Robotics: Science and Systems 2021
Held Virtually, July 12–16, 2021

MAGIC: Learning Macro-Actions for Online POMDP Planning

Yiyuan Lee, Panpan Cai and David Hsu
School of Computing, National University of Singapore

***Abstract*—The partially observable Markov decision process (POMDP) is a principled general framework for robot decision making under uncertainty, but POMDP planning suffers from high computational complexity, when long-term planning is required. While temporally-extended *macro-actions* help to cut down the effective planning horizon and significantly improve computational efficiency, how do we acquire good macro-actions? This paper proposes Macro-Action Generator-Critic (MAGIC), which performs offline learning of macro-actions optimized for online POMDP planning. Specifically, MAGIC learns a macro-action *generator* end-to-end, using an online planner's performance as the feedback. During online planning, the generator generates on the fly situation-aware macro-actions conditioned on the robot's belief and the environment context. We evaluated MAGIC on several long-horizon planning tasks both in simulation and on a real robot. The experimental results show that the learned macro-actions offer significant benefits in online planning performance, compared with primitive actions and handcrafted macro-actions.**

I. INTRODUCTION

The partially observable Markov decision process (POMDP) [12] is a principled approach for planning under uncertainty, and has been successful in numerous real-world robotics settings [11, 22, 28, 29, 50]. Many real-world tasks require long-horizon reasoning and involve a continuous action space. However, state-of-the-art online POMDP planners suffer from an exponential complexity w.r.t the planning horizon and the size of the action space. *Macro-actions* serve as a promising tool to cut down the exponential complexity of planning. However, it is challenging to acquire a good, compact set of macro-actions for a given task. Automatically constructing macro-actions is costly, often requiring extensive computation over the state space using carefully-defined criteria [9, 20, 24, 26, 27]. Moreover, a good choice of macro-actions is often situation-specific, making them hard to handcraft.

We propose to learn situation-aware open-loop macro-actions from data and to plan using these learned macro-actions. This is achieved by learning a macro-action *generator* – a mapping from the current belief and context (collectively, the online *situation*) to a finite set of macro-actions. The set is then used by a planner to perform efficient online planning. The objective of the learning is to find the optimal macro-action generator that maximizes the downstream planning performance. We argue that this cannot be efficiently solved using typical learning methods: it is hard to manually design effective situation-aware macro-actions for supervised learning; the space of macro-action sets is also exponentially larger than the robot's action space, making exhaustive search or trial-and-error schemes futile.

This paper presents a novel learning method, *Macro-Action Generator-Critic* (MAGIC) (Fig. 1), to learn the generator from the feedback of online planning. The key observation is, value estimates output by the planner represent the best task performance achievable when using a macro-action set. By optimizing the generator w.r.t. the value estimates, we learn the generator end-to-end, *directly* for the task performance. This is done by learning a differentiable surrogate objective – a *critic*, to approximate the value function of the planner. We can then apply the critic to optimize the generator via gradient-ascent. After learning offline, the generator is deployed to generate macro-action sets for online planning.

The architecture of Macro-Action Generator-Critic can be related to Actor-Critic [13], if we loosely regard the generator as a "meta-actor" interacting with both the planner and the physical environment as a joint entity. The core difference is, our generator does not learn from raw environment rewards, but from the values fed back by the planner.

We evaluate MAGIC on various long-horizon planning tasks that include continuous action spaces, dynamic and interactive environments, and partial observability. Results show that MAGIC learns high-quality macro-actions that lead to superior performance for all tasks, outperforming primitive actions and handcrafted macro-actions. MAGIC can efficiently solve large-scale realistic problems such as driving in urban crowds, and has been successfully deployed to a real-world robot platform for mobile manipulation.

II. BACKGROUND

A. POMDP Preliminaries

A POMDP [12] is written as a tuple $\langle S, A, T, R, \Omega, O, \gamma \rangle$, where S, A, Ω denote spaces of states, actions and observations respectively. The *reward function* $R(s, a)$ quantifies the desirability of taking action a at state s. The *transition function* $T(s, a, s') = p(s' \mid s, a)$ represents the dynamics of the world, specifying the probability of transiting from s to s' by taking action a. The *observation function* $O(a, s', o) = p(o \mid s', a)$ specifies the probability of observing o from sensors after taking action a to reach s'.

POMDPs capture partial observability of the system using a *belief* – a probability distribution over s, denoted as $b(s)$. A POMDP policy $\pi : B \to A$ prescribes an action for all beliefs in the *belief space* B, the set of all possible probability distributions over S. Solving a POMDP requires finding the *optimal policy* maximizing the expected future reward, or the *value*, for all $b \in B$. The value of a policy π at belief b is defined as:

$$V_\pi(b) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(b_t)) \mid b_0 = b\right]. \quad (1)$$

B. Online POMDP Planning

POMDPs can be solved either *offline* using dynamic programming or *online* using forward search. Offline POMDP solvers [1, 19, 33, 44] try to generate the full policy for all belief states. However, they suffer heavily from the "curse of dimensionality", since the size of the belief space is exponential to that of the state space.

Online POMDP planners [36, 40] replan at each step, and focus only on reachable contingencies from the current belief. This is typically conducted with a *belief tree search* over a tree, rooted at the current belief and containing all reachable future beliefs within the planning horizon, via recursively branching over actions and observations. State-of-the-art belief tree search planners [18, 40, 51] leverage Monte Carlo sampling and heuristic search to scale up. A representative example is the DESPOT [51] algorithm, which samples K *deterministic scenarios* to approximate the uncertain future. DESPOT reduces the complexity of belief tree search by an exponential factor in the observation space, and can handle continuous states and complex observations. One can also bias action selection using a learned prior policy function [2, 41, 42, 43].

The limitation of DESPOT, however, is the requirement of a small, discretized action space. The complexity of planning with a large or even continuous action space is still considered intractable. A possible workaround is to adaptively sample the action space and to progressively widen the belief tree to focus only on useful subsets of the original action space. This class of algorithms includes POMCPOW [46], GPS-ABT [39], QBASE [48], and CBTS [30]. These methods make POMDP planning under continuous action space practical for conceptual or simple real-world problems. However, the complexity of planning over a long horizon still prohibits their application to complex real-world tasks.

C. Planning with Macro-Actions

Macro-actions are a form of temporal abstraction to tackle long-horizon planning, by reducing the planning depth linearly and thus the planning complexity exponentially. For deterministic planning and MDPs, a macro-action can be represented as a sequence of primitive actions or a local trajectory [6, 10, 16, 37]. Such trajectories can be extracted from past experience, typically generated by planning using primitive actions. A more sophisticated formulation of macro-actions is Semi-MDP (SMDP) [47]. A macro-action in SMDPs is referred to as an "option", a temporally-extended close-loop policy for reaching sub-goals [27, 45], sub-regions [26], landmarks [25], or executing low-level skills [14, 32]. A separate high-level planning problem is formulated as an MDP, with an abstract action space consisting of all available options.

In POMDPs, macro-actions can also be discovered using handcrafted criteria. For offline planning, MIGS [20] and IGRES [24] pre-build a road-map over the state space by sampling states with high reward gain or information gain, and use the nodes and edges of the road-map to form macro-actions. For online planning, PUMA [9] uses sub-goal-oriented macro-actions. It first evaluates all states using reward and information gain heuristics. During online belief tree search, it samples the best-evaluated states as sub-goals, and solves the induced local MDPs to produce macro-actions.

Previous algorithms, however, can hardly scale-up to large-scale problems due to the complexity of pre-computing offline macro-actions over the entire state space. The above algorithms have also decoupled the problem of constructing macro-actions with that of downstream planning. In this paper, we seek to learn open-loop macro-actions optimized *directly* for planning in an end-to-end fashion.

D. Learning to Plan

The use of learning to improve planning has been a field of recent interest. A natural and effective approach is to learn planner sub-components, such as dynamics/observation models [5, 8], heuristics [3, 35], and sampling distributions [21], and to leverage them to scale up planning for complex domains. Contrary to prior works, our approach learns action sets for planning – specifically, a set of open-loop action sequences (or, *macro-actions*) to constrain long-horizon planning under continuous action spaces. The idea of learning macro-actions is also related to learning motion primitives or low-level skills [15, 17, 38, 49], which can be used for task and motion planning. Here, instead of manually specifying a set of skills and learning them independently, MAGIC seeks to automatically discover the most efficient ones, by learning to maximize the planning performance *directly* in an end-to-end fashion.

III. Overview

MAGIC learns situation-aware macro-actions offline and leverages them for long-horizon online planning. We achieve this by learning a macro-action *generator* from planning experience. MAGIC consists of three key components: an online *planner*, a learned macro-action *generator*, and a learned *critic* to facilitate generator learning.

We represent each macro-action, denoted by m, as an open-loop trajectory, where we assume a suitable parameterization is given according to a task. For instance, 2D Bezier curves (Fig. 11a) can be used for planning with holonomic mobile robots. These trajectories are discretized into sequences of primitive actions for both planning and execution.

Our planner, Macro-DESPOT (Section IV), conditions online planning on a set of candidate macro-actions, M_Φ, where Φ is the collection of the parameters of all macro-actions in the set. Given a predefined model of the world dynamics and observations, the planner computes an optimal close-loop plan over macro-actions for the robot, depending on the current *belief* b over world states and the *context* c of the environment. At each time step, the planner also outputs a *value estimate* v, which approximates the optimal task performance achievable using online planning when conditioned on the given macro-action set M_Φ. The key

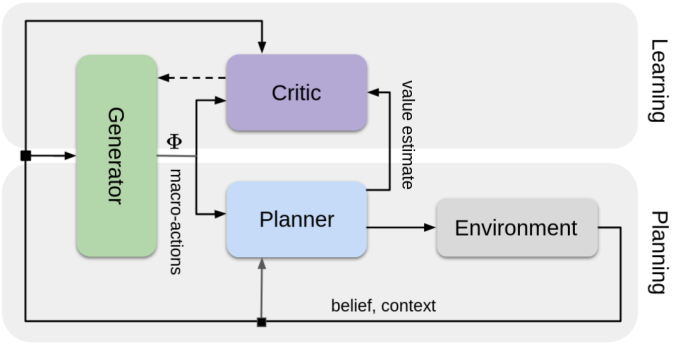


Fig. 1: Overview of MAGIC. MAGIC learns a macro-action *generator*, which prescribes a macro-action set, Φ, for efficient online planning, conditioned on the belief and the environmental context. The generator is trained to maximize the *planner*'s performance via gradient-ascent (the dashed arrow), using a *critic* as a differentiable surrogate objective.

Algorithm 1: Online planning using learned macro-actions

1: $e \leftarrow$ INITENVIRONMENT()
2: $b \leftarrow$ INITBELIEF()
3: **while** ISNOTTERMINAL(e) **do**
4: $\quad \Phi = \mathbb{E}[G_\theta(b, c)]$
5: $\quad m, v \sim$ MACRO-DESPOT(b, c, Φ)
6: $\quad$ **for** a in $m = (a_1, \ldots, a_L)$ **do**
7: $\quad\quad e, o \sim$ EXECUTEACTION(e, a)
8: $\quad\quad b \leftarrow$ UPDATEBELIEF(b, a, o)

observation is that better macro-action sets will induce higher values. We thus seek to learn macro-actions that maximize this value.

In particular, we learn a *generator*, a neural network, G_θ, that outputs a most-suitable macro-action set M_Φ for each online situation specified by (b, c) (Section V). Such a set is characterized by a uni-modal distribution over Φ. We seek to optimize the parameters θ of the generator, such that the *expected planner performance*,

$$\mathop{\mathbb{E}}_{\Phi \sim G_\theta(b,c)}[V(b, c, \Phi)], \tag{2}$$

is maximized at every belief b and context c, with the expectation taken over the generator's output distribution.

The above objective function is, however, very expensive to evaluate for all online situations. We instead learn a *critic*, another neural network $\hat{V}_\psi$, to approximate the planner's value function V. Using the critic as a differentiable surrogate objective, the generator G_θ is trained to maximize Eq. (2) via gradient ascent.

Fig. 1 shows the workflow of the algorithm in the training phase. The planner acts episodically, as illustrated in Algorithm 1. At each time step, the current belief b and context c are first input to the generator to produce parameters Φ for the candidate macro-action set M_Φ (Line 4). The planner then uses M_Φ to perform long-horizon planning, taking b, c, and Φ as inputs, and outputs the optimal macro-action $m \in M_\Phi$ (Line 5). The robot executes m (Line 7) and updates the tracked belief using the received observations (Line 8). Additionally, the data point (b, c, Φ, v) is sent to a *replay buffer*, which holds past experience for learning. The learner optimizes both the critic and the generator using gradient ascent. In each iteration, it samples a batch of data from the replay buffer. Using the data, the learner first updates the critic to fit the planner's value estimates; it then updates the generator to maximize for the value approximated by the critic as a differentiable surrogate objective. The updated generator is then used for planning at the next time step. Acting and learning are repeated until convergence or reaching a given limit on training time or data.

At test time, we deploy the generator together with Macro-DESPOT to perform online planning. The process follows Algorithm 1, similarly as in the training phase.

IV. Online Planning with Macro-Actions

In this section, we present the Macro-DESPOT algorithm in detail, which modifies a state-of-the-art belief tree search algorithm, DESPOT [51], to perform macro-action-based planning. For completeness, we first provide a brief summary of DESPOT, followed by our modifications.

A. DESPOT

DESPOT samples a set of *scenarios* to represent the current belief and the uncertain future. Each scenario encodes a sampled initial state and a sequence of random numbers that *determinize* the effect of future actions and observations. Using the sampled scenarios, DESPOT constructs a sparse belief tree that ensembles Monte Carlo simulations of the future, generated using a black-box *simulative model* – a deterministic step function conditioned on the sampled scenarios. Each node in the belief tree contains a set of scenarios whose corresponding states collectively form an approximate representation of a belief. The belief tree starts from the current belief and branches over all actions, but only on observations encountered under the sampled scenarios. To obtain the optimal policy and its value estimate, DESPOT backs up rewards in the search tree using the Bellman equation:

$$V(b) = \max_{a \in A} \left\{ R(b, a) + \gamma \sum_{o \in \Omega} p(o \mid b, a) V(b') \right\} \tag{3}$$

where b and b' denote an arbitrary belief node and a corresponding child of that node; o denotes the observation leading to b'; V is the value estimate maintained for each belief node and γ is the discounting factor that prioritizes short-term returns over long-term returns.

The DESPOT algorithm additionally performs anytime heuristic search to incrementally construct the tree. We refer readers to [51] for the precise algorithmic details.

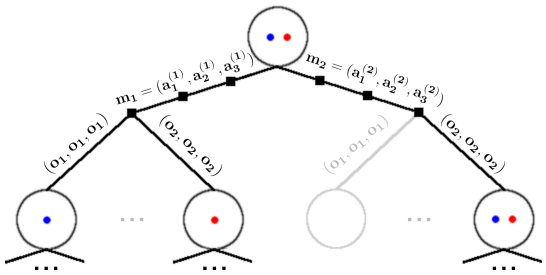


Fig. 2: Macro-DESPOT– online planning using learned macro-actions. A sparse belief tree is searched, branching over possible selections of macro-actions (m_1 and m_2) and over macro-observations visited under sampled scenarios (red and blue dots). The macro-actions step through multiple actions at once to cut down the exponential complexity.

B. Macro-DESPOT

Macro-DESPOT (Fig. 2) modifies DESPOT by replacing the primitive action space A with a set of macro-actions M_Φ. Under each belief node b, the Macro-DESPOT tree branches over all macro-actions $m \in M_\Phi$. Each branch corresponds to a macro-action m being executed entirely in the form of a sequence of primitive actions, $(a_1, \ldots, a_L)$, with the environment stepped along the way using the simulative model. Under each sampled scenario, the macro-action leads to a sequence of observations, $(o_1, \ldots, o_L)$, which is treated as a single *macro-observation* $o \in \Omega^L$. The belief tree further branches with distinct macro-observations encountered under different scenarios, leading to different child belief nodes b'.

The backup process in Macro-DESPOT is accordingly modified from the Bellman equation (Eq. (3)) to work with macro-actions and macro-observations, yielding

$$V(b) = \max_{m \in M_\Phi} \left(R(b,m) + \gamma^L \sum_{o \in \Omega^L} p(o \mid b, m) V(b') \right) \quad (4)$$

where $R(b,m) = \sum_{i=1}^{L} \gamma^{i-1} R(b_{i-1}, a_i)$ represents the cumulative reward along the local trajectories generated by executing m, in expectation over b; and where $p(o \mid b, m) = \prod_{i=1}^{L} p(o_i \mid b_{i-1}, a_i)$ represents the likelihood of receiving the macro-observation o when executing m from b. In both $R(b,m)$ and $p(o \mid b, m)$, b_i represents the belief updated from b using the partial sequence of actions, $(a_1, \ldots, a_i)$, where $b_0 = b$ is simply the parent belief.

The belief tree is expanded in an anytime fashion until the allowed planning time is exhausted. Thereafter, Macro-DESPOT outputs the optimal macro-action under the root node to be executed by the robot. During the training phase, it additionally outputs a *value estimate* v of the optimal macro-action, which approximates the optimal performance that the robot can achieve when constraining online planning to M_Φ. It can thus be interpreted as a *quality measure* of M_Φ for the given situation, and is used as the learning signal to train the macro-action generator.

V. Learning Macro-Actions for Online Planning

We propose a novel method to learn an optimal generator that maximizes the expected planning performance defined in Eq. (2). We decompose the learning problem into two stages. First, we learn a *critic* function, $\hat{V}_\psi$, to model the core element in Eq. (2) – the value function of the planner, V. Then, we train the *generator*, G_θ, using the critic as a surrogate objective, to approximately maximize Eq. (2). The critic and the generator are trained jointly from scratch, and will converge toward the true objective (Eq. (2)) and its optimal solution, respectively, when continuously provided with data under proper optimization.

We represent both the critic and the generator as neural networks, and train them using the planner's experience via gradient descent. We represent an experience or a data point as (b, c, Φ, v), where b is the input belief, c is the context, Φ denotes parameters of the macro-action set used for planning, and v is the optimal value as estimated by Macro-DESPOT.

Due to the stochastic nature of the planner's value estimates, we use a Gaussian distribution to represent each $\hat{V}_\psi(b, c, \Phi)$. We also make the generator's output $G_\theta(b, c)$ distributional, parameterized as a Gaussian, to enable random exploration at training time. Network architectures used for the critic and the generator are presented in Appendix A and shown in Fig. 9 and Fig. 10, respectively.

A. Learning the Critic

MAGIC directly fits the critic to the planner's value function, by minimizing, for all b, c, and Φ, the KL-divergence between the output distribution of the critic, $\hat{V}_\psi(b, c, \Phi)$, and the actual distribution of the planner's value estimates. Particularly, given a data set D, MAGIC updates $\hat{V}_\psi$ to maximize the log-likelihood of the observed value estimates:

$$J(\psi) = \mathop{\mathbb{E}}_{(b,c,\Phi,v)\sim D} [\log p_\psi(v)], \quad (5)$$

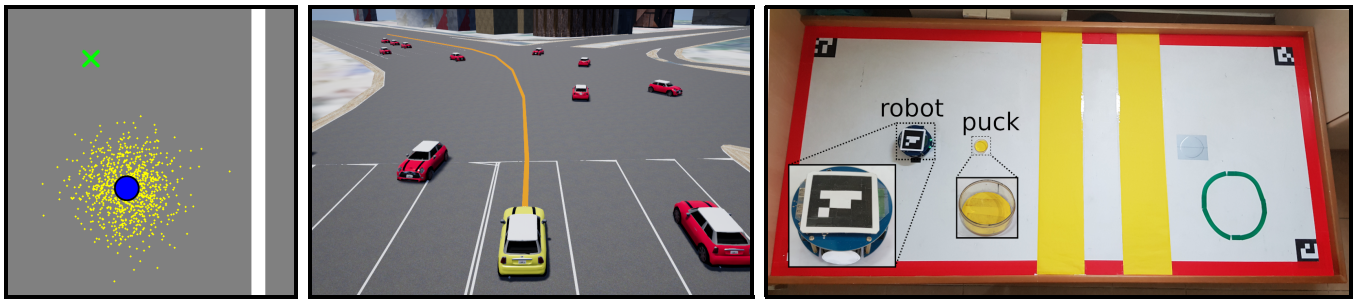
where p_ψ denotes the probability density function corresponding to $\hat{V}_\psi(b, c, \Phi)$.

B. Learning the Generator

MAGIC optimizes the generator to output, for any (b, c), a distribution over macro-action set parameters Φ, that maximizes the expected planning performance as estimated by the critic. Given a data set D, we update G_θ using gradient descent, to maximize the following objective:

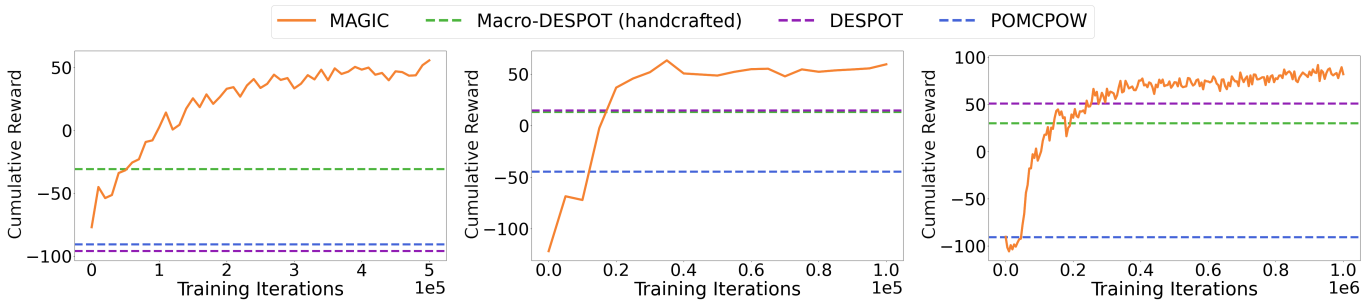
$$J(\theta) = \mathop{\mathbb{E}}_{(b,c)\sim D} \left[\mathop{\mathbb{E}}_{\Phi \sim G_\theta(b,c)} \left[\mathbb{E}[\hat{V}_\psi(b, c, \Phi)] \right] + \alpha \mathcal{H}(G_\theta(b, c)) \right]. \quad (6)$$

The first term is a surrogate objective approximating Eq. (2) and allows gradients to propagate back to the generator. We include an additional entropy regularization term over the generator's output distribution, $G_\theta(b, c)$, which is used to enforce a minimum level of exploration during training. When the regularization weight α is gradually annealed to zero, the above objective will converge to Eqn. (2).



(a) Light-Dark (b) Crowd-Driving (c) Puck-Push

Fig. 3: (a) Light-Dark task. The robot (blue) navigates to a goal (green) through darkness (gray background), using the light (white strip) to localize itself. Particles show the robot's belief over its position. (b) Crowd-Driving task. The robot vehicle (yellow) drives through a busy intersection, making maneuvers to follow the intended path (orange) steadily without colliding with the other vehicles (red). (c) Puck-Push task. The robot pushes a plastic puck (yellow) to a goal region (dark green circle), past two visually-occluding regions (yellow strips).



(a) Light-Dark (b) Crowd-Driving (c) Puck-Push

Fig. 4: Performance (cumulative reward) of MAGIC over training progress for (a) Light-Dark, (b) Crowd-Driving, and (c) Puck-Push. Horizontal lines show the performances of the baselines for comparison.

The original form of Eq. (6) contains a sampling process which is not differentiable. To fix this, we reparameterize G_θ as a deterministic function $f_\theta(\epsilon; b, c)$ conditioned on a random seed $\epsilon \sim \mathcal{N}(0, 1)$. The gradient of Eq. (6) is thus rewritten as:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{(b,c)\sim D}\left[\mathop{\mathbb{E}}_{\epsilon\sim\mathcal{N}(0,1)}\left[\nabla_\theta J(\theta)|_{(b,c,\epsilon)}\right]\right] \quad (7)$$

where $\nabla_\theta J(\theta)|_{(b,c,\epsilon)}$ is the point-wise gradient calculated as:

$$\nabla_\Phi \mathbb{E}[\hat{V}_\psi(b, c, \Phi))]\nabla_\theta f_\theta(\epsilon; b, c) - \alpha \cdot \nabla_\theta \log p_\theta(\Phi) \quad (8)$$

with $\Phi = f_\theta(\epsilon; b, c)$, and p_θ being the probability density function of the distribution $G_\theta(b, c)$ output by the generator. The first term of Eq. (8) applies the chain rule to calculate the gradient of Eq. (6) w.r.t. the generator's parameters, θ.

The regularization weight α in Eq. (6) is automatically adjusted using gradient-ascent to fulfill a desired minimum target entropy $\mathcal{H}_0$ for the generator's output distributions. This is done by maximizing α with the following objective:

$$J(\alpha) = \alpha \cdot \mathop{\mathbb{E}}_{(b,c)\sim D}\left[\mathcal{H}_0 - \mathcal{H}(G_\theta(b^{(i)}, c^{(i)}))\right]. \quad (9)$$

Intuitively, α is increased when the expected entropy of the generator's output falls below the desired target $\mathcal{H}_0$, and decreased otherwise. α is kept non-negative with a rectifier.

Application of the chain rule and entropy regularization is inspired by a model-free RL algorithm, SAC [7], where an entropy-regularized soft-Q function is learned as a surrogate objective to help optimize a policy. Here, our "policy" instead generates an entire macro-action set for downstream planning, and the learning signals are not raw rewards from the environment, but the value estimates from a planner.

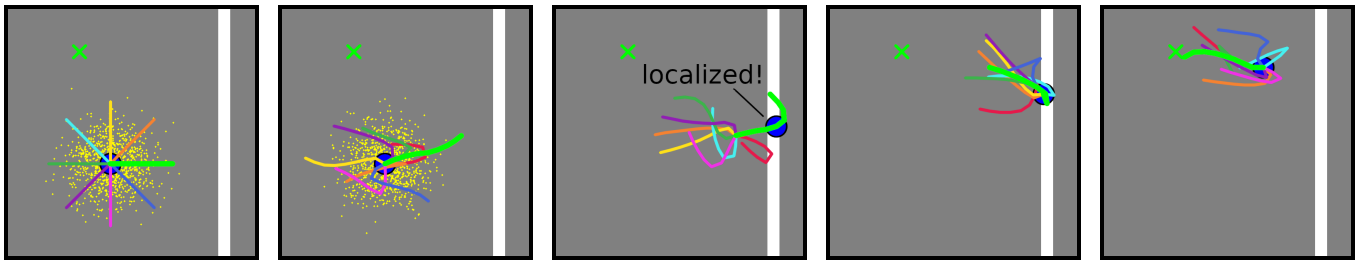
VI. Experiments

In our experiments, we aim to answer a few questions:

1) How much do the learned macro-actions benefit online POMDP planning?
2) Can MAGIC generalize to novel situations and transfer to realistic environments?
3) Can MAGIC scale up to complex tasks?
4) What has MAGIC learned in its macro-action set?

We apply MAGIC to a set of virtual and real-world tasks, including Light-Dark (Fig. 3a), a classical POMDP benchmark requiring active information gathering; Crowd-Driving (Fig. 3b), a realistic task involving interactions with many other agents; and Puck-Push (Fig. 3c), a real-world robot mobile manipulation task. These tasks cover important challenges in robot planning such as partial observability, acting and sensing uncertainties, complex interactions, and large-scale environments. Each task has a continuous action space and critically requires long-horizon reasoning.

To understand the advantage of planning using macro-actions, we first compare MAGIC with primitive-action-based planners. These include *DESPOT* [51], which uses discretized primitive actions to perform belief tree search;



(a) (b) (c) (d) (e)

Fig. 5: Handcrafted (a) and learned (b-e) macro-actions for Light-Dark, with the belief over the robot's position shown as yellow particles and the optimal macro-actions chosen by the planner highlighted in bright green: (b) episode starts; (c) robot localized; (d-e) learned macro-actions concentrate toward the goal post-localization.

TABLE I: Performance (cumulative reward) of MAGIC and planning algorithms across all evaluation tasks. Numbers in brackets represent standard errors (throughout this paper).

	Light-Dark	Crowd-Driving (POMDP Sim.)	Puck-Push (POMDP Sim.)
MAGIC	**54.1 (1.0)**	**58.6 (2.0)**	**87.9 (1.0)**
Macro-DESPOT (Handcrafted)	-30.9 (1.0)	13.1 (2.0)	34.0 (2.0)
DESPOT	-96.1 (1.0)	14.8 (2.0)	53.0 (2.0)
POMCPOW	-90.7 (0.5)	-44.7 (1.0)	-94.1 (1.0)

TABLE II: Detailed performance of MAGIC and planning algorithms on Light-Dark.

	Acc. Reward	Success Rate (%)	Steps Taken[1]	Min. Tracking Error
MAGIC	**54.1 (1.0)**	**79.0 (0.5)**	37.6 (0.1)	**0.28 (0.01)**
Macro-DESPOT (handcrafted)	-30.9 (1.0)	37.1 (0.5)	44.7 (0.1)	0.67 (0.01)
DESPOT	-96.1 (1.0)	4.9 (0.5)	59.8 (0.2)	1.59 (0.01)
POMCPOW	-90.7 (0.5)	6.2 (0.3)	**22.1 (0.4)**	1.55 (0.01)

and *POMCPOW* [46], which progressively samples primitive actions from the continuous action space. To further understand the advantage of *learning* macro-actions, we also compare MAGIC with *Macro-DESPOT (handcrafted)*, which uses handcrafted macro-actions instead of learned ones.

Our results show that macro-actions tremendously benefit long-horizon planning over continuous action spaces. By efficiently learning the macro-actions, MAGIC fully exploits this benefit, achieving the best planning performance for all evaluation tasks. MAGIC successfully scales up on the Crowd-Driving task, which has an enormous state space and complex dynamics. MAGIC also generalizes well to novel environments. While MAGIC is trained completely in randomized simulations built upon POMDP models, it successfully transfers to a realistic simulator for Crowd-Driving and a real-world setup for Puck-Push.

By visualizing the learned macro-actions, we show that MAGIC learns to cover both reward-exploiting and information-gathering behaviors, which help to focus the search tree. MAGIC also learns to adapt the macro-action sets to different beliefs and contexts, further improving the performance of online planning.

Additional visual results are demonstrated in this video: leeyiyuan.info/links/magic-rss21.

A. Training and Planning Performance

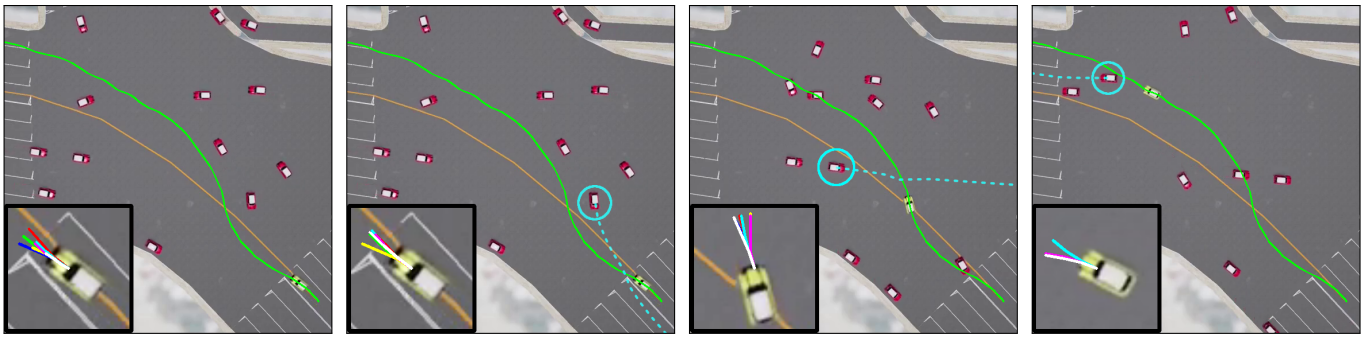
Fig. 4a-c show the learning curves of MAGIC on the three evaluation tasks. As the generator is randomly initialized, the initial performance is relatively low. During training, MAGIC quickly learns to generate useful macro-actions and begins to outperform the best of the other approaches. This happens at around 1.5×10^5, 2×10^4, and 2.5×10^5 updates for Light-Dark, Crowd-Driving, and Puck-Push respectively, corresponding to around 2, 0.4, and 3.5 hours of training using a single Nvidia RTX 2080 GPU with batch-size 256 and 16 workers.

Table I shows the overall planning performance of MAGIC, in terms of the average cumulative reward of episodes, as compared to standard planning algorithms. Planning with macro-actions (MAGIC and Macro-DESPOT (handcrafted)) generally outperforms planning with primitive ones (DESPOT and POMCPOW), because all the evaluation tasks critically require long-horizon policies. Among the macro-action-based planners, MAGIC significantly outperforms handcrafted macro-actions. This is achieved by learning macro-actions which directly maximize the planning performance. The improvements are prominent on all tasks. For Light-Dark, the learned macro-actions help to perform long-term information gathering and precise goal seeking; For Crowd-Driving, they help to tackle the large-scale dynamic environment; For Puck-Push, they assist the robot to reliably manipulate the puck using complex interactions.

Note that MAGIC relies entirely on the POMDP model for both planning and learning, and did not require additional domain knowledge compared to learn good macro-actions. Conceptually, the training process is analogous to the hyperparameter tuning of online planners, where, in contrast, our hyperparameters are the macro-action sets specific to the online situations, which cannot be efficiently tuned in standard ways. MAGIC offers a principled and efficient way to do so.

Next, we investigate the benefits of MAGIC on each evaluation task independently.

[1]Calculated over successful runs.



(a) (b) (c) (d)

Fig. 6: Handcrafted (a) and learned (b-d) macro-actions for Crowd-Driving. The target path is shown in orange. The final trajectory of the robot is shown in green: (b) avoiding incoming vehicle; (b) cutting past incoming vehicle; (c) overtaking.

TABLE III: Detailed performance of MAGIC and planning algorithms on Crowd-Driving in both the POMDP simulator and the SUMMIT simulator.

	POMDP Simulator				SUMMIT			
	Acc. Reward	Distance Covered	Stall Rate	Search Depth	Acc. Reward	Distance Covered	Stall Rate	Search Depth
MAGIC	**58.6 (2.0)**	**111.0 (1.0)**	**0.069 (0.001)**	**22.5 (0.1)**	**32.2 (1.0)**	97.2 (1.0)	**0.100 (0.001)**	**18.3 (0.2)**
Macro-DESPOT (handcrafted)	13.1 (2.0)	98.1 (1.0)	0.099 (0.001)	18.0 (0.2)	4.9 (1.0)	**101.5 (1.0)**	0.285 (0.001)	**18.0 (0.2)**
DESPOT	14.8 (2.0)	91.8 (1.0)	**0.071 (0.001)**	17.6 (0.1)	-8.2 (2.0)	66.7 (1.0)	0.259 (0.002)	11.8 (0.5)
POMCPOW	-44.7 (1.0)	70.4 (1.0)	0.130 (0.001)	2.0 (0.1)	-18.3 (1.0)	75.3 (1.0)	0.312 (0.001)	1.9 (0.1)

B. Light-Dark – active information gathering

1) Task: Light-Dark (Fig. 3a) is adapted from the classical POMDP benchmark in [34]. The robot (blue) attempts to move to and stop *precisely* at a goal (green) in the dark area, where the robot's position can only be observed within the light area (white), thus requiring active information gathering. The goal position, light position, and initial distribution of the robot are *randomly generated* for each episode. The former two constitute the context c of the environment. See a detailed description of the task in Appendix D.

The optimal policy for Light-Dark is to localize within the light before moving to the goal. The light is constrained to be far from the robot's initial position, requiring long-horizon reasoning to discover the value of such active information-gathering actions.

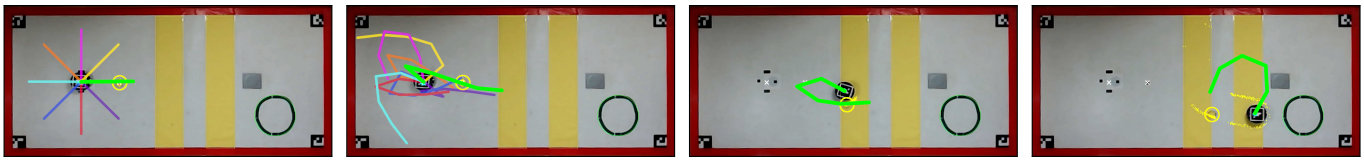
2) Macro-action parameterization: We use 2D cubic Bezier curves (Fig. 11a) to represent macro-actions in Light-Dark. Each macro-action is parameterized by 6 real numbers, corresponding to the 3 control points of the curve. A macro-action is executed by scaling and discretizing the curve into fixed-length line segments, each corresponding to a primitive action; the primitive actions are then executed consecutively. Each macro-action set contains 8 macro-actions each of length 8. Φ is thus a vector of $6 \times 8 = 48$ real numbers, obtained by concatenating the control points of all curves. The handcrafted macro-actions (Fig. 5a) are straight-line trajectories of length 6 pointing to 8 general directions. All macro-action lengths are selected using hyperparameter search.

3) Analysis: Table II shows the detailed performance of the tested algorithms. MAGIC achieves the highest success rate, leading to the best cumulative rewards. A crucial observation is that MAGIC maintains more concentrated beliefs over the robot's position than other algorithms, as shown by the smaller minimum tracking error achieved within episodes. As visualized in Fig. 5, the above advantages are achieved by learning to cover information-gathering trajectories pointing to the light in the starting phase, and to concentrate on goal-seeking trajectories that exploit reward after the robot is well-localized. In contrast, by planning with uniformly-directed handcrafted macro-actions, the robot struggles to reach the goal precisely; primitive-action-based planners such as DESPOT and POMCPOW frequently fail to gather information and do not localize using the light, due to insufficient search depths.

C. Crowd-Driving – complex environments

1) Task: In *Crowd-Driving* (Fig. 3b), a robot vehicle drives along a target path at a busy, unregulated intersection through crowded traffic, seeking to achieve both safety and efficiency when controlled using speed and steering commands. The robot can observe the positions, orientations, and velocities of all vehicles, but requires inferring the exo-agents' intended paths, or *intentions*, from interaction history. All vehicles are *randomly spawned* over the map, seeking to cross the intersection. The context c of the environment is the ego-vehicle's target path, which is also randomly selected. A detailed description of the task is available in Appendix D.

The Crowd-Driving task has an enormous state space, consisting of the joint state of all nearby exo-vehicles, inducing a complex planning problem. Additionally, the uncertain intentions, non-holonomic kinematics, and high-order dynamics of traffic participants require the planner



(a) (b) (c) (d)

Fig. 7: Handcrafted (a) and learned (b-d) macro-actions for Puck-Push, with the belief shown as yellow particles and the optimal macro-actions chosen by the planner highlighted in bright green: (b) episode starts; (c) reversing to re-push; (d) exploring the occlusion region to re-localize the puck.

TABLE IV: Detailed performance of MAGIC and planning algorithms on Puck-Push evaluated on both the POMDP simulator and the real-world setup.

	POMDP Simulator				Real-World Setup			
	Acc. Reward	Success Rate (%)	Steps Taken[1]	Search Depth	Acc. Reward	Success Rate (%)	Steps Taken[1]	Search Depth
MAGIC	**87.9 (1.0)**	**95.3 (0.5)**	35.3 (0.1)	**89.4 (0.1)**	**90.9 (5.2)**	**97.5 (2.5)**	39.4 (2.0)	**83.1 (2.5)**
Macro-DESPOT (handcrafted)	34.0 (2.0)	70.0 (1.0)	42.1 (0.6)	61.0 (0.4)	9.6 (15.9)	58.5 (7.8)	57.1 (4.6)	51.3 (4.2)
DESPOT	53.0 (2.0)	78.6 (1.0)	**26.2 (0.6)**	69.1 (0.5)	3.5 (16.4)	55.0 (8.0)	35.9 (6.2)	64.1 (4.5)
POMCPOW	-94.1 (1.0)	7.6 (0.5)	31.0 (1.7)	2.1 (0.1)	-84.1 (11.0)	12.5 (5.3)	27.8 (13.9)	4.2 (3.0)

to perform sophisticated long-horizon reasoning to avoid collisions and to progress efficiently.

2) Macro-action parameterization: We use *turn-and-go* curves (Fig. 11b) to represent macro-actions in Crowd-Driving. Each macro-action is parameterized by 2 real numbers, $speed$ and $steer$. The first half of the curve turns with a fixed steering angle, $steer$, and a constant command speed, $speed$; the second half drives straight, using the same command speed and with no steering. Actual trajectories of the vehicle are subject to its kinematics and dynamics. Each macro-action set contains 7 macro-actions each of length 3. Φ is thus a vector of $2 \times 7 = 14$ real numbers, obtained by concatenating the parameters of the individual macro-actions. The handcrafted macro-actions (Fig. 6a) are parameterized in the same way, but uniformly covering trajectories with maximum and zero steering, as well as maximum, half-of-maximum, and zero command speeds.

3) Analysis: We train MAGIC using a world simulator built with the above POMDP model, and test MAGIC on a high-fidelity simulator, SUMMIT [4] (Fig. 3b), that features realistic physics. Both simulators use a real-world map replicating the Meskel Square intersection in Addis Ababa, Ethiopia.

Table III shows the performance of MAGIC on both the POMDP simulator and on SUMMIT. MAGIC offers the best driving policy among all algorithms, safely traveling the longest distances with the least amount of stalling. This combination of safety and efficiency can only be achieved by reacting to potential collisions ahead of time. We observe that MAGIC generally searches deeper than other algorithms, as the learned macro-actions induce more focused search.

MAGIC also successfully transfers to random crowds on the SUMMIT simulator, despite the non-negligible differences in the environment dynamics. Consistent with the POMDP results, MAGIC significantly outperforms planning using primitive actions and handcrafted macro actions, thus achieving safer and more efficient driving.

Fig. 6 provides examples of the learned macro-actions. MAGIC learns to include in its macro-action set three types of local trajectories – those that deviate from the target path to avoid exo-vehicles; those that recover back to the path; and those that drive straight with a high speed whenever possible to maximize the driving smoothness. MAGIC adapts the set to different online situations. For instance, in Fig. 6b and 6c, the sets are biased toward smooth driving and collision avoidance with incoming vehicles, while in Fig. 6d, it concentrates on ways to overtake the slow-moving vehicle in front and to subsequently return to the target path.

D. Puck-Push – real robot experiment

In *Puck-Push* (Fig. 3c), the robot attempts to *push* a puck from a start position to a goal region. The robot acts similarly as in Light-Dark, except that it pushes the puck around upon contact. The puck has a relatively smooth surface, and thus slides across the robot's circumference when being pushed. The robot can observe the position of itself and the puck with slight noise via a bird-view camera, except when the puck passes two *occluding regions* (yellow), where the puck's color fuses into the background and thus becomes undetectable. The robot and the puck's initial positions are fixed, but the goal is *randomly generated*. The context c is thus the goal position. See a detailed description of the task in Appendix D.

Puck-Push requires fine interactions with the puck, and mistakes have long-term consequences: if the puck slides past the robot completely during some particular push, the robot is required to navigate back to re-push it toward the goal; if the puck reaches the boundary, it would be impossible to be pushed to the goal. Long-horizon planning is thus critical.

1) Macro-action parameterization: We use 2D Bezier curves (Fig. 11a), similar to Light-Dark, to parameterize macro-actions in Puck-Push. Each macro-action set contains 8 macro-actions each of length 5. Φ is thus a vector of $6 \times 8 = 48$ real numbers, obtained by concatenating the

TABLE V: Ablation study of MAGIC by using only certain situational information.

	Light-Dark	Crowd-Driving (POMDP Sim.)	Puck-Push (POMDP Sim.)
Belief + Context	**54.1 (1.0)**	**58.6 (2.0)**	**87.9 (1.0)**
Belief Only	-51.2 (1.0)	**53.7 (2.0)**	82.5 (1.0)
Context Only	35.1 (1.0)	**56.6 (2.0)**	66.2 (1.0)
Unconditioned	43.9 (1.0)	15.9 (2.0)	64.3 (1.0)

control points of all curves. The handcrafted macro-actions (Fig. 7a) are straight-line trajectories of length 2 pointing to 8 general directions. All macro-action lengths are selected using hyperparameter search.

2) Analysis: We train MAGIC solely on the POMDP simulator and test it on a real-world setup (Fig. 3c). Unlike in simulation, the goal position in the real-world experiment is fixed. Physical coefficients in the POMDP model are manually calibrated using real-world data.

Table IV shows the performance of MAGIC on both the POMDP simulator and on the real robot with comparisons to other planning algorithms. MAGIC achieves a remarkably higher success rate. Compared to planning with primitive actions, MAGIC can avoid unrecoverable mistakes, such as pushing the puck toward the wall, through effective long-horizon reasoning. Compared to the handcrafted macro-actions (Fig. 7a), MAGIC learns much more flexible local trajectories (Fig. 7b-d), allowing reliable interactions with the puck. Particularly, the learned macro-actions include different ways to manipulate the puck, either by pushing it directly toward the goal (Fig. 7b), or by retracting to re-push it whenever the puck slides past the robot (Fig. 7c), or by exploring the occluding regions to re-localize the puck (Fig. 7d). Such diverse and meaningful macro-actions help the planner to discover flexible strategies and search significantly deeper, thus achieving superior performance.

Furthermore, MAGIC successfully transfers to the real-world setup with unstable camera observations and real-world dynamics, and can push the puck to the goal in almost all trials. This transfer capability is enabled by simulating the stochasticity of the real-world using properly randomized transitions and observations in the POMDP model, and applying the randomization in both planning and learning.

E. Ablation study – partially conditioned macro-actions

We additionally conduct an ablation study (Table V) to investigate the performance of MAGIC when partially conditioning the macro-action generator on only the belief or context. We also compare MAGIC with the unconditioned version, which directly learns a fixed macro-action set for all situations using gradient ascent, without the generator.

The results show that MAGIC always performs the best when conditioning on *both* the belief and the context ("belief + context"); it typically works the worst when unconditioned on either information ("unconditioned"). Specifically, removing belief conditioning ("context only") harms both Light-Dark and Puck-Push significantly, where the desirable macro-actions depend strongly on the location of the robot/puck and its uncertainty. Removing context conditioning ("belief only") causes a significant drop in performance for Light-Dark. This is because the task requires knowing the light position for efficient localization and the goal position to precisely stop at it. Without the information, the macro-action set is always scattered even after perfect localization (Fig. 8), which leads to less concentrated macro-actions and thus inefficient planning. This phenomenon is observed for both the unconditioned and the belief-only variants. At the goal-reaching phase, the belief-only variant generates particularly unreasonable macro-actions, because the generator overfits to the goal positions in the training data.

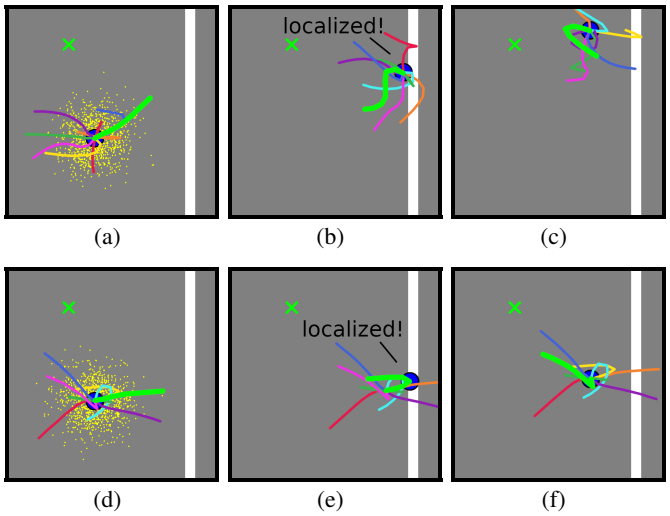


Fig. 8: Learned macro-actions by the belief-only variant (a-c) and the unconditioned variant (d-f) of MAGIC: (a,d) episodes start; (d,e) robot localized; (c,f) goal reaching.

VII. Conclusion

We have presented Macro-Action Generator-Critic (MAGIC), an algorithm to learn a macro-action generator from data, and to use it to empower long-horizon online POMDP planning. MAGIC efficiently learns situation-aware open-loop macro-actions optimized *directly* for planning. By integrating the learned macro-actions with online planning, our planner, Macro-DESPOT, achieves superior performance on various tasks compared to planning with primitive actions and handcrafted macro-actions. We have trained MAGIC in simulation and successfully deployed it on a realistic high-fidelity simulator and a real-world setup. More importantly, the core idea of MAGIC is generalizable to learning other planner parameters other than macro-action candidates. This would be our future direction.

Acknowledgements

This research is supported in part by the A*STAR Undergraduate Scholarship (SE/AUS/15/016), the A*STAR National Robotics Program (Grant No. 192 25 00054), and the National Research Foundation, Singapore under its AI Singapore Program (AISG Award No. AISG2-RP-2020-016).

REFERENCES

[1] Haoyu Bai, David Hsu, Wee Sun Lee, and Vien A Ngo. Monte carlo value iteration for continuous-state pomdps. In *Algorithmic foundations of robotics IX*, pages 175–191. Springer, 2010.

[2] Panpan Cai, Yuanfu Luo, Aseem Saxena, David Hsu, and Wee Sun Lee. Lets-drive: Driving in a crowd by learning from tree search. In *Robotics: Science & Systems*, 2019.

[3] Panpan Cai, Yuanfu Luo, Aseem Saxena, David Hsu, and Wee Sun Lee. Lets-drive: Driving in a crowd by learning from tree search. *arXiv preprint arXiv:1905.12197*, 2019.

[4] Panpan Cai, Yiyuan Lee, Yuanfu Luo, and David Hsu. Summit: A simulator for urban driving in massive mixed traffic. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4023–4029. IEEE, 2020.

[5] Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. *arXiv preprint arXiv:1910.13395*, 2019.

[6] Ricahrd E Fikes, Peter E Hart, and Nils J Nilsson. Some new directions in robot problem solving. Technical report, STANFORD RESEARCH INST MENLO PARK CA, 1972.

[7] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[8] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[9] Ruijie He, Emma Brunskill, and Nicholas Roy. Puma: Planning under uncertainty with macro-actions. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[10] Glenn A Iba. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3(4):285–317, 1989.

[11] Shervin Javdani, Henny Admoni, Stefania Pellegrinelli, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization for teleoperation and teaming. *The International Journal of Robotics Research*, 37(7):717–742, 2018.

[12] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[13] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014. Citeseer, 2000.

[14] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Autonomous skill acquisition on a mobile manipulator. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, 2011.

[15] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[16] Richard E Korf. Learning to solve problems by searching for macro-operators. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1983.

[17] Oliver Kroemer and Gaurav Sukhatme. Meta-level priors for learning manipulation skills with sparse features. In *International Symposium on Experimental Robotics*, pages 211–222. Springer, 2016.

[18] Hanna Kurniawati and Vinay Yadav. An online pomdp solver for uncertainty planning in dynamic environment. In *Robotics Research*, pages 611–629. Springer, 2016.

[19] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland., 2008.

[20] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 30(3):308–323, 2011.

[21] Katherine Liu, Martina Stadler, and Nicholas Roy. Learned sampling distributions for efficient planning in hybrid geometric and object-level representations. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9555–9562. IEEE, 2020.

[22] Wei Liu, Seong-Woo Kim, Scott Pendleton, and Marcelo H Ang. Situation-aware decision making for autonomous driving on urban road using online pomdp. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1126–1133. IEEE, 2015.

[23] Yuanfu Luo and Panpan Cai. Gamma: A general agent motion prediction model for autonomous driving. *arXiv preprint arXiv:1906.01566*, 2019.

[24] Hang Ma and Joelle Pineau. Information gathering and reward exploitation of subgoals for pomdps. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[25] Timothy A Mann, Shie Mannor, and Doina Precup. Approximate value iteration with temporally extended actions. *Journal of Artificial Intelligence Research*, 53: 375–438, 2015.

[26] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71, 2004.

[27] Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. 2001.

[28] Malika Meghjani, Yuanfu Luo, Qi Heng Ho, Panpan Cai, Shashwat Verma, Daniela Rus, and David Hsu. Context and intention aware planning for urban driving. In *2019*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2891–2898. IEEE, 2019.

[29] Philippe Morere, Roman Marchant, and Fabio Ramos. Sequential bayesian optimization as a pomdp for environment monitoring with uavs. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6381–6388. IEEE, 2017.

[30] Philippe Morere, Roman Marchant, and Fabio Ramos. Continuous state-action-observation pomdps for trajectory planning with bayesian optimisation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8779–8786. IEEE, 2018.

[31] Michael E Mortenson. *Mathematics for computer graphics applications*. Industrial Press Inc., 1999.

[32] Scott D Niekum. Semantically grounded learning from unstructured demonstrations. 2013.

[33] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, pages 1025–1032, 2003.

[34] Robert Platt Jr, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. 2010.

[35] Florian Pusse and Matthias Klusch. Hybrid online pomdp planning and deep reinforcement learning for safer self-driving cars. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1013–1020. IEEE, 2019.

[36] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32: 663–704, 2008.

[37] Earl D Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2):115–135, 1974.

[38] Connor Schenck, Jonathan Tompson, Sergey Levine, and Dieter Fox. Learning robotic manipulation of granular media. In *Conference on Robot Learning*, pages 239–248. PMLR, 2017.

[39] Konstantin M Seiler, Hanna Kurniawati, and Surya PN Singh. An online and approximate solver for pomdps with continuous action space. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 2290–2297. IEEE, 2015.

[40] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[41] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[42] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[43] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[44] Trey Smith and Reid Simmons. Point-based pomdp algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412*, 2005.

[45] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

[46] Zachary Sunberg and Mykel Kochenderfer. Online algorithms for pomdps with continuous state, action, and observation spaces. *arXiv preprint arXiv:1709.06196*, 2017.

[47] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[48] Erli Wang, Hanna Kurniawati, and Dirk P Kroese. An on-line planner for pomdps with large discrete action space: A quantile-based approach. In *ICAPS*, pages 273–277, 2018.

[49] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021.

[50] Yuchen Xiao, Sammie Katt, Andreas ten Pas, Shengjian Chen, and Christopher Amato. Online planning for target object search in clutter under partial observability. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8241–8247. IEEE, 2019.

[51] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. *Journal of Artificial Intelligence Research*, 58:231–266, 2017.

APPENDIX

A. Neural network architectures

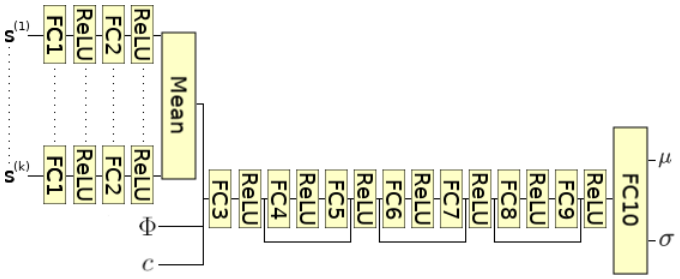


Fig. 9: Architecture of Critic-Net

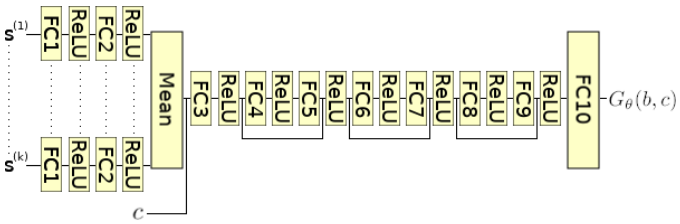


Fig. 10: Architecture of Generator-Net.

Fig. 9 shows the architecture of the Critic-Net. It takes real-valued inputs consisting of the current particle-belief b, the context c, and the macro-action set parameters Φ. Each particle $s^{(i)}$ in b is passed through a FC-ReLU stack, with weights shared across all $s^{(i)}$. This maps the states to a common latent space. The latent particles are then averaged and concatenated with c and Φ, and passed through another FC-ReLU stack with residual connections. The final FC layer in $\hat{V}_\psi$ outputs a mean and a standard deviation representing a Gaussian distribution over the planner's value estimates. Fig. 10 shows the architecture of the Generator-Net. It processes the input belief b using the same parallel FC-ReLU stack as Critic-Net. Then, it concatenates the averaged latent particles with the context c, and passes them through another FC-ReLU stack. The final FC output layer produces a real vector corresponding to the parameters of the distribution over Φ.

B. The training pipeline

Algorithm 2 illustrates the training pipeline in MAGIC. It starts with randomly initialized weights for the generator and critic, θ and ψ (Lines 1-2), together with an empty *replay buffer* (Line 3) to hold experience data. It then launches multiple asynchronous workers to collect episodes and perform training (Lines 4-5). These asynchronous workers share the same neural network weights and the replay buffer, which are access-controlled via mutexes.

A worker resets its environment and its initial belief at the start of each episode (Line 8-9), generating a new context c. Then, it alternates between a planning phase, an execution phase, and a learning phase until the end of the episode. In the planning phase, the worker first queries the generator G_θ (Line 11) for the current belief b and context c to propose the parameters Φ of the macro-action set (Line 12). Macro-DESPOT then uses M_Φ to perform belief tree search; it outputs a macro-action m, together with the quality measure v. The experience (b, c, Φ, v) is then appended to the replay buffer (Line 13). In the execution phase, the worker executes m as a sequence of primitive actions (Lines 14 to 15), updating the belief (Line 16) along the way. Finally, in the learning phase, a mini-batch of experience $\{(b^{(i)}, c^{(i)}, \Phi^{(i)}, v^{(i)})\}_i$ is sampled from the replay buffer (Line 17). It first updates $\hat{V}_\psi$ (Line 18) using Eqn. (5). The updated $\hat{V}_\psi$ is then used to train the generator G_θ (Line 19) via Eqn. (8). Finally, the entropy regularization weight α is adjusted (Line 20) via Eqn. (9).

The three phases repeat until the end of the episode (Line 10), after which the next episode starts. The worker repeats the loop (Line 7) until training terminates. In our experiments, we terminate MAGIC after reaching a certain number of updates to the neural networks (Lines 18-19).

Algorithm 2: Macro-Action Generator-Critic (MAGIC)

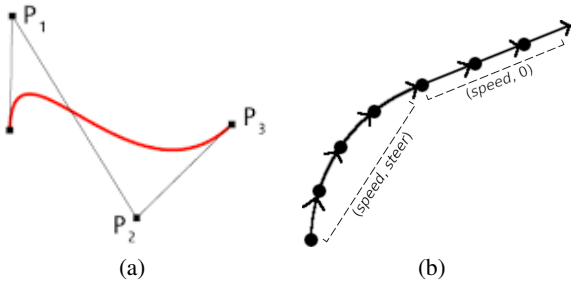
1: $\theta \leftarrow$ INITGENERATORNETWEIGHTS()
2: $\psi \leftarrow$ INITCRITICNETWEIGHTS()
3: $D \leftarrow$ INITREPLAYBUFFER()
4: **for** $i = 1, \ldots, W$ **do**
5: STARTTHREAD(WORKER())
6: **function** WORKER()
7: **while** TRAININGNOTCOMPLETE() **do**
8: $e, c \leftarrow$ INITENVIRONMENT()
9: $b \leftarrow$ INITBELIEF()
10: **while** ISNOTTERMINAL(e) **do**
11: $\Phi \sim G_\theta(b, c)$
12: $m, v \sim$ MACRO-DESPOT(b, c, Φ)
13: $D \leftarrow$ APPEND$(D, (b, c, \Phi, v))$
14: **for** a in $m = (a_1, \ldots)$ **do**
15: $e, o \sim$ EXECUTEACTION(e, a)
16: $b \leftarrow$ UPDATEBELIEF(b, a, o)
17: $b^{(i)}, c^{(i)}, \Phi^{(i)}, v^{(i)} \leftarrow$ SAMPLE(D, N)
18: $\psi \leftarrow \psi + \frac{1}{N} \sum_{i=1}^{N} \nabla_\psi J^{(i)}(\psi)$
19: $\theta \leftarrow \theta + \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta J^{(i)}(\theta)$
20: $\alpha \leftarrow \alpha + \frac{1}{N} \sum_{i=1}^{N} \nabla_\alpha J^{(i)}(\alpha)$

C. Macro-action parameterizations

Fig. 11a shows an example of the macro-action parameterization used for 2D navigation tasks, *i.e.*, Light-Dark and Puck-Push. A macro-action is represented as a 2D Bezier curve [31], defined using three control points P_1, P_2, P_3. The corresponding trajectory is a curve bounded within the convex hull of the control points. Fig. 11b illustrates the turn-and-go curve used for Crowd-Driving, characterized by two real parameters, *speed* and *steer*. Here, *speed* denotes the command speed across the entire trajectory, increasing which results in longer trajectories; *steer* specifies the steering angle used in the first half of the curve, increasing

TABLE VI: Performance (cumulative reward) of MAGIC over different conditionings and macro-action lengths.

Macro-Action Length	Light-Dark				Crowd-Driving (POMDP Simulator)				Puck-Push (POMDP Simulator)			
	Context + Belief	Belief Only	Context Only	None	Context + Belief	Belief Only	Context Only	None	Context + Belief	Belief Only	Context Only	None
2	-91.2 (1.0)	-93.1 (1.0)	-95.5 (1.0)	-89.6 (1.0)	40.4 (2.0)	30.5 (2.0)	36.5 (2.0)	-0.2 (2.0)	79.7 (1.0)	73.0 (1.0)	45.6 (1.0)	53.0 (1.0)
3	-92.1 (1.0)	-92.3 (1.0)	-88.7 (1.0)	-100.4 (1.0)	**58.6** **(2.0)**	**53.7** **(2.0)**	**56.6** **(2.0)**	7.7 (2.0)	74.2 (1.0)	**82.5** **(1.0)**	34.8 (1.0)	58.8 (1.0)
4	-75.9 (1.0)	-82.6 (1.0)	-64.2 (1.0)	-30.4 (1.0)	33.6 (2.0)	30.1 (2.0)	33.8 (2.0)	-12.7 (2.0)	81.1 (1.0)	55.3 (1.0)	54.1 (1.0)	**64.3** **(1.0)**
5	-63.9 (1.0)	-69.4 (1.0)	-48.2 (1.0)	-3.5 (1.0)	43.6 (2.0)	39.8 (2.0)	42.6 (2.0)	**15.9** **(2.0)**	**87.9** **(1.0)**	62.1 (1.0)	**66.2** **(1.0)**	24.6 (1.0)
6	-24.5 (1.0)	-68.8 (1.0)	-22.2 (1.0)	22.2 (1.0)	23.7 (2.0)	24.0 (2.0)	20.5 (2.0)	11.3 (2.0)	65.9 (1.0)	13.7 (1.0)	49.0 (1.0)	46.9 (1.0)
7	22.1 (1.0)	-52.9 (1.0)	**35.1** **(1.0)**	42.7 (1.0)	24.0 (2.0)	18.8 (2.0)	20.1 (2.0)	11.0 (2.0)	60.2 (1.0)	-13.1 (1.0)	16.9 (1.0)	15.8 (1.0)
8	**54.1** **(1.0)**	**-51.2** **(1.0)**	-15.6 (1.0)	**43.9** **(1.0)**	2.1 (2.0)	1.9 (2.0)	-5.1 (2.0)	-3.3 (2.0)	34.0 (1.0)	27.8 (1.0)	-1.5 (1.0)	-13.8 (1.0)



(a) (b)

Fig. 11: Macro-action representations: (a) 2D Bezier curves for Light-Dark and Puck-Push, and (b) turn-and-go curves for Crowd-Driving.

which produces sharper turns.

D. Details of Evaluation Tasks

1) Light-Dark: A mobile robot operates in a 2D world. At every step, the robot can either move along an arbitrary direction for a fixed distance, or *STOP* the robot, hopefully at the goal. A vertical light area exists in the world, within which the robot receives accurate readings of its position, and outside which the robot receives completely no observation of its position. The robot receives a reward of -0.1 at every step, $+100$ for stopping correctly at the goal, and -100 for stopping incorrectly. Otherwise, $STOP$ is automatically executed when exhausting the steps allowed. Each episode allows a maximum of 60 steps. Planners use a maximum search depth of 60, with $0.1\,\mathrm{s}$ of maximum planning time.

2) Crowd-Driving: A robot vehicle drives along a target path at a busy intersection through crowded traffic. The robot can observe the positions, orientations, velocities of all vehicles including the ego-agent's, but not their intentions. The robot is controlled by speed and steering commands, with the maximum speed, steering, and acceleration constrained to be $6.0\,\mathrm{m/s}$, $15°$, and $3.0\,\mathrm{m/s^2}$. Exo-vehicles in the crowd are modeled using GAMMA [23], a recent traffic motion model based on Reciprocal Velocity Obstacles (RVO). Exo-agents are also controlled by speed and steering, with the maximum speed, steering, and acceleration constrained to be $4.0\,\mathrm{m/s}$, $15°$, and $2.0\,\mathrm{m/s^2}$. At each step, the robot receives a reward equal to the distance traveled along the target path, and receives a penalty of -3.61 if the robot's speed falls below half its maximum speed. The episode terminates if a collision occurs, and the robot receives a collision penalty of -100. A maximum of 150 steps is allowed and a planning depth of 40 is used. We execute each macro-action while concurrently planning for the next step. The amortized planning rate is $5\,\mathrm{Hz}$.

3) Puck-Push: A mobile robot pushes a puck to a goal in a 2D workspace. When being pushed, the puck passively slides along the robot's circumference. The sliding motion is modeled by $\theta' = \theta e^{\mu d}$, where θ is the direction of the puck w.r.t. to the robot's ego-frame, d is the distance the robot moved during contact, and μ is a sliding rate coefficient determined by physical properties of the puck. Gaussian noises are added to the motion of both the robot and the puck to simulate uncertain transitions. Observations come from a bird's-eye view camera. When the puck passes the yellow occluding regions, the camera provides no observation. Outside the regions, the perception pipeline also fails occasionally, which is modeled by omitting observations with probability 0.1. The robot receives a reward of -0.1 per step, $+100$ for successful delivery, and -100 for exhausting the allowed steps or colliding with the boundary. Each episode allows a maximum of 100 steps. Planners use a maximum search depth of 100, and are given $0.1\,\mathrm{s}$ of planning time.

E. Detailed results over conditioning and macro-action length

Table VI shows the performance of MAGIC using different macro-action lengths. Performance generally improves with increasing macro-action length up to a certain length, after which performance degrades again. This is because short macro-actions do not cover enough depth for long-horizon planning, while very long macro-actions are hard to learn using gradient ascent.