APPEARED IN
Proc. Robotics: Science & Systems, 2019

LeTS-Drive: Driving in a Crowd by Learning from Tree Search

Panpan Cai, Yuanfu Luo, Aseem Saxena, David Hsu, Wee Sun Lee
School of Computing, National University of Singapore, 117417 Singapore

***Abstract*—Autonomous driving in a crowded environment, e.g., a busy traffic intersection, is an unsolved challenge for robotics. The robot vehicle must contend with a dynamic and partially observable environment, noisy sensors, and many agents. A principled approach is to formalize it as a Partially Observable Markov Decision Process (POMDP) and solve it through online belief-tree search. To handle a large crowd and achieve real-time performance in this very challenging setting, we propose LeTS-Drive, which integrates online POMDP planning and deep learning. It consists of two phases. In the offline phase, we learn a policy and the corresponding value function by imitating the belief tree search. In the online phase, the learned policy and value function guide the belief tree search. LeTS-Drive leverages the robustness of planning and the runtime efficiency of learning to enhance the performance of both. Experimental results in simulation show that LeTS-Drive outperforms either planning or imitation learning alone and develops sophisticated driving skills.**

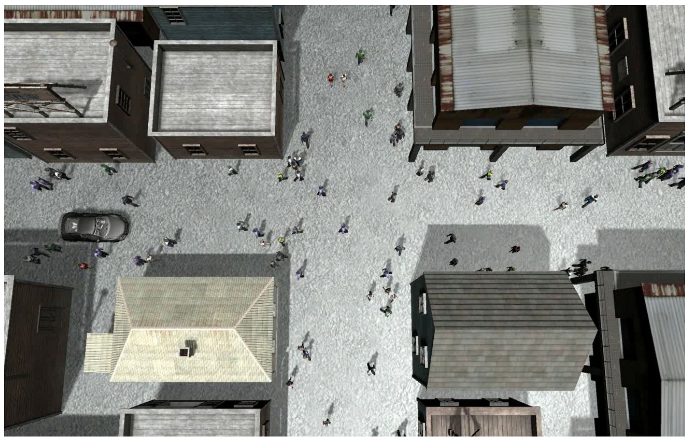
Fig. 1: Autonomous driving in a crowd. A robot vehicle drives amidst many moving pedestrians. Each pedestrian moves towards his/her own goal while interacting with other pedestrians and the vehicle.

I. Introduction

Experienced humans can drive in crowded streets, markets, or squares without colliding with any others and make their way efficiently through the crowd. However, robot vehicles can easily fail in these scenarios. The environment is highly dynamic, comprising many agents interacting with each other, while the robot only has limited perception capabilities. Autonomous driving in a crowd remains an open problem. This paper studies a representative of such problems (Fig. 1): driving among a crowd of moving pedestrians in a map-constrained environment. Typical approaches for crowd driving using local collision avoidance [1, 2] not only generate jerky and zig-zags motions, but can also be easily trapped in local optima, resulting in the vehicle getting stuck in the crowd. Successful driving requires more sophisticated skills, e.g., detouring to by-pass pedestrians or inching forward to make space in a dense crowd. The key here is to perform long-term planning: predict the motion of pedestrians for multiple steps and plan for the vehicle accordingly. However, the planning problem is extremely challenging. A planning algorithm needs to reason in a high-dimensional, partially observable state space formed by surrounding pedestrians, and needs to model a plethora of uncertainties: noisy sensing, unknown intentions of pedestrians, and complex interactions between them. Failing to handle these uncertainties leads to severe or even fatal accidents.

A principled approach for planning under uncertainty is the Partially Observable Markov Decision Process (POMDP). POMDP captures partially observability in a *belief*, which is a probability distribution over states, and reasons about the stochastic effects of robot actions, sensor information, and environment dynamics on the belief. Complex problems require online planning: perform a look-ahead search in a *belief tree* to compute a *policy*, execute the first action in the policy, and re-plan at each time step. However, the computational complexity of belief tree search grows polynomially with the size of the vehicle action space and exponentially with the number of surrounding pedestrians and the planning horizon.

To handle the complexity of driving a vehicle among many pedestrians, state-of-the-art planning methods utilize two approaches: restricting POMDP to control only the accelerations along a pre-planned path [3], and massively parallelizing the planning with GPUs [4]. These approaches achieved impressive performance in driving through real crowds of pedestrians. However, we argue that only controlling accelerations is too restrictive for long-term strategies required in crowded streets of cities like Beijing, Bombay, or Hanoi. We seek to relax this constraint.

In this paper, we scale up POMDP planning to much larger search spaces by integrating it with learning. The idea is to use a learned policy to represent prior knowledge and use planning to further optimize the policy for a particular problem instance. Our method learns two deep neural networks to help the search: a policy network that guides search to useful parts of the search space, and a value network to estimate the value of belief subtrees without having to search them. We further exploit the domain knowledge in planning to design our deep neural networks: we use a Gated Path Planning Network (GPPN) [5], a neural network that approximates the value iteration algorithm, to provide an initial plan, which is refined by another neural network component.

Our proposed method, LeTS-Drive, first executes imitation learning [6] using an existing belief tree search algorithm [4] as an expert to generate training data. It then uses the learned policy and value functions as heuristics to guide the belief tree search, which generates actions to drive the vehicle. LeTS-Drive exploits the robustness of planning and the runtime efficiency of learning, and integrates them to advance the capabilities of both. On one hand, LeTS-Drive uses the learned policy to characterize the robot's long-term behaviors and avoid searching a deep tree. On the other hand, it exploits a prediction model at hand to optimize the robot's short-term behaviors, and in the meantime, corrects possible mistakes raised by neural networks.

LeTS-Drive is inspired by the AlphaGO [7] that integrates Monte Carlo tree search (MCTS) with neural networks to learn board games like GO, Shogi, and chess from self-play. However, crowd driving is substantially different and much more challenging than board games. First, instead of competing with one opponent, the robot vehicle interacts with many agents who further interact with each other. In this case, it is intractable to search over the actions of all agents. Instead, we predict pedestrians' motion by reasoning about their intended navigation goals and modeling their reciprocal interactions. Second, states in board games are fully observable and action executions are deterministic. However, in crowd driving, intentions of pedestrians are partially observable: the vehicle can not directly observe pedestrians' navigation goals. One can only model pedestrians' intentions with a belief, which needs to be inferred from the action-observation history. Both planning and learning have to be performed in the enormous-scale belief space.

Our results in simulation show that LeTS-Drive outperforms either planning or imitation learning alone in both seen and novel environments. It successfully develops sophisticated driving skills, and enables the vehicle to achieve its long-term goal more safely and efficiently.

II. Background and Related Work

A. Online POMDP Planning

A major challenge in real-world planning tasks is *partial observability*: system states are not known exactly and one can only receive observations from the world, which reveals some information about the true state. POMDP offers a principled way to handle partial observability and uncertainties in sensing and control.

Formally, a POMDP model is represented as a tuple (S, A, Z, T, O, R), where S represents the state space of the world, A denotes the set of all possible actions, and O represents the observation space. The transition function T characterizes the dynamics of the world. When the robot takes an action a at state s, the world transits to a new state s' with a probability $T(s,a,s') = p(s'|s,a)$. After that, the robot receives an observation z with probability $p(z|s',a) = O(s',a,z)$, and also a real-valued reward $R(s,a)$.

To perform planning, the robot maintains a belief b, represented as a probability distribution over S. POMDP planning searches for a policy $\pi : B \rightarrow A$ which prescribes an action a that optimizes future values at each belief b. For infinite horizon POMDPs, the value of a policy π at a belief b is defined as the expected total discounted reward achieved by executing the policy π from b onwards:

$$V_{\pi}(b) = E\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(b_t)) \; |b_0 = b\right) \tag{1}$$

Online POMDP planning interleaves planning and execution to efficiently handle complex tasks. Assume that the robot starts from an initial belief b_0. At each time step t, the planning computes an optimal action a^* for the current belief, executes it immediately, and re-plans for the next time step. Online planning usually performs a look-ahead search from the current belief b. The planning constructs a belief tree and searches for an optimal policy π^* that maximizes the value: $V_{\pi^*}(b) = \max_{\pi}\{V_{\pi}(b)\}$. The robot then executes the first action in the optimal policy and updates the current belief based on the action a^* taken and the observation z_t received. The belief update, denoted as $b_t = \tau(b_{t-1}, a, z)$, is performed using the Baye's rule:

$$b_t(s') = \eta O(s', a, z_t) \sum_{s \in S} T(s, a, s') b_{t-1}(s) \tag{2}$$

where η is the normalization constant. The new belief b_t then becomes the entry of the next planning cycle.

POMDP planning suffers from the well-known "curse of dimensionality" and "curse of history" [8]: the complexity of planning grows exponentially with the size of the state space and the planning horizon. Two recent belief tree search algorithms, POMCP [9] and DESPOT [10] made online POMDP planning practical for real-world tasks. Both of them use Monte Carlo simulations to sample transitions and observations. POMCP [9] performs Monte Carlo tree search (MCTS) on the belief space. The DESPOT algorithm [10] samples a fixed set of scenarios and constructs a sparse belief tree conditioned under these scenarios. DESPOT further performs branch-and-bound pruning in the belief tree. Both POMCP and DESPOT are able to solve moderate-scale POMDP problems efficiently, while DESPOT achieves significantly better worst-case performance. DESPOT also offers a better chance for parallelization, enabling a massively parallel planner, HyP-DESPOT [4].

B. Navigation or Driving in a Crowd

Existing work in crowd driving or navigation fall into three main categories: local collision avoidance, learning-based, and planning-based approaches.

One type of local collision avoidance algorithms is social force [11–13] that assumes pedestrians are driven by virtual attraction forces exerted by their navigation goals and repulsing forces exerted by obstacles. Social force-based algorithms are suitable for simulating crowds, but can easily be stuck in local optima when controlling robots. Another class are Velocity Obstacle (VO) [14] and RVO [15–17]-based methods. They

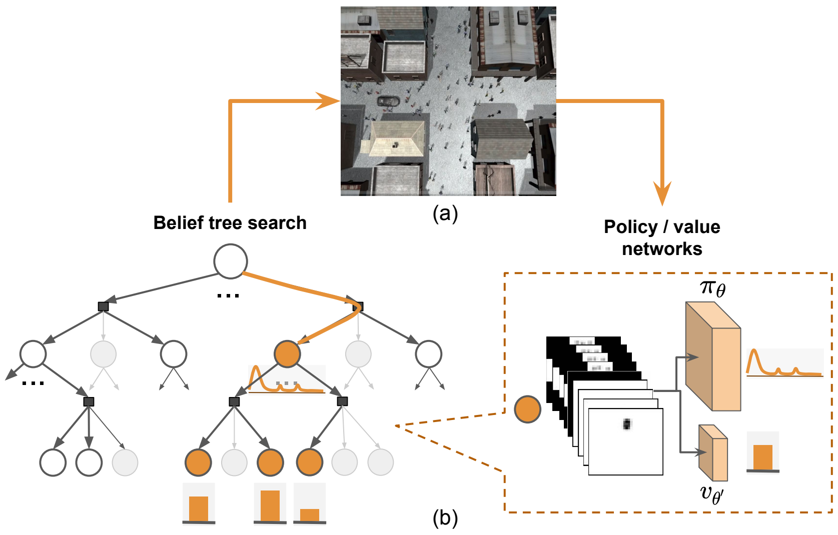

Fig. 2: Overview of the two stages in LeTS-Drive. (a) "Imitation": training a policy network and a value network using a belief tree search expert, and (b) "Improvement": guide the belief tree search with the learned policy and value.

compute command velocities by performing optimization in the feasible velocity space. A representative is ORCA [1], that models homogeneous agents assuming that they perform collision avoidance with each other in a reciprocally optimal way. Recently, PORCA [2] improves the model to handle pedestrian-vehicle interactions.

Learning-based approaches fall into two categories. Some learn to predict pedestrian motions [18, 19] and use the predictions for navigation planning. These approaches lack a mechanism to handle uncertainties in the learned model and thus could be sensitive to prediction errors. Other methods directly learn a navigation policy using imitation learning [20] or reinforcement learning [21–23]. These approaches enabled robots to successfully navigate among real pedestrians. However, learning from an ORCA expert limits the learned policy to imitate local collision avoidance behaviors. Instead, our expert (Section III-B) demonstrates global planning for the policy network.

Planning-based approaches suffer from high computation burden. To make the problem tractable, prior methods [2, 3] decoupled the driving actions and solved two sub-problems: a path planner to determine the steering angle, and a POMDP to control the acceleration. The decoupled approach significantly reduced the computational cost, but it also breaks the cooperation between steering and acceleration, which is often the key to sophisticated driving: human drivers often speed up and maneuver in the meantime to cut through others' way.

C. *Integrating Tree Search and Learning*

LeTS-Drive is inspired by AlphaGO [7], AlphaGO Zero [24], and AlphaZero [25] that integrate Monte Carlo Tree Search with neural networks to learn board games like GO, Shogi, and chess from self-play. The idea has also been applied to Hexagon [26], another perfect information two-player game. Recently, DeepStack [27] extends the idea to solve a partial information poker game using a value function based on the belief of the opponent's hands.

LeTS-Drive faces two major challenges in applying the scheme to real-world robotics tasks. First, instead of a single opponent, the vehicle interacts with many pedestrians in crowd driving. It is intractable to search over the actions of all pedestrians. Instead, we use a motion model (Section III-A3) to predict pedestrians' motion, and condition the model on their intended navigation goals. Second, such pedestrian intentions are partially observable to the vehicle. We can only infer a belief over pedestrians' intentions from the interaction history. LeTS-Drive maintains the belief using Bayesian filtering [28]. Then, it plans in the belief space, and conditions the neural networks on history states. Note that although DeepStack also includes a notion of belief, the immediate consequences of player's actions are deterministic. In crowd driving, transitions of pedestrians are highly uncertain due to their complex interactions, producing significantly more complex belief trees to search.

III. LeTS-Drive

LeTS-Drive integrates POMDP planning and deep learning to drive a vehicle among many moving pedestrians. The method (Fig. 2) contains an *imitation* stage and an *improvement* stage.

In the imitation stage (Fig. 2(a)), LeTS-Drive uses neural networks to approximate the global planning behavior of a belief tree search expert. It generates demonstration trajectories in simulation to train a policy network and a value network, the architectures of which are designed using our domain knowledge in navigation. These networks take as input the current history state and the intention of the vehicle, and predict distributions on the vehicle's steering and acceleration, as well as the expected value of the policy.

In the improvement stage (Fig. 2(b)), LeTS-Drive uses the learned policy and value functions to guide the belief tree search, leveraging their prior knowledge to efficiently explore promising paths and avoid searching deep subtrees. The guided search constructs an improved policy over the learned one, and corrects critical errors made by the neural networks. At runtime, LeTS-Drive executes the guided search to drive a vehicle.

In this section, we present our approach in detail, including the POMDP model, the architectures and the learning procedure of the neural networks, as well as how they are used to guide the belief tree search.

A. The POMDP model

1) State and Observation Modeling: A state in crowd driving consists of the vehicle state and the states of 20 nearest pedestrians. The vehicle state consists of its 2D position, heading direction, and instantaneous speed. A pedestrian state contains its position and speed, as well as the intention represented as his/her navigation goal. We assume that positions and velocities are fully observable, and pedestrians adopt a finite set of possible navigation goals, i.e., destination locations, as known for each environment. However, their actual intentions are partially observable to the vehicle, and thus need to be modeled as beliefs and must be inferred from the interaction history.

2) Action Modeling: Sophisticated driving relies on collaborations of steering and acceleration. Therefore, we plan in a two-dimensional joint action space comprising the steering of the front wheel, discretized using a resolution of 5 degrees, and the acceleration of the vehicle, containing three discrete values, ACCELERATE, DECELERATE, and MAINTAIN.

3) Transition Modeling: The transition function models the dynamics of the vehicle and nearby pedestrians. We use a bicycle model to transit the vehicle according to specific steering and acceleration commands. For pedestrians, we use PORCA [2] to predict their motion conditioned on the inferred intentions. PORCA assumes that pedestrians optimize their instantaneous velocities to approach the intended goals and comply with collision avoidance constraints in the meantime. The model further assumes that interactions among pedestrians are "reciprocal", and vehicles and pedestrians share different responsibilities in reciprocal collision avoidance due to their own limitations.

4) Reward Modeling: The reward function encourages the vehicle to drive safely, efficiently, and smoothly. For safety, we give a large penalty $R_{\text{col}} = -1000 \times (v^2 + 0.5)$, varying with the driving speed v, to the vehicle if it collides with any pedestrian. For efficiency, we assign a small cost $R_{\text{time}} = -0.1$ to each time step and issues a reward $R_{\text{goal}} = 0$ to the vehicle when it reaches the goal. For smoothness of the drive, we add a small penalty $R_{\text{acc}} = -0.1$ if the action performs ACCELERATE or DECELERATE, to penalize the excessive speed changes.

B. Learning from Tree Search

In the imitation stage, we train our neural networks using a belief tree search expert. Note that it is intractable to directly solve the POMDP in Section III-A. Instead, we use Decoupled-Action-POMDP [2, 3] as the expert. The algorithm decouples the planning of steering and acceleration to achieve real-time performance. At each time step, it plans a path using Hybrid A* [29] and restricts POMDP to control only the accelerations along the path.

1) Dataset: We first use Decoupled-Action-POMDP to drive the vehicle in simulation and generate demonstration trajectories. The trajectories are sparsely sampled to form a data set D. Each data point in D, denoted as (H, I, α, a, V), where the history state H and the reference path I are inputs to the neural networks, and the actions α, a and the value V are labels to predict. Particularly, $H = (H_c, H_e)$ represents a fixed-length history of the vehicle and the environment (moving pedestrians and static obstacles), respectively. This history not only encodes the current state and the dynamics of the involved agents, but also reveals the intentions of pedestrians. The reference path I, generated by Hybrid A* considering only static obstacles, represents the intention of the robot vehicle. Finally, labels in the data point record the steering α and the acceleration a chosen by the expert, and the accumulative reward V that the vehicle collected from the current time step till the end of the trajectory.

2) Training: Using this dataset, LeTS-Drive trains a policy network parameterized by θ:

$$\pi_\theta : \ (H_c, H_e, I) \rightarrow (f(\alpha), f(a))$$

where $f(\alpha)$ and $f(a)$ are distributions over the steering and the acceleration of the vehicle. We also fit a value network parameterized by θ':

$$v_{\theta'} : \ (H_c, H_e, I) \rightarrow V$$

where V is the predicted value of the history state.

The policy network π_θ and the value network $v_{\theta'}$ are trained separately using supervised learning. The loss functions, $l(\theta, D)$ and $l(\theta', D)$, measure the errors in action and value predictions, respectively:

$$\begin{aligned} l(\theta, D) &= H_\alpha(\theta, D) + H_a(\theta, D) && (3) \\ l(\theta', D) &= MSE_V(\theta', D) && (4) \\ &= \frac{1}{N}\sum_i^N (v_{\theta'}(H_i, I_i) - V)^2 && (5) \end{aligned}$$

where

$$\begin{aligned} H_\alpha(\theta, D) &= \frac{1}{N}\sum_i^N (-\alpha_i \log \pi_\theta^\alpha(H_i, I_i)) && (6) \\ H_a(\theta, D) &= \frac{1}{N}\sum_i^N (-a_i \log \pi_\theta^a(H_i, I_i)) && (7) \end{aligned}$$

Here, N is the size of the data set, H represents the cross-entropy loss [30] of action predictions, and MSE denotes the mean square error of value predictions. All inputs (H_c, H_e, I) are initially encoded as 1024×1024 images and down-sampled to 32×32 using Gaussian pyramids [31] before inputting to the neural networks.

C. Guided Belief Tree Search

In the improvement stage, LeTS-Drive uses HyP-DESPOT [4], a massively parallel belief tree search algorithm, to plan vehicle motions. LeTS-Drive incorporates the prior knowledge learned in the policy and value networks into the heuristics of HyP-DESPOT in order to search efficiently within the limited planning time.

For completeness, we provide a brief summary of HyP-DESPOT. See [4] for details. HyP-DESPOT samples a small set of K scenarios as representatives of the stochastic future. Each scenario, $\boldsymbol{\phi} = (s_0, \varphi_1, \varphi_2, ...)$, contains a sampled initial state s_0 and random numbers $\varphi_1, \varphi_2, ...$ that determinize the outcomes of future actions and observations. HyP-DESPOT iteratively constructs a sparse belief tree conditioned on the sampled scenarios. Each node b of the tree contains a set of scenarios $\boldsymbol{\Phi}_b$, whose starting states represents a belief. The tree starts from an initial belief. It branches on all actions, but only on observations encountered under the sampled scenarios. In each trial, HyP-DESPOT starts from the root node b_0 and searches a single path down to expand the tree. At each node along the path, HyP-DESPOT chooses an action branch and an observation branch according to the heuristics computed using an upper bound and a lower bound value, u and l. It ends a trial when it is no longer beneficial, and immediately backs-up the upper bounds and lower bounds to update the root. The search terminates until the gap between the upper and lower bounds at the root is sufficiently small or the planning time limit is reached.

LeTS-Drive integrates HyP-DESPOT with the neural networks to solve the joint-action POMDP efficiently. During the forward search, LeTS-Drive uses the learned policy to bias action selections, so that the search prefers to explore actions chosen by the expert:

$$a^* = \arg\max_{a \in A} \left\{ u(b,a) + c\pi_\theta(x_b, a) \sqrt{\frac{N(b)}{N(b,a)+1}} \right\} \tag{8}$$

Here, the first term exploits actions with higher upper bound values $u(b,a)$. The second term is the exploration bonus considering the visitation counts of the node, $N(b)$, and that of the action a under the node, $N(b,a)$. It also depends on the prior probability $\pi_\theta(x_b, a)$, of choosing action a at the history state x_b, as suggested by the policy network π_θ. In effect, the heuristics trade off the exploitation of high-quality branches and the exploration of less-visited ones with high prior probabilities.

When encountering a new belief node, LeTS-Drive queries the value network to provide a tight estimation of the lower bound and expedite the convergence of the search:

$$l_0(b) = v_{\theta'}(x_b) \tag{9}$$

where $v_{\theta'}(x_b)$ is the value predicted by the value network $v_{\theta'}$ at history state x_b. This tight estimation can otherwise only be acquired by sufficiently searching the corresponding subtree.

Fig. 2(b) illustrates the guided belief tree search in LeTS-Drive.

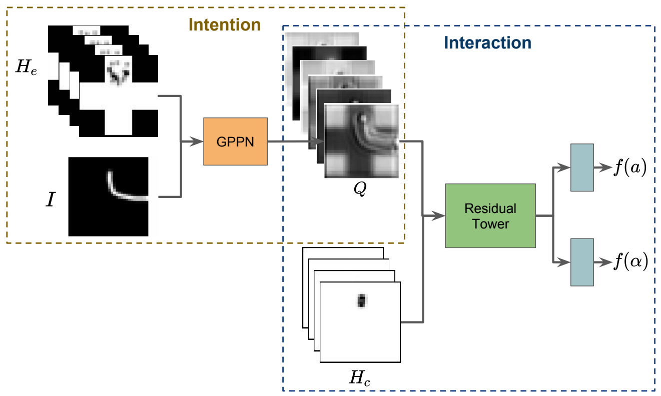

Fig. 3: The architecture of the LeTS-Drive policy network with an intention module and an interaction module. The intention module takes as input the dynamic environment H_e and the intended path I of the vehicle, and outputs action-values Q for all locations on the map. Then, the interaction module takes these action-values, and outputs action probabilities according to the current location and the history of the vehicle.

D. Neural Network Architecture

LeTS-Drive requires the neural networks to represent high-quality policies with low computational complexity. We design the neural network architectures according to the domain knowledge in planning. An optimal policy should convey two key aspects: conduct the vehicle's intention and interact with pedestrians to avoid collisions. Therefore, our policy network first computes an initial plan using the vehicle's intention, then refines the policy to incorporate interactions with another neural network module.

Particularly, our policy network has an "intention" module and an "interaction" module. The intention module takes as input the dynamic environment and the intention of the vehicle, and outputs *action-values*, i.e., the value to be achieved if the vehicle takes a specific action at the current step, for all locations on the map. Then, the interaction module takes these action-values, and outputs action probabilities according to the current location and the history of the vehicle.

The architecture of the intention module follows the spirit of Value Iteration Networks (VINs) [32], which embeds an MDP model [33] and the value iteration algorithm [34] in a recurrent neural network to perform end-to-end training for the model and the algorithm simultaneously. Formally, we use a Gated Path Planning Network (GPPN) [5] which is an improved variant of VIN that provides faster training and better generalization. It first maps the history of the environment H_e and the vehicle's intended path I into reward images, then performs recurrent convolutions using an LSTM [35] on the reward images to perform general value iterations on the map. The intention module outputs a set of action-value images Q. Channels of Q correspond to actions encoded in a learned latent space, and each pixel in Q corresponds to a location on the map.

The interaction module uses these Q images to determine

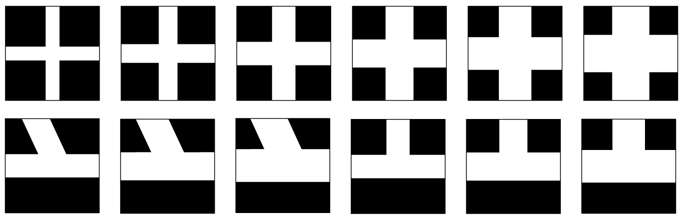

Fig. 4: Driving maps with varying road widths and layouts used for training: black regions represent static obstacles (i.e., buildings) and white regions represent roads that the vehicle and pedestrians can drive or walk on. Road widths vary from 8 meters to 16 meters.

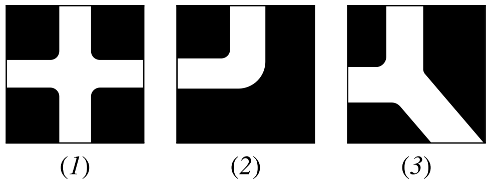

Fig. 5: Maps for generalization test of LeTS-Drive.

the vehicle's action according to the vehicle's current history. A typical way is to use a simple attention mechanism for this: extract the Q values at the vehicles current position and calculate the best actions from the extracted action-values [32]. However, this approach fails in dynamic environments, because values on the map will change through time. Instead, LeTS-Drive uses a Convolutional Neural Network (CNN) [36] to account for the complex interactions between the vehicle and the dynamic environment. The interaction module first stacks the Q images with the current history of the vehicle H_c and pass them through a residual tower concatenating 11 convolutional layers. The images are first processed with 2 residual blocks [37] with 32 filters with kernel size 3×3 and stride 1. The processed images are then down-sampled to 16×16 using a convolutional layer with 64 filters with kernel size 7×7 and stride 2 followed by batch normalization and rectifier nonlinearity. The down-sampled images further pass through 3 residual blocks with 64 filters with kernel size 3×3 and stride 1 and finally input to two action heads to produce steering and acceleration commands. The heads first apply a 50% dropout on the images, Then, it uses 4 filters of kernel size 1×1 to obtain 4 feature images, flattens them as a 1-dimensional vector and uses a fully-connected layer to produce the final predictions. Fig. 3 illustrates the high-level architecture of this policy network.

The value network in LeTS-Drive needs to be queried much more frequently in the belief tree search (Section III-C) than the policy network, thus needs to be computationally light-weighted. Therefore, we removed the GPPN module from the value network. Instead, it directly stacks the dynamic environment H_e, the history of the vehicle H_c and the intended path I and processes them with a similar but smaller residual tower as that in the interaction module. This residual tower only has 2 residual blocks after the down-sampling layer and uses only 16 filters in the last 5 convolutional layers. The output images from the residual tower are passed through a value head, which is similar to the action heads in structure, to produce value predictions for the current history state.

IV. Results and Discussions

We used LeTS-Drive to drive a vehicle in simulated crowds to analyze its performance. Our results show that global planning is important for driving in a crowd, and our policy network can efficiently learn the behavior of global planning. By further integrating the learned policy with the belief tree search, LeTS-Drive achieves superior driving performance in simulation, compared to local collision avoidance, POMDP planning, and imitation learning.

A. Driving Simulation

The driving simulator is built using a sophisticated game engine Unity3D. We built 12 different driving maps of size $40m \times 40m$ to evaluate LeTS-Drive (Fig. 4). Six of them are crossroad maps with different road widths, and the other six are junctions. Roads in the maps are occupied by crowds of people, each pedestrian has his/her own destinations unknown to the robot vehicle a priori. To generate realistic scenes, we simulate the crowds using PORCA, which has been shown to produce accurate predictions of pedestrian motion in the presence of vehicles [2].

In all driving scenarios, we randomized the initial positions and navigation goals of pedestrians, as well as the starting positions and initial directions of the vehicle. In effect, the vehicle faces a new environment in each drive.

B. Performance comparisons

We tested the driving performance of five algorithms, including *NH-PORCA*, *Joint-Action-POMDP*, *Decoupled-Action-POMDP*, *LeTS-Drive-Imitation*, and LeTS-Drive. NH-PORCA performs local collision avoidance for the vehicle. Similar to PORCA [2] for pedestrians, NH-PORCA directly controls the velocity of the vehicle. To generate non-holonomic vehicle motions, it constrains the command velocities around the vehicle's current heading direction, and uses a pure-pursuit controller to execute the velocity. Joint-Action-POMDP applies HyP-DESPOT to directly solve the POMDP in Section III-A. To generate reasonable behaviors within the limited search time, we augmented the reward function to encourage the vehicle drive near the global path. Decoupled-Action-POMDP is the training expert that decouples the planning for steering and acceleration. It also uses HyP-DESPOT to plan for accelerations. LeTS-Drive-Imitation executes our policy network that imitates the Decoupled-Action-POMDP expert. The proposed algorithm, LeTS-Drive, executes the belief tree search guided by the neural networks. In the following experiments, LeTS-Drive, Decoupled-Action-POMDP, and Joint-Action-POMDP use $0.3s$ of planning time (3 Hz frequency), while LeTS-Drive-

TABLE I: Comparisons on the driving performances of LeTS-Drive, Joint-Action-POMDP, Decoupled-Action-POMDP, LeTS-Drive-Imitation, and NH-PORCA in training maps.

	Collision rate	Success rate	Time-to-goal	# Decelerations
NH-PORCA	0.171	0.573	37.2 $\pm$ 0.67	24.6 $\pm$ 1.16
Joint-Action-POMDP	0.016	0.774	33.5 $\pm$ 0.51	53.8 $\pm$ 2.04
Decoupled-Action-POMDP	0.005	0.998	43.2 $\pm$ 0.73	38.8 $\pm$ 1.14
LeTS-Drive-Imitation	0.012	0.946	43.5 $\pm$ 0.96	46.4 $\pm$ 0.89
LeTS-Drive	**0.002**	**0.998**	**29.6 $\pm$ 0.41**	**18.2 $\pm$ 0.54**

TABLE II: Comparisons on the driving performances of LeTS-Drive, Decoupled-Action-POMDP, and LeTS-Drive-Imitation in the three test maps.

Test Map	Algorithm	Collision rate	Success rate	Time-to-goal	# Decelerations
Map 1	Decoupled-Action-POMDP	0.003	1.0	44.5 $\pm$ 0.81	41.6 $\pm$ 1.15
	LeTS-Drive-Imitation	0.003	0.99	38.2 $\pm$ 0.78	26.5 $\pm$ 0.85
	LeTS-Drive	**0.001**	**1.0**	**28.2 $\pm$ 0.47**	**15.7 $\pm$ 0.46**
Map 2	Decoupled-Action-POMDP	0.017	0.98	52.8 $\pm$ 1.10	56.2 $\pm$ 1.57
	LeTS-Drive-Imitation	0.005	0.96	44.8 $\pm$ 1.37	32.8 $\pm$ 0.98
	LeTS-Drive	**0.005**	**1.0**	**28.9 $\pm$ 0.65**	**16.6 $\pm$ 0.51**
Map 3	Decoupled-Action-POMDP	0.002	**1.0**	36.5 $\pm$ 0.70	32.2 $\pm$ 1.07
	LeTS-Drive-Imitation	**0.0**	0.94	31.6 $\pm$ 0.64	23.6 $\pm$ 0.86
	LeTS-Drive	0.0007	0.99	**24.6 $\pm$ 0.50**	**11.2 $\pm$ 0.40**

Imitation and NH-PORCA generate control commands at 10 Hz.

The driving performance of the algorithms is measured using safety, indicated by the portion of trials where the vehicle collides with pedestrians or static obstacles (*collision rate*), and efficiency, measured by the rate of reaching the goal within 2 minutes (*success rate*) and the average time used to reach the goal in successful trials (*time-to-goal*). We further include the number of decelerations (*# decelerations*) to measure the smoothness of a drive. Table I shows the average performance of around 1000 drives in simulation for each tested algorithm.

We first conclude that local collision avoidance is insufficient for driving in a crowd. NH-PORCA suffers in both safety and efficiency of driving as compared to planning, imitation learning, and LeTS-Drive. It often fails to reach the goal and frequently collides with pedestrians. This is due to the incapability of multi-step look-ahead, which is crucial for driving in a dynamic environment.

Joint-Action-POMDP drives much more safely than NH-PORCA because it simulates the dynamics of the vehicle and pedestrians using forward search. However, due to the complexity of searching in the joint action space, the algorithm often fails to generate long-term plans. Consequently, the vehicle misses the goal in around 22 % of drives and generates jerky motions with frequent decelerations. In contrast, Decoupled-Action-POMDP achieves much better real-time performance.

The performance of LeTS-Drive-Imitation, matching up with its expert Decoupled-Action-POMDP, shows that our policy network (Section III-D) learns effectively from global planning.

Finally, by integrating belief tree search with the learned policy and value functions, LeTS-Drive significantly outperforms Decoupled-Action-POMDP and LeTS-Drive-Imitation, achieving the lowest collision rate, and reducing the traveling time by roughly 30 % and the number of decelerations by more than 50 %.

C. Generalization

Since LeTS-Drive uses neural networks in belief tree search, it is important that the guided search generalizes to unseen environments. Results in Section IV-B show that LeTS-Drive generalizes very well on randomized positions and intentions of pedestrians. In this section, we further built 3 novel maps (Fig. 5) to inspect the generalization capability of LeTS-Drive. The test maps have different layouts of static obstacles, which also affect the distribution and dynamics of pedestrians. Table II shows the driving performance of algorithms on the test maps.

The first test map has a similar crossroad layout as training maps (0)-(6) but has novel road positions, widths, and corner shapes. Both LeTS-Drive and LeTS-Drive-Imitation maintain their high performance as in the training maps. Among them, LeTS-Drive with guided belief tree search achieves the best safety, efficiency, and smoothness.

The second and third test maps have very different road layouts from the training maps. LeTS-Drive still achieves significantly higher performance than either planning or imitation learning alone. Noticeably, the success rate of imitation learning decreases in the third map because the layout is significantly different. In contrast, LeTS-Drive recovers the performance by applying additional search upon the learned networks, achieving 99% of success rate, and improving the efficiency by 22% and the smoothness by 53%, approximately.

D. Case Studies

Human drivers reveal sophisticated driving behaviors in daily life: they drive around others who block their way, and can interact with pedestrians to make their way through even in very limited space. These behaviors, however, are hard to acquire

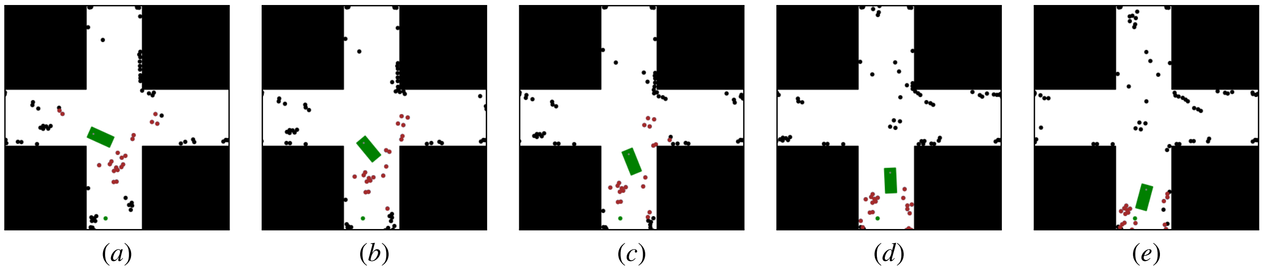

Fig. 6: LeTS-Drive case study: The vehicle moving south maneuvers to by pass pedestrians. The vehicle and its destination are shown in green. Pedestrians considered in the belief tree search are shown in brown color.

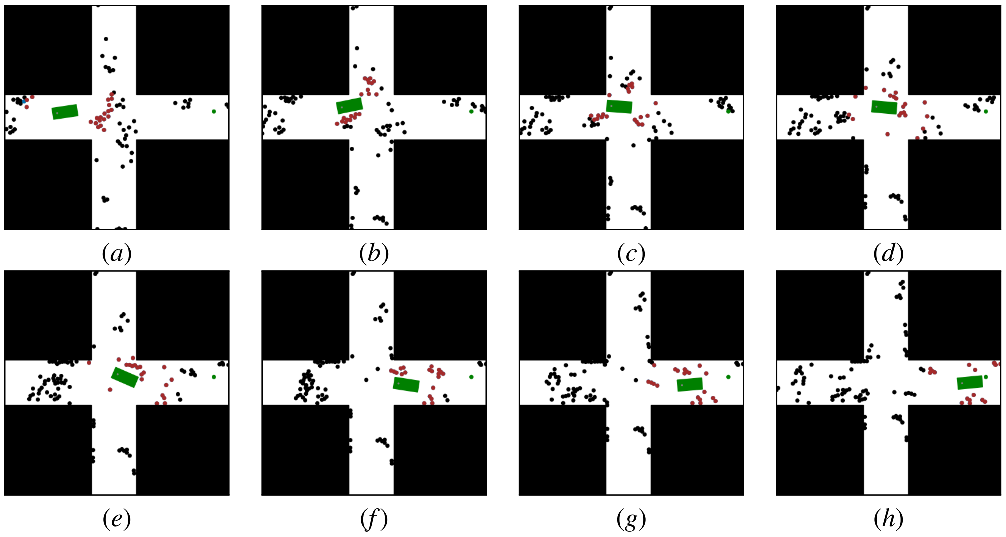

Fig. 7: LeTS-Drive case study: the vehicle (heading east) makes its way through the crowd by interacting with pedestrians. The vehicle and its destination are shown in green. Pedestrians considered in the belief tree search are shown in brown color.

for autonomous vehicles. In this section, we demonstrate that LeTS-Drive developed sophisticated driving behaviors after performing sufficient search.

In Fig. 6, a group of pedestrians walk in right-front of the vehicle and blocks the way. Understanding that they intend at a similar destination as the vehicle, LeTS-Drive steered the vehicle around the crowd to take a longer, but more efficient path.

In Fig. 7, many pedestrians walk near the vehicle and block its way and the free-space has only narrow passages. LeTS-Drive can still drive efficiently across the crowd by interacting with pedestrians with local maneuvering.

Check this video https://youtu.be/oghGK3QJFVo for real-time driving cases in the Unity simulator.

V. Conclusion

This work addresses the challenge of autonomous driving among many pedestrians. LeTS-Drive leverages the robustness of planning and the runtime efficiency of learning. It integrates them to advance the performance of both. LeTS-Drive applies a two-stage approach. It learns a policy by imitating the belief tree search and then guides the belief tree search using the learned policy and its value function. LeTS-Drive outperforms the planning or the learning approach alone in both safety and efficiency, and demonstrates sophisticated driving behaviors in simulation.

There are two main limitations in our current work. First, the efficiency of online search is constrained by the complexity of the learned neural networks. To further improve performance, we plan to investigate more efficient policy and value function representations. Second, it remains a challenge to generalize the learned policies and value functions over diverse maps and agent behaviors. We plan to expand the training environment with rich real-world maps and more sophisticated agent motion models.

Acknowledgments

We want to thank Huaiyu Liu for help in implementing the neural networks for tree search. This work is supported in part by the MoE AcRF grant 2016-T2-2-068 and National Research Foundation Singapore through the SMART IRG program.

References

[1] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*, pp. 3–19, Springer, 2011.

[2] Y. Luo, P. Cai, A. Bera, D. Hsu, W. S. Lee, and D. Manocha, "Porca: Modeling and planning for autonomous driving among many pedestrians," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3418–3425, 2018.

[3] 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. on Robotics & Automation*, 2015.

[4] P. Cai, Y. Luo, D. Hsu, and W. S. Lee, "HyP-DESPOT: A hybrid parallel algorithm for online planning under uncertainty," in *Proc. Robotics: Science & Systems*, 2018.

[5] L. Lee, E. Parisotto, D. S. Chaplot, E. P. Xing, and R. Salakhutdinov, "Gated path planning networks," in *International Conference on Machine Learning*, 2018.

[6] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 21, 2017.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[8] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99 – 134, 1998.

[9] D. Silver and J. Veness, "Monte-carlo planning in large POMDPs," in *Advances in Neural Information Processing Systems*, pp. 2164–2172, 2010.

[10] N. Ye, A. Somani, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," *J. Artificial Intelligence Research*, vol. 58, pp. 231–266, 2017.

[11] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, p. 4282, 1995.

[12] R. Löhner, "On the modeling of pedestrian motion," *Applied Mathematical Modelling*, vol. 34, pp. 366–382, 2010.

[13] G. Ferrer, A. Garrell, and A. Sanfeliu, "Robot companion: A social-force based approach with human awareness-navigation in crowded environments," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2013.

[14] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robotics Research*, vol. 17, pp. 760–772, 1998.

[15] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proc. IEEE Int. Conf. on Robotics & Automation*, 2008.

[16] J. Van Den Berg, S. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Proc. Int. Symp. on Robotics Research*, 2009.

[17] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *IEEE Transactions on Robotics*, vol. 27, pp. 696–706, 2011.

[18] F. Large, D. A. V. Govea, T. Fraichard, and C. Laugier, "Avoiding cars and pedestrians using v-obstacles and motion prediction," in *Proc. of the IEEE Intelligent Vehicle Symp.*, 2004.

[19] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning motion patterns of people for compliant robot motion," *Int. J. Robotics Research*, vol. 24, pp. 31–48, 2005.

[20] P. Long, W. Liu, and J. Pan, "Deep-Learned Collision Avoidance Policy for Distributed Multiagent Navigation," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 656–663, 2017.

[21] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 285–292, IEEE, 2017.

[22] 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.

[23] T. Fan, X. Cheng, J. Pan, D. Monacha, and R. Yang, "Crowdmove: Autonomous mapless navigation in crowded scenarios," *arXiv preprint arXiv:1807.07870*, 2018.

[24] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[25] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[26] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," in *Advances in Neural Information Processing Systems*, pp. 5360–5370, 2017.

[27] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[28] S. Särkkä, *Bayesian filtering and smoothing*, vol. 3. Cambridge University Press, 2013.

[29] T. S. Stanley, "The robot that won the darpa grand challenge: research articles," *J. Robotics System*, vol. 23, pp. 661–692, 2006.

[30] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

[31] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984.

[32] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

[33] D. J. White, "Markov decision processes," *Encyclopedia of Statistical Sciences*, vol. 7, 2004.

[34] R. Bellman, *Dynamic programming*. Courier Corporation, 2013.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.