovic, C. DiSalvo, D. Busquets, L. Hiatt, N. Melchior, and R. Simmons. Socially distributed perception: GRACE plays social tag at AAAI-05. *Autonomous Robots*, 22, 2007.

[13] J. Mueller, C. Stachniss, K. Arras, and W. Burgard. Socially inspired motion planning for mobile robots in populated environments. In *Proc. of International Conference on Cognitive Systems*, 2008.

[14] A. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proc. of the International Conference on Machine Learning*, 2000.

[15] E. Prassler, J. Scholz, and P. Fiorini. A robotic wheelchair for crowded public environments. *IEEE Robotics & Automation Mag.*, 7(1), 2001.

[16] C.E. Rasmussen and C.K.I. Williams. *Gaussian processes for machine learning*. The MIT Press, 2005.

[17] N. Ratliff, J. Bagnell, and M. Zinkevich. Maximum margin planning. In *Proc. of the International Conference on Machine Learning*, 2006.

[18] R. Siegwart, K. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, R. Philippsen, R. Piguet, G. Ramel, G. Terrien, and N. Tomatis. Robox at expo.02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42, 2003.

[19] R. Simmons et.al. GRACE: An autonomous robot for the AAAI robot challenge. *AAAI Magazine*, 24(2), 2003.

[20] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 25(3), 2006.

[21] J. van den Berg, S. Patil, J. Sewall, D. Manocha, and M. Lin. Interactive navigation of individual agents in crowded environments. In *Proc. of ymposium on Interactive 3D Graphics and Games*, 2008.

[22] B. Ziebart, A. Maas, J. Bagnell, and A. Dey. Maximum entropy inverse reinforcement learning. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2008.

[23] B. Ziebart, A. Maas, J. Bagnell, and A. Dey. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *International Conference on Ubiquitous Computing (UbiComp)*, 2008.