Backprop-MPDM: Faster risk-aware policy evaluation through efficient gradient optimization

Dhanvin Mehta[1] Gonzalo Ferrer[2] Edwin Olson[1]

***Abstract*— In Multi-Policy Decision-Making (MPDM), many computationally-expensive forward simulations are performed in order to predict the performance of a set of candidate policies. In risk-aware formulations of MPDM, only the worst outcomes affect the decision making process, and efficiently finding these influential outcomes becomes the core challenge. Recently, stochastic gradient optimization algorithms, using a heuristic function, were shown to be significantly superior to random sampling.**

In this paper, we show that accurate gradients can be computed – even through a complex forward simulation – using approaches similar to those in deep networks. We show that our proposed approach finds influential outcomes more reliably, and is faster than earlier methods, allowing us to evaluate more policies while simultaneously eliminating the need to design an easily-differentiable heuristic function. We demonstrate significant performance improvements in simulation as well as on a real robot platform navigating a highly dynamic environment.

I. Introduction

Multi-Policy Decision Making (MPDM) [1], [2] dynamically switches between a set of candidate closed-loop policies, allowing it to adapt to different situations encountered while navigating in an uncertain dynamic environment.

When first proposed, MPDM evaluated policies according to the expected cost of the outcomes which naturally weights outcomes according to their likelihood. But in risk-aware settings [3], policies are evaluated according to the *maximum* value of $P(x)C(x)$, where $P(x)$ is the probability density of the outcome and $C(x)$ is its cost. This risk-aware approach tends to skew decision-making to award policies that have fewer potentially dangerous outcomes. A key advantage is that the evaluation of a policy can be seen as an optimization problem, as opposed to the computation of a probabilistic expectation.

The major challenges arise from the large space of possible initial configurations as well as the sensitivity of the forward simulation to the initial configuration. As illustrated in Fig. 2, initial configurations that result in bad outcomes may be individually likely (high probability density), but scarce (low total probability mass) and are hence unlikely to be discovered through random sampling, resulting in poor performance. Recently, stochastic gradient optimization algorithms using heuristic functions were shown to be significantly superior to random sampling [3]. Although these approximate gradients are useful for guiding search, they are still sub-optimal and limit the number of policies that can be evaluated reliably in real-time. With limited options, the robot is often forced to stop as seen in Fig. 1.

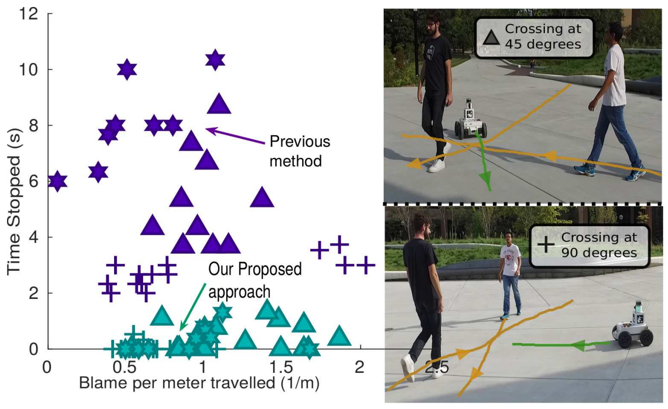


Fig. 1. Repeated real-world experiments. We collect real-world data from three repeatable experiments represented by different symbols 1) pedestrians crossing orthogonal to the robot's trajectory (+), 2) pedestrians crossing the robot's path obliquely at 45 degrees (△) and 3) pedestrians walking slowly in front of the robot (star). We measure the *Time Stopped* for every goal reached as well as the *Blame per meter traveled* by the robot accumulated by inconveniencing pedestrians. Lower the *Time Stopped* and *Blame*, the better. Our proposed approach (green) can evaluate more policies in real-time than earlier possible. With more candidate policies, the robot can find good policies and can navigate safely without stopping unnecessarily.

The key idea in this paper is representing a forward simulation as a differentiable deep network and enabling effective backpropagation. We show that despite our objective function being highly non-convex, we can efficiently calculate accurate gradients that can be used to update all agents simultaneously. The performance benefit of our proposed approach increases as the number of other agents in the environment increases, due to the increasing inadequacy of the previously used heuristic function.

The contributions of this paper include the following:

1) We show that our forward simulation can be encoded in a differentiable deep network and propose an efficient procedure using backpropagation to compute an accurate gradient of the objective function without the need for heuristics (Sec. IV). Unlike Stochastic Gradient Ascent (SGA) [3], our computed gradient accounts for interactions with other agents and static obstacles.

2) We demonstrate the limitations of the SGA algorithm in crowded configurations. We show that our proposed method not only outperforms SGA in simple configurations but also scales well with the number of agents.

3) Additionally, we show that our method allows us to reliably evaluate more policies in real-time than pre-

[1]The authors are at the University of Michigan, Ann Arbor. {dhanvinm,ebolson}@umich.edu.

[2]G. Ferrer is with the CDISE department, Skolkovo Institute of Science and Technology, Moscow, Russia. g.ferrer@skoltech.ru

This project was supported by the NSF CyberSees grant CCF 1442773.

viously possible, which results in significant performance improvements on a real robot platform navigating in a highly dynamic environment (Sec. V-B).

II. Related Work

In this section, we summarize prior work related to both the problem and the optimization technique: first, discussing previous approaches in the area of autonomous navigation in dynamic environments, then discussing techniques that use backpropagation for optimal control and parameter optimization.

Planning techniques and learning-based methods are commonly used for navigation in dynamic environments. Learning-based approaches [4], [5] use relevant features that might explain the interactions taking place between dynamic agents, but may be limited by the training datasets used. Chen et al. [6] developed a deep reinforcement learning approach to learn a time-efficient navigation policy that respected common social norms. The work of Fulgenzi et at. [7] predicted a set of trajectories for each agent over a time horizon. However, they did not address the interactions of those predicted trajectories with the actual robot trajectory. The work of Trautman et al. [8] is an example of the reciprocal evaluation of future trajectories where the robot plans using Gaussian Process. Ferrer [9] considers temporal constraints and optimizes over several objectives. The essence of the present paper is to efficiently capture those interactions, particularly looking for likely adversarial outcomes.

POMDPs provide a rigorous formalization for incorporating uncertainty into planning, but rapidly become intractable as the dimensionality of the state-space grows. Recently, approximate POMDP methods based on scenario sampling and forward simulation have been applied to navigation [10] and mapping [11]. Multi Policy Decision Making (MPDM) [1], [2] is a framework for navigation in dynamic environments under uncertainty, by dynamically switching between a set of policies evaluated using forward simulations.

Classic techniques for trajectory optimization and optimal control such as LQR for linear systems [12], or nonlinear approaches such as iLQG [13] and LQR-Trees [14], use a backwards process to efficiently update solutions. While our proposed method uses a similar process, our application is quite different. In our target domain, the complexity arising from interactions between multiple agents in the environment makes finding an optimal solution impractical. Hence, MPDM chooses from a fixed set of closed-loop policies and in this paper, the proposed optimization *evaluates* these candidate policies.

Backpropagation has been the de-facto method for parameter optimization in neural networks since the 80's. Seminal works [15], [16] showed encouraging results using neural networks to learn control actions from perception. With increased computational power, Levine et al. [17] have applied deep learning to much harder problems of end-to-end robotic manipulation. Our representation of the forward simulation as a differentiable deep network makes risk-aware MPDM more amenable to learning methods. However, end-to-end learning is non-trivial in our domain. Complex interactions between pedestrians and the robot's closed-loop policies make data collection and generalization challenging.

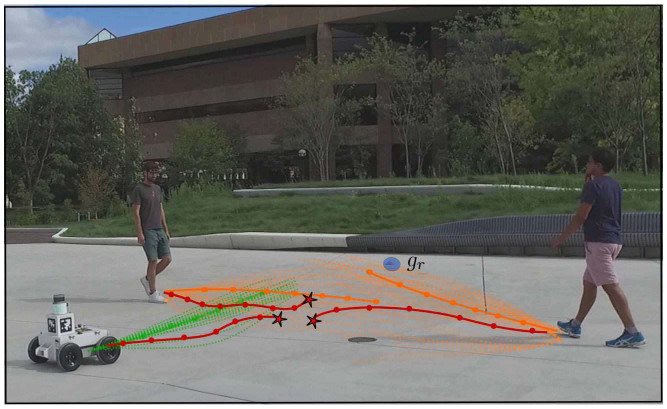


Fig. 2. An illustration motivating risk-aware planning. The trajectories arising from the most likely initial configuration for the agents (orange) and the robot (green), like most outcomes of possible initial configurations (dashed lines) are benign. Near misses are mundane as the agents in the forward simulation tend to avoid collision. As a result, there may be a few dangerous initial configurations that may be individually likely and yield high-cost outcomes (red). While evaluating policies, likely collisions (red stars) should be discovered as quickly as possible to allow larger candidate policy sets for MPDM.

III. Risk-Aware Multi-Policy Decision Making

In this paper, we use non-holonomic motion models for each observed agent i as well as for the robot. The robot maintains a probabilistic estimate of each observed agents' state - i.e. its position, velocity, angular velocity and inferred policy. An agent's policy $\pi_i = (v_{des}, \boldsymbol{g}_{sub})$, expresses an intent to move towards sub-goal $\boldsymbol{g}_{sub}$ at a desired speed v_{des}. The collective state $\boldsymbol{x}_t \in \mathcal{X}$ consists of the states of the robot and all observed agents at time t. Throughout the paper, we will refer to $\boldsymbol{x}_0$ as the collective state of all agents and the robot's state at the current time. The probabilistic estimate $P(\boldsymbol{x}_0)$ is based on past observations of the pedestrians' positions[1]. The robot's policy π is elected from amongst a set of closed-loop policies Π.

An initial sampled configuration $\boldsymbol{x}_0$ is forward simulated H time-steps (through $t = 1, \ldots, H$), by recursively applying the transition function $T : \mathcal{X} \to \mathcal{X}$ to yield a trajectory

$$\begin{aligned}\boldsymbol{X}(\boldsymbol{x}_0) &= \{\boldsymbol{x}_0, T(\boldsymbol{x}_0), T^2(\boldsymbol{x}_0), \ldots, T^H(\boldsymbol{x}_0)\} \\ &= \{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_H\},\end{aligned}$$

where $\boldsymbol{x}_t \in \mathcal{X}$ is the collective state consisting of the robot state plus all the agents at time t of the forward simulation. The transition function $T()$ captures the trajectory that each agent is executing while at the same time considering the interactions with all other agents.

The cost function $C\big(\boldsymbol{X}(\boldsymbol{x}_0)\big)$ assigns a scalar value to the outcome of a simulation. Like earlier work [3], we use a cost

[1]Several methods can be used for estimating $P(\boldsymbol{x}_0)$ based on past trajectories of agents. We use a Kalman Filter to infer position and velocity and a Naive Bayes Classifier to infer an agent's policy parameters.

function that penalizes the inconvenience the robot causes to other agents in the environment (*Blame*) along the predicted trajectory and rewards the robot's progress towards its goal (*Progress*).

In Risk-aware MPDM, the robot's policies are evaluated based on the most influential (likely and high-cost) outcome that may occur. Such outcomes are discovered by optimizing a probabilistic cost surface $\max_{\boldsymbol{x}_0}\{P(\boldsymbol{x}_0)C(\boldsymbol{X}(\pi,\boldsymbol{x}_0))\}$, instead of the expected value of the cost function approximated by sampling.

Algorithm 1 describes the policy election for risk-aware MPDM. Provided with a probability distribution over initial configurations, $P(\boldsymbol{x}_0)$, a set of candidate policies, Π, and a forward simulation budget, N_π, each candidate policy is evaluated (scored) according to the most influential (worst-case) outcome discovered within the computational budget.

The objective function $P(\boldsymbol{x}_0)C(\boldsymbol{X})$ can have multiple local-minima depending on the number of agents and the complexity of the initial configuration. Finding the global maximum through exhaustive search is computationally infeasible due to the large state-space. Our goal is to quickly find an influential configuration whose value is comparable to the global optimum even if it may not be the highest-valued configuration.

Algorithm 1 Policy Election for Risk-aware MPDM
```
 1: function POLICY-ELECTION LOOP($P(\boldsymbol{x}), \Pi, N_\pi$)
 2:     for $\pi \in \Pi$ do
 3:         Initialize $U_\pi, n \leftarrow 0$
 4:         while $n < N_\pi$ do
 5:             Sample $\boldsymbol{x}_0 \sim P(\boldsymbol{x})$
 6:             $\boxed{U^*, n_{opt} \leftarrow \text{Optimize}(\boldsymbol{x}_0, \pi)}$
 7:             $n \leftarrow n + n_{opt}$
 8:             $U_\pi \leftarrow \max\{U^*, U_\pi\}$
 9:         end while
10:     end for
11:     $\pi^* \leftarrow \arg\min_\pi U_\pi$
12: end function
```

Our algorithm samples an initial configuration from $P(\boldsymbol{x}_0)$ (Line 5) and optimizes it, perturbing the sampled configuration iteratively towards increasingly influential outcomes until convergence to a local optima whose objective function value is U^* (Line 6). The number of forward simulations n_{opt} used by an optimization procedure corresponds to its rate of convergence. Upon convergence, a new initial configuration is sampled and this process is repeated until the forward simulation budget N_π is consumed. The utility of a policy U_π is the most influential (highest-valued) configuration encountered. The policy with the least risk is elected.

In the next section, we address the problem of computing the function Optimize$(\boldsymbol{x}_0, \pi)$ efficiently using backpropagation (BP), overcoming the limitations of our previous optimization technique, Stochastic Gradient Ascent (SGA).

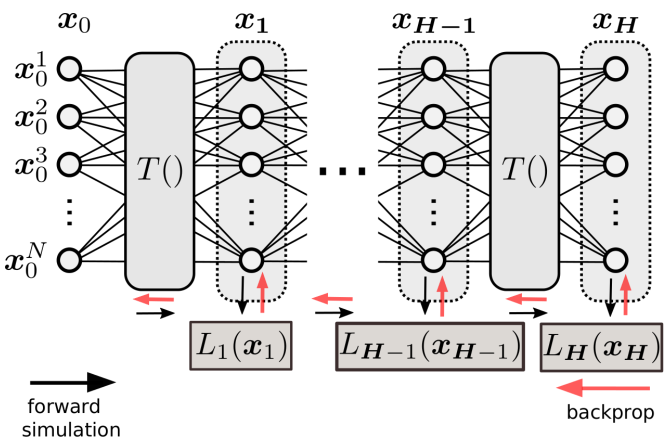


Fig. 3. A deep network representation for our cost function. The initial configuration $\boldsymbol{x}_0$ propagates through several layers, each representing the transition function T. The output of layer t determines a cost $L_t(\boldsymbol{x}_t)$. Our cost function $C(\boldsymbol{X}(\boldsymbol{x}_0))$ accumulates costs calculated at each time-step along the forward simulated trajectory.

IV. Computing Accurate Gradients

A. Network Architecture

Deep neural networks model complex functions by composing (chaining) relatively simple functions (convolutions or ReLU modules). Similarly, a forward simulation captures the complex dynamics of the system using simple one-step transition functions T.

Since our cost function is a linear combination of costs computed along the trajectory, we can conceptualize the forward simulation as a deep network (Fig. 3) that outputs a trajectory cost $C(\boldsymbol{X}(\boldsymbol{x}_0))$ based on the input initial configuration $\boldsymbol{x}_0$.

Let $L_t(\boldsymbol{x}_t)$ be the cost accrued at time-step t for the state $\boldsymbol{x}_t$. We define a function $\Phi(t,\boldsymbol{X})$ that accumulates the cost of a trajectory, from the final time H backwards to the initial time $t = 0$

$$\Phi(t,\boldsymbol{X}) = \sum_{\tau=t}^{H} L_\tau(\boldsymbol{x}_\tau). \tag{1}$$

Our objective cost can be expressed as $C(\boldsymbol{X}) = \Phi(0,\boldsymbol{X})$. We can formulate Φ recursively as:

$$\Phi(t,\boldsymbol{X}) = \Phi(t+1,\boldsymbol{X}) + L_t(\boldsymbol{x}_t). \tag{2}$$

We want to compute $\nabla_{\boldsymbol{x}_0} C(\boldsymbol{X}) = \nabla_{\boldsymbol{x}_0}\Phi(0,\boldsymbol{X})$. The gradient of the cost at time-step H is

$$\nabla_{\boldsymbol{x}_H}\Phi(H,\boldsymbol{X}) = \frac{\partial\Phi(H,\boldsymbol{X})}{\partial\boldsymbol{x}_H} = \frac{\partial L_H(\boldsymbol{x}_H)}{\partial\boldsymbol{x}_H}. \tag{3}$$

We can compute the gradient iteratively from time-step H

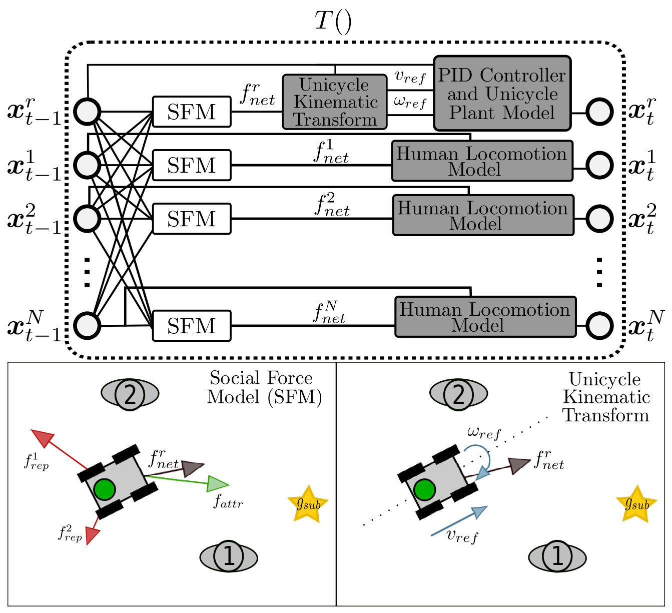


Fig. 4. Block diagram of the transition function. At each time-step, an agent i (in this case, the robot) is repelled by other agents (f_{rep}^j) and attracted towards its sub-goal g_{sub} in accordance to the Social Force Model (SFM). Pedestrians are modeled using the HSFM model where the social force [18] acts as a control input for the Human Locomotion Model. The robot is modeled like a unicycle and the social force f_{net}^r is transformed into a compliant reference signal (v_{ref}, ω_{ref}) for a lower-level velocity controller.

backwards to $t = 0$ by applying (2) and expanding terms:

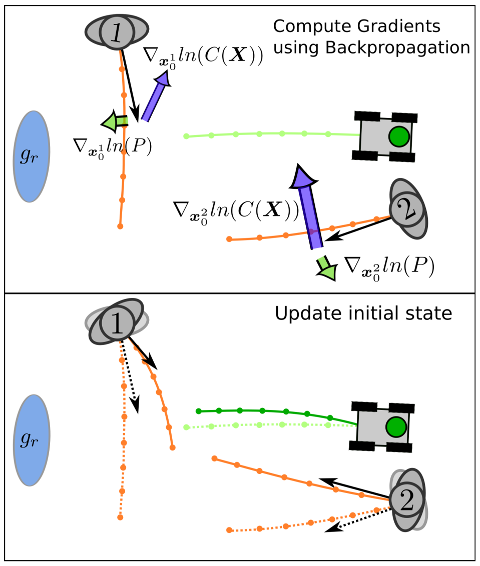
$$\begin{aligned}\nabla_{\boldsymbol{x}_t}\Phi(t,\boldsymbol{X}) =& \frac{\partial\Phi(t,\boldsymbol{X})}{\partial\boldsymbol{x}_t} = \frac{\partial\{\Phi(t+1,\boldsymbol{X})+L_t(\boldsymbol{x}_t)\}}{\partial\boldsymbol{x}_t}\\ =& \frac{\partial\Phi(t+1,\boldsymbol{X})}{\partial\boldsymbol{x}_t} + \frac{\partial L_t(\boldsymbol{x}_t)}{\partial\boldsymbol{x}_t}\\ =& \frac{\partial\Phi(t+1,\boldsymbol{X})}{\partial\boldsymbol{x}_{t+1}}\frac{\partial\boldsymbol{x}_{t+1}}{\partial\boldsymbol{x}_t} + \frac{\partial L_t(\boldsymbol{x}_t)}{\partial\boldsymbol{x}_t}\\ =& \frac{\partial\Phi(t+1,\boldsymbol{X})}{\partial\boldsymbol{x}_{t+1}}\boxed{\frac{\partial T(\boldsymbol{x}_t)}{\partial\boldsymbol{x}_t}} + \frac{\partial L_t(\boldsymbol{x}_t)}{\partial\boldsymbol{x}_t}.\end{aligned} \tag{4}$$

Eqn. 4 can be used to efficiently compute $\nabla_{\boldsymbol{x}_0}C(\boldsymbol{X})$ as long as the gradient of transition function can be computed effectively.

B. Limitations of Earlier Approaches

We have recognized that the kinematic models used for the agents have an impact on the quality of the gradients. Previous implementations of MPDM used a simple double integrator model for all agents with heuristics to restrict lateral motion for more realistic simulation [2]. While the simple model was useful for fast forward simulation, the heuristics contain hard thresholds that manifest as zeros in the matrix $\frac{\partial T(\boldsymbol{x}_t)}{\partial\boldsymbol{x}_t}$. As a result, useful gradients are truncated (as highlighted by the box in Eqn. 4 hampering effective backpropagation.

Previous methods [3] used a surrogate cost function designed to avoid computing the gradient of the transition function. Although these approximate gradients are useful



Compute Gradients using Backpropagation
$\nabla_{\boldsymbol{x}_0^1} ln(C(\boldsymbol{X}))$
$\nabla_{\boldsymbol{x}_0^1} ln(P)$
g_r
$\nabla_{\boldsymbol{x}_0^2} ln(C(\boldsymbol{X}))$
$\nabla_{\boldsymbol{x}_0^2} ln(P)$
Update initial state

Fig. 5. Backpropagation finds increasingly influential outcomes. The forward propagated outcome of the sampled initial configuration (*Top*) is not discouraging for the robot as it does not inconvenience either agent. For agents $i = \{1, 2\}$, the computed gradients $\nabla_{\boldsymbol{x}_0^i} ln(C(\boldsymbol{X}))$ (Blue) drive the agents towards configurations where the robot would inconvenience them under its current policy while $\nabla_{\boldsymbol{x}_0^i} ln(P(\boldsymbol{x}_0))$ (Green) drive them to more likely configurations. The agents can be simultaneously updated resulting in a more influential configuration (*Bottom*).

for guiding search, they are still sub-optimal. Moreover, designing such a function is difficult and ultimately ad-hoc.

C. Our proposed transition function

In this paper, we use non-holonomic kinematic models that augment the agent's state with angular velocity to capture the effect of lateral forces. This model ensures the differentiability of T while maintaining realistic human motion in the forward simulation.

Specifically, we use the headed social force model (HSFM) [18] for all the pedestrians and a unicycle-like model for the robot as described below. For the robot, the net force is computed using the SFM f_{net}^r, but due to the inherent constraints on a wheeled platform, we transform f_{net}^r into a compliant reference signal (v_{ref}, ω_{ref}) for a lower-level velocity controller

$$\begin{bmatrix} v_{ref} \\ \omega_{ref}\end{bmatrix}_{t+1} = \begin{bmatrix}\cos(\theta) & \sin(\theta)\\ -\sin(\theta) & \cos(\theta)\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & \frac{1}{l}\end{bmatrix} f_{net}^r. \tag{5}$$

The lookahead distance l determines the tendency of the robot to turn to compensate the lateral force. The robot's state is then propagated towards the reference signal using a first-order model for each of the independent wheel velocity controllers and a unicycle plant model.

D. Backpropagation

Our proposed transition function layer $T(\boldsymbol{x}_t)$ (Fig. 4) allows us to compute accurate gradients of the transition function. Eqn. 4 can now be implemented efficiently via

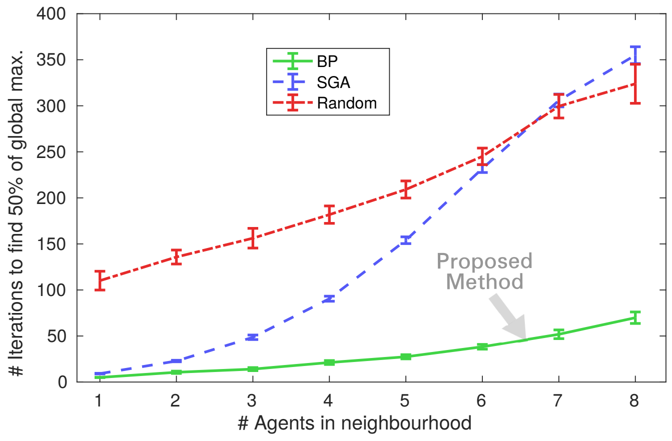


Fig. 6. Degradation of SGA in crowded scenarios. For each algorithm, we estimate the mean and standard error of the number of iterations (forward simulations) taken to discover an influential outcome varying the number of agents in the robot's vicinity, and thereby the dimensionality of the search space. The lower the slope, the better, more robust the algorithm to complex scenarios with high-dimensional search spaces. Random sampling, as expected, requires many samples even in simpler configurations (1 agent). SGA cannot find influential outcomes efficiently in complex scenarios with multiple agents, scaling so poorly that for more than 6 agents it performs worse than random sampling. BP is able to find those adverse outcomes even for crowded scenarios with 8 people.

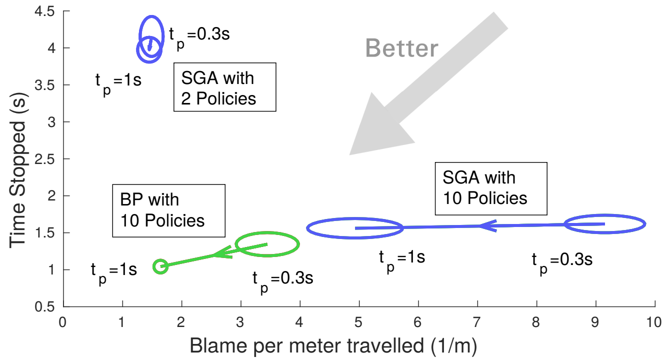


Fig. 7. Our proposed method, BP can evaluate 10 policies reliably in real-time, while SGA cannot. We compare the performance of various algorithms on 6 hours of navigation in our simulated environment. We measure the *Time Stopped* for every goal reached as well as the *Blame per meter traveled* by the robot. For each algorithm, we use bootstrap sampling to estimate the mean and standard error for these metrics, represented by the axes of an ellipse. Lower the *Blame* or *Time Stopped*, the better. We run the simulator in real-time allowing a planning time $t_p = 0.3s$. Although SGA can evaluate the smaller policy set reliably in real-time, the lack of options results in frequent *Stopping*. Unfortunately, SGA cannot evaluate a larger policy set of 10 policies reliably and accumulates large *Blame*. Since BP can evaluate the larger policy set more quickly and reliably than SGA , the robot navigates safely (low *Blame*) in real-time without *Stopping* unnecessarily. The benefits of risk-aware MPDM with the larger policy set can also be observed in the attached video. Upon slowing down the simulator (three times slower than real-time) to allow an unrealistic planning time of $t_p = 1s$, we observe that SGA with 10 policies is able to drastically reduce *Blame*. However, even then BP outperforms SGA.

backpropagation, where $\frac{\partial T(\boldsymbol{x}_t)}{\partial \boldsymbol{x}_t}$ and $\frac{\partial L_t(\boldsymbol{x}_t)}{\partial \boldsymbol{x}_t}$ are computed during the forward propagation and cached.

Fig. 5 illustrates one iteration of gradient ascent using backpropagation through a simple initial configuration $\boldsymbol{x}_0$ consisting of two agents and the robot executing the *Go-Solo* policy, where the robot moves straight towards its goal g_r, while trying to avoid pedestrians. The heuristic-based stochastic gradient method (SGA) computed approximate gradients for each agent and perturbed one agent at a time to avoid divergence. In contrast, by computing accurate gradients, we can perturb all the agents simultaneously without divergence. The gradient also accounts for agent-agent interactions as well as static obstacles. Each agent's update rate is determined using line-search along the gradient direction

V. EXPERIMENTS

The simulated environment consists of an open space, freely traversed by 15 agents that can randomly change speed or direction while the robot tries to reach its goal. The unconstrained domain allows for a large number of possible outcomes and makes the trajectories more dependent on initial configurations, which makes policy evaluation challenging. MPDM relies on quick decision making and re-planning (every 300ms) to react to sudden and unexpected changes in the environment.

Probabilistic Estimates: A pedestrian can suddenly come to a stop, slow down or speed up. We model this as a distribution over the preferred speed of each agent that is a mixture of two truncated Gaussians - one centered around the estimated most-likely current speed with a $\sigma = 0.4m/s$ to account for speeding up or slowing down and a truncated half Gaussian with a peak at 0 and $\sigma = 0.2m/s$ to account for coming to a sudden stop.

A pedestrian can also suddenly change direction without signaling. In order to account for uncertain direction for each agent, the robot assumes a Gaussian centered around the agent's estimated most-likely orientation and $\sigma = 30°$ that determines the agent's way-point. All truncated Gaussians are restricted to $\mu \pm 1.5\sigma$.

A pedestrian's sub-goal is inferred from a set of salient points using a Naive Bayes classifier.

Cost Function: For a sampled initial configuration, the predicted trajectory is evaluated using a cost function. High-cost outcomes correspond to those where the robot inconveniences other agents by driving too close to them, thus accumulating high *Blame*. The robot is also rewarded according to the *Progress* it makes towards the goal. These metrics are defined in greater detail in [3].

A. *Simulation Experiments*

We have used the same simulation environment that was used in [3] to demonstrate the limitations of Stochastic Gradient Ascent (SGA) in crowded scenarios, and validate the scalability of our proposed approach using backpropagation (BP).

1) Varying the number of agents: We generated a dataset consisting of 16k randomly chosen simulated scenarios where at least one agent was present within 5m of the robot. We then sort them based on the number of agents in the robot's neighborhood.

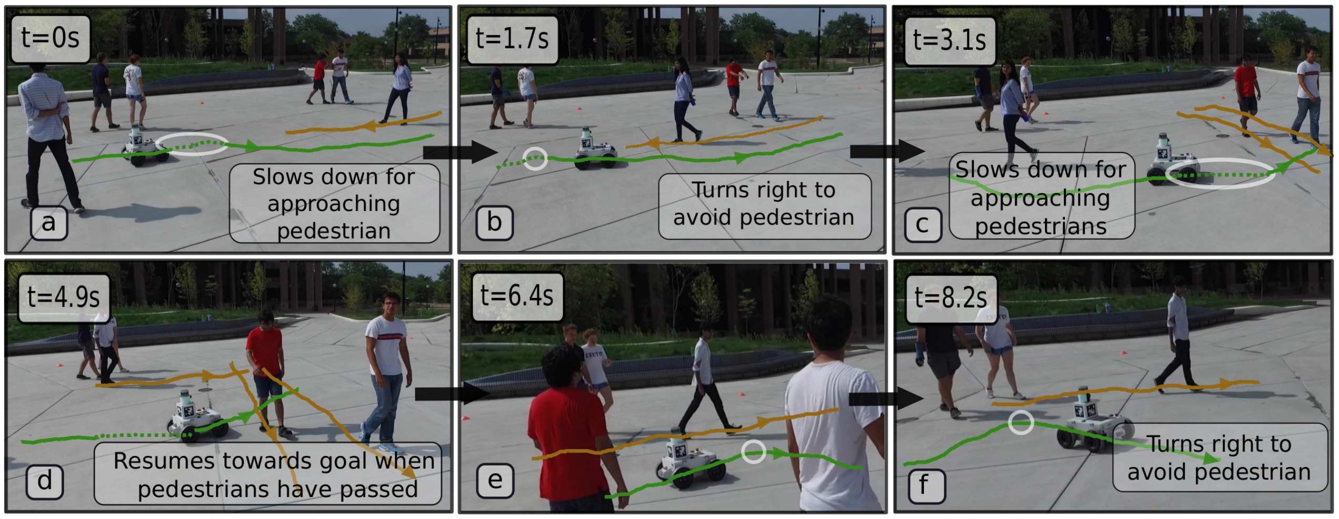

t=0s
Slows down for approaching pedestrian
a
t=1.7s
Turns right to avoid pedestrian
b
t=3.1s
Slows down for approaching pedestrians
c
t=4.9s
Resumes towards goal when pedestrians have passed
d
t=6.4s
e
t=8.2s
Turns right to avoid pedestrian
f

Fig. 8. Real situations illustrating the benefits of risk-aware MPDM with many policies. The sequence of images depict different situations that the robot encounters as it makes its way towards its goal. Dashed segments denote portions of the robot's trajectory (green lines) where it slowed down. The orange tracks represent the trajectories of relevant pedestrians. The lines were manually superimposed on the video after careful examination of the corresponding logs. The robot slows down upon discovering possible imminent collision (a and c) and turns appropriately to avoid them when possible (a and e) as denoted by the white ellipses. By dynamically switching between multiple policies, the robot is able to navigate safely, without stopping unnecessarily.

Our objective function $P(\boldsymbol{x}_0)C(\boldsymbol{X})$ is defined over innumerable possible initial configurations belonging to a high-dimensional continuous space that scales linearly with the number of agents considered. For each scenario, 2k random samples were optimized and the worst-case outcome was used to approximate the global optimum.

We now vary the number of agents in the robot's vicinity, thus increasing the complexity of the scenario and the dimensionality of the state space. For reliable real-time policy evaluation, influential outcomes must be detected quickly. We estimate the number of iterations needed by each algorithm to achieve a certain fraction (50%) of the worst outcome in the dataset (find an influential outcome).

For each algorithm, the experiment is run 1k times on each scenario. We use bootstrap sampling (with replacement) on our data-set as in [3] to estimate the mean and standard error of their performance.

Stochastic Gradient Ascent computes approximate agent-specific gradients of a simplified cost function. In order to limit the divergence arising due to these approximations, the stochastic gradients are ranked using a heuristic function and only the most promising agent is perturbed at a time. Despite performing well in scenarios involving few agents, this method does not scale well to more challenging crowded settings. Fig. 6 shows that although all the algorithms take longer to find influential outcomes as the complexity of the environment grows, the performance of SGA deteriorates sharply for more than 3 agents. Beyond 6 agents, it performs as poorly as random sampling since it takes a long time to converge from a sampled initial configuration to a local optimum. Backpropagation, on the other hand, overcomes these limitations as it computes accurate gradients, and all agents can simultaneously be updated without divergence.

2) Increasing the number of candidate policies for Risk-aware MPDM: Through 6 hours of navigation in our simulated environment, we demonstrate that our proposed approach, unlike SGA, can reliably evaluate a large policy set. Each simulation 'epoch' consists of a random initialization of agent states followed by a 5 minute simulated run at a granularity $\Delta t = 0.15$s. In our simulator, the observations $\boldsymbol{z}$ are modeled using a stationary Gaussian distribution with uncorrelated variables for position, speed and orientation for the agent. We parameterize this uncertainty by a scale factor $\{\sigma_{p_x}, \sigma_{p_y}, \sigma_{|v|}, \sigma_\theta\} = \{10cm, 10cm, 10cm/s, 15°\}$. The corresponding diagonal covariance matrix is denoted by $\mathrm{diag}(\sigma_{p_x}, \sigma_{p_y}, \sigma_{|v|}, \sigma_\theta)$. We do not perturb the goal and assume no angular velocity (ignoring any uncertainty). These uncertainties are propagated in the posterior state estimation $P(\boldsymbol{x}|\boldsymbol{z})$.

Our simulation experiments are run on an Intel i7 processor and 8GB RAM to mimic the computational capabilities of our robot. In order to react to sudden changes, MPDM relies on quick re-planning. The robot must replan every 300ms for effective real-time navigation. We evaluate the performance of risk-aware MPDM using 2 candidate sets of policies - a large candidate set with 10 policies, and a small set with 2 policies:

1) 2 Policies - $\{$*Go-Solo, Stop*$\}$ - The robot evaluates going straight towards the goal at maximum speed ($1.5m/s$) and stops if it senses danger.
2) 10 Policies - $\{$ *(Fast, Medium, Slow)*$\times$*(Straight, Left, Right), Stop*$\}$ - Rather than going straight towards the goal at maximum speed, the robot may also choose to go at *Medium* speed ($0.9m/s$) or *Slowly* ($0.2m/s$). Simultaneously, the robot can also choose to create a sub-goal to the *Left* or *Right* of the goal instead of going *Straight* to the goal as in *Go-Solo*.

We record the *Time Stopped* per goal reached, as well as the *Blame* normalized by the distance to goal (*Blame per meter traveled*). *Time Stopped* indicates the failure of the planner to find a safe policy. With a larger policy set, the robot is more likely to find a safe policy, and *Stops* less often. However, if the robot cannot evaluate its policy set quickly enough, it is unable to react to sudden changes in the environment and accumulates *Blame*. Ideally we would like a robot navigate safely (low *Blame*), with minimal Stop-and-Go motion.

Fig. 7 shows how the inefficiencies in SGA become a performance bottleneck. While SGA can navigate safely (low *Blame*) with the small policy set, it often fails to find safe policies and stops. With 10 policies, SGA fails to find influential outcomes fast enough resulting in high *Blame*. Our proposed method, BP can reliably evaluate the large policy set in real-time, which significantly improves navigation performance.

B. Real-World Experiments

We implemented our system on the MAGIC robot [19], a differential drive platform equipped with a Velodyne VLP-16 laser scanner used for tracking and localization. We use LCM [20] for inter-process communication. Every 300ms, MPDM evaluates a set of policies and chooses the least risky one. Although the policy election is slow, the robot is responsive as the policies themselves run at 50Hz.

Seven volunteers were asked to move towards marked points around an open space for 45 minutes. Fig. 8 demonstrates the emergent behavior through an 8 second timeline. We encourage the reader to see our attached video (`https://goo.gl/WgXW55`) demonstrating the advantages of large policy sets for risk-aware MPDM.

Fig. 1 shows data from 90 minutes of real-world experiments in which volunteers were asked to repeat three fixed scenarios while the robot made its way towards its goal. For both, our proposed approach as well as SGA, each scenario was repeated for 15 minutes. As observed in simulation, SGA was too slow to evaluate the larger policy set reliably and was unsafe to deploy on our robot. Using SGA with two policies (purple), the robot fails to find safe policies and stops often. Our proposed method (green) can reliably evaluate 10 policies in real-time (similar Blame as compared to SGA with just two policies) and as a result, it is more likely to find safe policies (low *Time Stopped*).

VI. Conclusions

We have presented a differentiable deep network that can encode a forward simulation, allowing effective backpropagation to compute the gradient of a complex cost function efficiently. Our approach quickly finds influential outcomes even in challenging scenarios with multiple agents. We overcome the limitations of our previous approach (SGA) [3] eliminating the need for heuristics, and its limitations. Unlike SGA, the gradient computed using Backprop-MPDM accounts for interactions with other agents and static obstacles.

We have shown that Backprop-MPDM can reliably evaluate more policies in real-time than previously possible, which results in significant performance improvements on a real robot platform navigating in a highly dynamic environment.

References

[1] A. G. Cunningham, E. Galceran, R. M. Eustice, and E. Olson, "MPDM: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving," in *Proc. IEEE Int. Conf. Robot. and Automation, Seattle, WA, USA*, 2015.

[2] D. Mehta, G. Ferrer, and E. Olson, "Autonomous navigation in dynamic social environments using multi-policy decision making," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016, pp. 1190–1197.

[3] ——, "Fast discovery of influential outcomes for risk-aware MPDM," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2017, https://april.eecs.umich.edu/papers/details.php?name=mehta2017icra.

[4] 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 *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 3931–3936.

[5] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *The International Journal of Robotics Research*, 2016.

[6] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," *arXiv preprint arXiv:1703.08862*, 2017.

[7] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Probabilistic motion planning among moving obstacles following typical motion patterns," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 4027–4033.

[8] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 335–356, 2015.

[9] G. Ferrer, "Social robot navigation in urban dynamic environments," Ph.D. dissertation, Universitat Politècnica de Catalunya, Spain, October, 2015.

[10] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online POMDP planning for autonomous driving in a crowd," in *Proc. IEEE Int. Conf. Robot. and Automation*, 2015, pp. 454–460.

[11] M. Lauri and R. Ritala, "Planning for robotic exploration based on forward simulation," *Robotics and Autonomous Systems*, vol. 83, pp. 15–31, 2016.

[12] R. F. Stengel, *Optimal control and estimation*. Courier Corporation, 2012.

[13] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the American Control Conference*. IEEE, 2005, pp. 300–306.

[14] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-Trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[15] D. A. Pomerleau, "ALVINN: an autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, 1989.

[16] K. J. Hunt, D. Sbarbaro, R. Żbikowski, and P. J. Gawthrop, "Neural networks for control systems: a survey," *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.

[17] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[18] F. Farina, D. Fontanelli, A. Garulli, A. Giannitrapani, and D. Prattichizzo, "When Helbing meets Laumond: the headed social force model," in *IEEE Conference on Decision and Control (CDC)*, 2016, pp. 3548–3553.

[19] E. Olson, J. Strom, R. Morton, A. Richardson, P. Ranganathan, R. Goeddel, M. Bulic, J. Crossman, and B. Marinier, "Progress toward multi-robot reconnaissance and the magic 2010 competition," *Journal of Field Robotics*, vol. 29, no. 5, pp. 762–792, 2012.

[20] A. S. Huang, E. Olson, and D. C. Moore, "LCM: Lightweight communications and marshalling," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 4057–4062.