Recurrent Macro Actions Generator for POMDP Planning

Yuanchu Liang and Hanna Kurniawati
School of Computing, Australian National University
{yuanchu.liang, hanna.kurniawati}@anu.edu.au

***Abstract*— Many planning problems in robotics require long planning horizon and uncertain in nature. The Partially Observable Markov Descision Process (POMDP) is a mathematically principled framework for planning under uncertainty. To alleviate the difficulties of computing good approximate POMDP solutions for long horizon problems, one often plans using macro actions, where each macro action is a chain of primitive actions. Such a strategy reduces the effective planning horizon of the problem, and hence reduces the computational complexity for solving. The difficulty is in generating a set of suitable macro actions. In this paper, we present a simple recurrent neural network that learns to generate suitable sets of candidate macro actions that exploits environment information. Key to this learning method is to represent the raw partial information from the environment as a latent problem instance, and sequentially generate macro actions conditioned on the past information. We compare our proposed method with state-of-the-art [1] on four different long horizon planning tasks with various difficulties. The results indicate the quality of the policies computed using macro actions generated by our proposed method consistently exceeds benchmarks. Our implementation can be accessed at https://github.com/YC-Liang/Recurrent-Macro-Action-Generator.**

I. Introduction

Planning under uncertainty is critical for reliable robot operation. The Partially Observable Markov Decision Process (POMDP) is a principled framework for such planning problems. Although finding the exact solution to a POMDP problem is intractable [2], tremendous advances have been made in approximate POMDP solvers via sampling-based methods [3], computing good solutions to problems that require long planning horizon remains difficult, due to the exponential growth of the searchable action space with respect to planning horizon.

Many methods have been proposed to alleviate this long planning horizon issue. They generally construct a more abstract action, and search for good POMDP solutions using this abstract action rather than the primitive single-step action. As a result, they reduce the effective planning horizon of the problem. Various methods to construct abstract actions have been proposed. Many [4]–[6] use temporally extended sequences of actions where actions or sub-policy are run until some termination conditions are met. These methods require sub-goals or termination conditions to be hand-designed to generate good problem decomposition. Automatic generation of abstract actions or sub-problems have also been proposed, e.g., [7] automatically generates sequences of primitive one-step actions for POMDP solving, [8] uses LQG policies as sub-policies, while [9] constructs abstract actions based on the value of information.

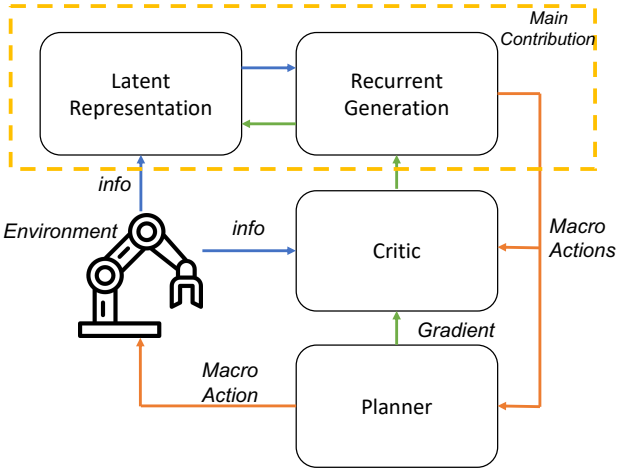


Fig. 1: An illustration of Recurrent Macro Actions Generator (RMAG). Information flow follows the blue arrows. The flow of macro actions are shown in orange arrows. And green arrows indicate the flow of gradients for training. The critic module is dropped during evaluation.

On the other hand, the recent growth of deep reinforcement learning also brings new solutions for decision making problems. Many agents equipped with fine-tuned neural networks are able to learn to map environment inputs directly to good actions [10]–[13]. Other works tackle the problem from a different direction by learning algorithmic components that accelerate the decision making process. In [14], deep learning is used to estimate leaf nodes' values to boost the performance of Monte Carlo Tree Search in mastering the game of Go. Macro actions are learned in [1] to enhance planner's performance by focusing on a small set of quality actions. And, [15] learns the optimal number of multi-layer perceptron (MLP) layers based on input instance.

By leveraging the advantages from the planning and deep reinforcement learning paradigms, we propose Recurrent Macro Actions Generator (RMAG, illustrated in Fig. 1) that learns to automatically generate macro-actions —a set of sequences of primitive one-step actions— that aim to maximise a planner's performance in computing good solutions to planning under uncertainty with long planning horizon. RMAG follows the overall actor-critic architecture of MAGIC [1], where the actor generates candidate macro action sets that maximises the critic's estimated values, while the critic and the planner estimate the planner's performance conditioned on the macro action set generated by the actor. However,

RMAG proposes a novel actor architecture, based on the recurrent neural network, to generate the candidate macro actions that exploits environment information. Specifically, RMAG builds a latent representation of the problem instance based on current beliefs and observations, and progressively infer a macro action set. Empirical evaluation in four environments with different underlying dynamics indicate the efficacy of this actor architecture: The policy generated by RMAG consistently and significantly outperforms other baselines, including those generated by state-of-the-art MAGIC [1], by up to 190%. Ablation studies are carried out to understand the impact of the recurrence module in RMAG.

II. Background

A. POMDP Formulations

A POMDP [16], [17] is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{R} \rangle$. The first three element, $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$, denote the state space, the action space and the observation space respectively. The last three elements in the tuple are functions, where $\mathcal{T}(s, a, s') = p(s'|s, a)$ is the state transition function, $\mathcal{Z}(s', a, o) = p(o|s', a)$ is the probability of receiving an observation given the new state and the current action, and finally the reward function, $\mathcal{R}(s, a)$, is the reward obtained for a given action at a given state.

Due to uncertainty in the transition and observation functions, a POMDP agent only has partial observability of its states. To capture this partial observability, at each time step, the agent maintains a belief $b \in \mathcal{B}$, which is a distribution over the state space $\mathcal{S}$. Here, $\mathcal{B}$ denotes the belief space, which is the set of all possible distributions over the state space $\mathcal{S}$. The best action to perform is then inferred with respect to beliefs. The solution to a POMDP problem is a policy $\pi : \mathcal{B} \to \mathcal{A}$ that maximises the value function:

$$V^*(b) = \max_{a \in \mathcal{A}} \left\{ \mathcal{R}(b, a) + \gamma \sum_{o \in \mathcal{O}} P(o|b, a) V^*(b') \right\} \quad (1)$$

Here γ is a discount factor that discount future rewards and b' denotes the updated belief according to the Bayes's rule,

$$b' = \frac{\mathcal{Z}(s', a, o) \sum_{s' \in \mathcal{S}} T(s, a, s') b}{\sum_{s'' \in \mathcal{S}} Z(s'', a, o) \sum_{s \in \mathcal{S}} T(s, a, s'') b} \quad (2)$$

We refer the reader to [3] for a comprehensive survey on various recent approximate methods for solving POMDPs.

B. Planning with Macro Actions

Despite advances in approximate POMDP solving, computing a good policy for problems that require long planning horizon remains difficult. An approach to reduce the effective planning horizon, and hence ease the computation burden, is to use macro actions. A macro action is a sequence of primitive actions and can be represented as $[a_1, \ldots, a_n]$, where $a_i \in \mathcal{A}$ and n is the length of the macro-action. Constructing a small set of useful macro-actions is difficult because the number of different macro-actions increases combinatorially with the length of the macro-action n.

Various methods are proposed to generate a small set of macro-actions. For instance, a number of works enforce each macro action to reach an explicitly defined sub-goals, sub-regions or landmarks [7], [18]–[21]. In [22], macro actions are used to obtain abstract hierarchies and planning can be carried out efficiently in abstract state space. Macro actions can also be generated according the value of information [9]. MAGIC [1] uses the actor critic method to learn and generate a set of candidate macro actions used by the DESPOT planner to search for the best macro action to take. 2D Bezier curves and turn-and-go curves are used to parameterise the macro actions. The former is used for holonomic robots whereas the later is for non-holonomic robots. A 2D quadratic Bezier curve is controlled by three points $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$ based on the given formula [23],

$$\mathbf{p}(u) = (1-u)^2 \mathbf{p}_0 + 2u(1-u)\mathbf{p}_1 + u^2 \mathbf{p}_2 \quad (3)$$

Here u is called the parametric variable and $0 \leq u \leq 1$. A turn-and-go curve is controlled by two real parameters, speed and steer. Speed is the speed used across the entire trajectory and the steer is the steering angle used in the first half of the trajectory where the second half is always set to 0. During simulation these curves are discretised into a sequence of primitive actions and hence forming a single macro action. In this paper, we follow the overall structure of MAGIC, but proposes a new method to generate situational aware macro actions that aims to maximise the planner's performances.

C. Deep Learning for POMDPs

Many deep neural networks have been proposed to learn various components of a POMDP. One paradigm is the end-to-end learning, where a single deep neural network is used to approximate the entire POMDP [13], [24], [25]. For instance, [24] proposed the QMDP net that embed solution structure into a neural network that is trained end-to-end to predict the best action to take. In contrast, separate modules can be learned to predict state transition or observation likelihood functions [26]–[28]. These functions can be used as a generative model for a planner to perform forward simulation and search for an optimal action to take according to Equation 1. We adopt the second approach and propose a Long Short Term Memory (LSTM) based model to learn suitable macro-actions. Details of RNN can be found in [29]. The method we propose in this paper can by viewed as a type of Actor Critic framework [30], in which a critic is used to guide an actor to learn policies.

III. Methods

A. Overview

Key to RMAG is an RNN architecture that can generate the set of suitable candidate macro-actions, taking into account environment information. This generation process is performed in two stages. First, it learns a compact latent representation of the environment instance based on partial information and observation. Specifically, given current environment information, ξ, RMAG learns a stochastic latent representation h of the problem, denoted as $p(h \mid \xi)$. Since the shape of the macro action depends on the initial condition and subsequent primitive actions, in the second stage RMAG

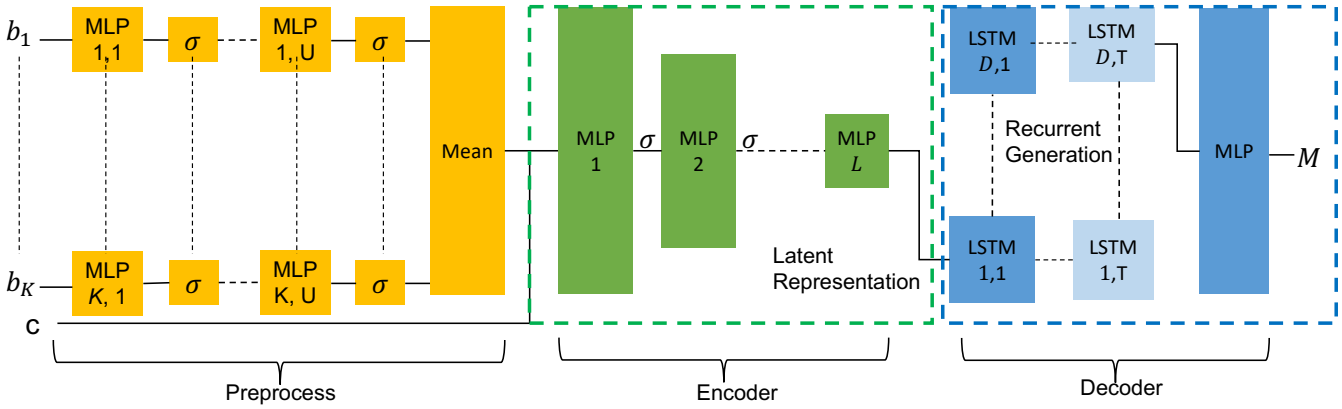


Fig. 2: The generator network with three channels, a pre-processor, an encoder and a decoder, each shown in a different colour. σ are arbitrary activation functions. The stacked LSTM is shown in an unrolled fashion where sequence dimension is shown horizontally and layer dimension is shown vertically. L denotes the number of MLP layers in the encoder, D represents the number of layers in the LSTM decoder and T denotes the sequence length.

uses an RNN to infer a suitable set of candidate macro actions, denoted as M. Details of this generator is presented in the next subsection.

The generated set of candidate macro-actions M will then be evaluated and progressively improved. RMAG can use any methods of evaluation and improvements of the set of candidate macro-actions. However, in this paper, the evaluation and improvements of the set of candidate policies in RMAG follow those of MAGIC [1], which relies on a critic network and a POMDP planner (the overall structure of RMAG is illustrated in Fig. 1). A POMDP planner that searches with macro action, instead of primitive actions, is utilised to find the best macro action to take from M according to the modified Bellman update equation:

$$V^*(b) = \max_{m \in M} \left\{ \mathcal{R}(b,m) + \gamma^L \sum_{\omega \in \Omega} P(\omega|b,m) V^*(b') \right\} \tag{4}$$

where $L = |M|$ is the size of the macro action set; $P(\omega \mid b, m) = \prod_{i=1}^{L} p(o_i \mid b_{i-1}, a_i)$ is the probability of observing a macro observations (a sequence of observations in which each observation is generated by the corresponding primitive action) conditioned on the belief and macro actions; and $\mathcal{R}(b,m) = \sum_{i=1}^{L} \gamma^{i-1} R(b_{i-1}, a_i)$ is the cumulative reward after executing a macro action $m \in M$. A critic model $\hat{V}_\psi$, parameterised by ψ, is used to predict the planner's value and maximise the following objective:

$$J(\psi) = \mathbb{E}_{\xi, M, v \sim D} \left[\log p_\psi(v) \right] \tag{5}$$

Here $p_\psi(v)$ is the probability density function of $\hat{V}_\psi$. The environment information ξ, the macro action set M and the planner value v are sampled from a replay buffer. And the objective of RMAG (denoted as G_θ), parameterised by θ, is to generate a set of macro actions that maximises the objective:

$$J(\theta) = \mathbb{E}_{\xi \sim D} \left[\mathbb{E}_M [\mathbb{E}[\hat{V}_\psi(\xi, M)]] + \alpha \mathcal{H}(G_\theta(\xi)) \right] \tag{6}$$

Where $\alpha \mathcal{H}(G_\theta(b,c))$ is the entropy regularisation term for controlling exploration by adjusting the positive scaler α according to [31]. Note that Equation 6 is made differentiable via the re-parameterisation trick for training [1].

B. The Generator Architectures

Let us now discuss the architecture of RMAG's candidate macro-actions generator. This generator consists of three neural network components that are trained end-to-end. It is illustrated in Fig. 2 and can be described as:

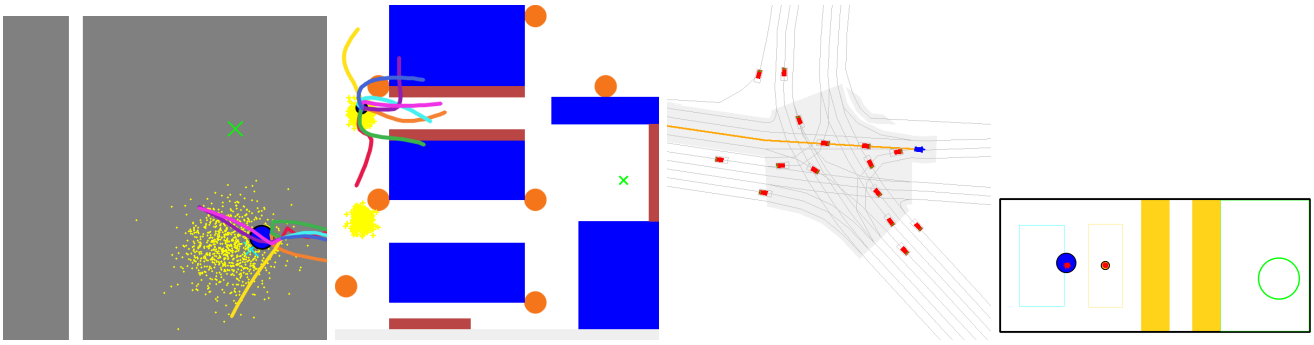
$$G_\theta(b,c) = decode \circ encode \circ preprocess(b,c) \tag{7}$$

where $preprocess$, $encode$, and $decode$ are the three neural network components and described below.

Similar to MAGIC [1], RMAG first performs a pre-processing of the environment information. RMAG architecture for this particular pre-processing component is similar to that of MAGIC, in the sense that RMAG projects ξ that contains the belief particles $[b_1, \ldots, b_k]$ and environment context c to a hyperspace $\hat{\xi} \in \mathbb{R}^W$ via the function $preprocess : \mathbb{R}^{dim(\mathcal{S})} \times \mathbb{R}^C \mapsto \mathbb{R}^W$, where $dim(\mathcal{S})$ is the dimension of the state space $\mathcal{S}$ (same dimension as a belief particle), C is the dimension of the context vector, and $W > dim(\mathcal{S})$ is the desired hyperspace dimension. The function itself is defined as:

$$preprocess(b,c) = \frac{1}{K} \sum_{i=1}^{K} \sigma(\text{MLP}_{iU} \ldots (\sigma(\text{MLP}_{i1}(b_i)))) \oplus c \tag{8}$$

where c is the context vector, K is the number of particles representing belief b and u is the number of layers of MLPs as displayed in Figure 2. Intuitively, this function takes the mean of the belief particles and concatenate them with the environment context vector, to provide an aggregate information about important features of the environment that resides in a high dimension such that a latent representation can be extracted in the next stage.



(a) Light Dark (b) Navigation 2D (c) Drive Hard (d) Puck Push

Fig. 3: Visualisation of the four experiments. (a) The Light Dark Environment: blue circle represents belief particles; coloured curves are macro actions produced by RMAG; light is represented by the white strip and crosses are initial and goal positions. (b) The Navigation2D Environment: the agent is represented as a blue circle; coloured curves are current macro actions; areas covered by beacons are represented by orange circles; blue rectangles are walls and red areas are danger zones. (c) The Drive Hard Environment: the ego-agent is coloured in blue and exo-vechiles are coloured in red; the orange path is the intended path followed by the agent. (d) The Puck Push Environment: The puck is represented as the red circle; covered zones where no observation can happen are shown as yellow strips; and goal region is shown as the green circle.

The second component, which is an encoder, $encode : \mathbb{R}^W \mapsto \mathbb{R}^Z$, learns the stochastic model $P(h \mid \hat{\xi})$ and outputs samples of the latent representation of the environment instance $h \in \mathbb{R}^Z$, where $Z < W$. The encoder is formed by a stacked of MLPs, in which the output dimension of each layer decreases monotonically and the output dimension of the last layer is Z, i.e.,

$$encode(\hat{\xi}) = \sigma(\text{MLP}_L(\ldots \sigma(\text{MLP}_1(x)))) \tag{9}$$

l is the the number of layers in the encoder network, and σ denotes arbitrary activation functions. With a compact latent representation, we can invoke the recurrence module next to generate macro actions.

The third component of the generator is a recurrent decoder. Once a latent environment representation is obtained, RMAG uses a stacked LSTM followed by a MLP to learn and sequentially produce a suitable set of candidate macro actions. The latent problem instance h is passed through multiple layers of LSTMs, where each LSTM aims to build progressively more complex macro action set conditioned on previous layer outputs. One can view this stage as a decoder modelled as $P(M \mid h)$. Concretely, we formulate this as, $decode(x) : \mathbb{R}^Z \mapsto \mathbb{R}^{dim(\mathcal{J})}$, where $dim(\mathcal{J})$ is the dimension of the parameter space of the macro action set M.

$$decode(x) = \text{MLP}(\text{LSTM}_D \circ \cdots \circ \text{LSTM}_1(x)) \tag{10}$$

where the final LSTM is post-processed by MLPs to produce the set of candidate macro-actions for planning, M.

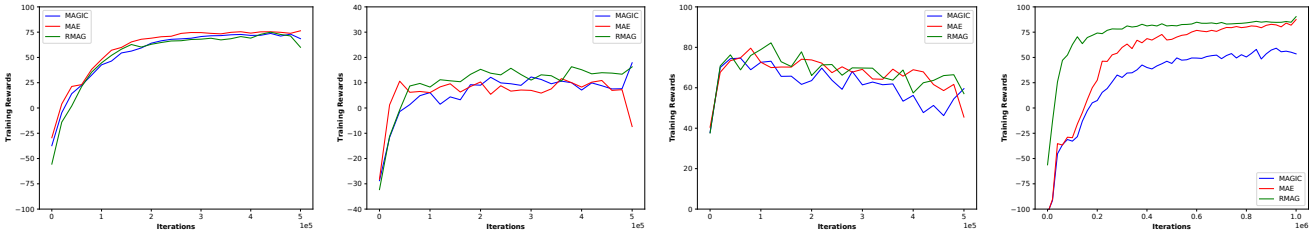
IV. Experiments

We perform experiments in simulation to understand the performance of RMAG's proposed candidate macro-actions generator and the effects of LSTM's depth to the performance. For this purpose, we compare RMAG with four different methods. First is POMCPOW [32] that computes the policy using primitive one-step actions (i.e. maco action size equals to 1). The second comparator is Handcrafted that uses predefined macro actions to search. The third comparator is MAGIC [1] that uses macro actions learned by a residual network to plan. Since the results of MAGIC published in their paper are different from the results of the published code ran on our machine, we include both results for fair comparisons. The last comparator is a base-line we designed to understand the effect of LSTM channels. We call this comparator Macro Action Encoder (MAE), which is similar to RMAG except that the candidate macro actions set is inferred straight away from the encoded environment via a single perceptron layer without passing through the recurrent stage. In other words, MAE performs the same preprocess and encoding stage as RMAG (see Figure 2) but the decoding stage contains only the MLP layer without any recurrence.

A. Scenarios used in Experiments

RMAG and each of the above comparators are tested on four environments with different complexities are used to test the generator. Three of these environments, *Light Dark*, *Drive Hard* and *Puck Push* come from the bench marking problems used in [1]. We further design another problem, *Navigation 2D*, that has larger map sizes and more complex uncertainties and dynamics to fully exploit our method's capability.

1) The Light Dark Environment: The first environment, see Figure 3a, simulates the problem of navigating a holonomic robot in a dark room. The room is 8 by 8 units wide and each discretised action is 0.5 units. The agent randomly starts at a position that is constrained to be far away from the light. The exact position is unknown to the agent and instead an initial belief is given according to the POMDP framework. Each primitive action has a cost of -0.1. The agent receives no observations unless it reaches the light position, in which



(a) Light Dark (b) Navigation 2D (c) Drive Hard (d) Puck Push

Fig. 4: Training plots of different methods over all iterations.

TABLE I: Performance comparisons. The MAGIC (git) row refers to the results running on our machine. RMAG LSTM stack size is fixed to be 2 for a fair comparison. Action length refers to the number of actions in a macro action.

	Light Dark Environment				**Navigation 2D Environment**			
Methods	Action Length	Steps Taken	Cumulative Reward	Success Rate %	Action Length	Steps Taken	Cumulative Reward	Collision Rate %
POMCPOW	1	22.1 (0.4)	-90.7 (0.5)	6.2 (0.3)	1	146.6 (0.55)	-28.1 (1.1)	4.3 (0.7)
Handcrafted	8	44.7 (0.1)	-30.9 (1.0)	37.1 (0.5)	16	134.8 (0.43)	2.81 (1.1)	12.6 (0.52)
MAGIC	8	37.6 (0.1)	54.1 (1.0)	79.0 (0.5)	16	129.2 (0.93)	5.70 (2.3)	15.4 (1.1)
MAGIC (git)	8	37.7 (0.26)	67.0 (2.2)	85.4 (1.1)	N/A	N/A	N/A	N/A
MAE	8	35.3 (0.19)	80.9 (1.7)	92.2 (0.85)	16	129.1 (0.9)	7.4 (2.3)	15.1 (1.2)
RMAG	8	35.7 (0.19)	78.3 (1.8)	90.9 (0.91)	16	128.6 (0.9)	16.0 (2.2)	10.9 (0.93)
	Drive Hard Environment				**Puck Push Environment**			
Methods	Action Length	Distance Covered	Cumulative Reward	Stall Rate %	Action Length	Steps Taken	Cumulative Reward	Success Rate %
POMCPOW	1	70.4 (1.0)	-44.7 (1.0)	13.0 (0.001)	1	31.0 (1.7)	-94.1 (1.0)	7.6 (0.5)
Handcrafted	3	98.1 (1.0)	13.1 (2.0)	9.9 (0.001)	5	42.1 (0.6)	34.0 (2.0)	70.0 (1.0)
MAGIC	3	111.0 (1.0)	58.6 (2.0)	6.9 (0.001)	5	35.3 (0.1)	87.9 (1.0)	95.3 (0.5)
MAGIC (git)	3	119.2 (1.4)	64.8 (2.4)	6.2 (0.0007)	5	40.4 (0.68)	73.1 (2.0)	88.9 (0.6)
MAE	3	113.5 (1.4)	59.3 (2.6)	6.2 (0.001)	5	129.1 (0.9)	7.43 (2.3)	15.4 (0.011)
RMAG	3	113.8 (1.4)	65.6 (2.4)	5.8 (0.0006)	5	30.9 (0.37)	92.5 (0.95)	97.8 (0.0046)

case it can fully observe its position. The objective is to stop precisely at a randomly selected goal position in the map, in which a reward of 100 will be given. However, the maximum step size is 60 and the episode terminates with cost of -100 if the agent has not reached the goal position after 60 steps. The discount factor is set to $\gamma = 0.98$. Clearly, a wise path would be to navigate to the light to localise and navigate towards the goal position afterwards. Hence, solving the problem optimally requires long horizon reasoning.

The actions are represented by 2D Bezier curves and each macro action is set to be of size 8. A total of 100 belief particles encoding the possible locations of the agent are drawn from the belief distribution. These belief particles along with a context vector that stores the goal and light positions are given to the learning model. During training or testing, the light position is uniformly drawn from the x-axis and its width doesn't change. The starting and end positions are drawn uniformly while being far away from the light.

2) The Navigation2D Environment: The Navigation2D environment (see Figure 3b) mimics the problem used in [7]. The map size is increased from 8 by 8 units to 60 by 60 units with each action takes 1 unit in length. There are two initial starting positions on the far left of the map, but the agent does not know where it starts and hence its initial belief scatters around two initial positions (this breaks the Gaussian property of beliefs in other scenarios). The goal position is restricted to be on the right half of the map where walls and danger zones are placed in between the initial position and the goal positions. The environment requires the agent to learn to take advantages of surrounding objects (walls and beacons) to localise and robustly reach the goal. There is no penalty given to the agent when bumping into the wall, however, upon entering the danger zone, the episodes terminates immediately and the agent receives a -100 penalty. Beacons are placed randomly on the map and the agent can completely localise itself when positioned within a certain distance from a beacon ; if the agent is farther than this distance, the agent receives no observation. The step penalty is set to -0.2, the goal reward is 100, the discount factor is set to 0.99. Since the map size is larger than that of Light Dark, the maximum step size is increased to 150 and similar to the previous problem, a penalty of -100 will be given to the agent if it does not completes the problem within the given number of step sizes.

Each action is represented by 2D Bezier curve and each macro action is set to be of size 16 to reflect the increase in the map size comparing to Light Dark. The context vector encodes the wall positions, light positions and goal positions. The walls, danger zones and two starting initial positions are fixed during training or testing. 7 beacons are randomly chosen from a pool of 20 beacons without replacement (a total of 77520 possibilities). And the goal positions are randomly drawn to be at the far right of the map.

3) The Drive Hard Environment: In this environment (see Figure 3c), the agent needs to control a non-holonomic vehicle to cross heavy traffic intersections while following a pre-determined path. The map size is irrelevant to the planning horizon of the problem as the agent's speed is controlled by accelerations. A total of 150 steps are allowed to execute in one episode and a penalty of -100 is given if the ego-vehicle collides with any exo-vehicles. At each step, a scaled penalty that depends on the current speed of the ego-vehicle is given and the agent receives a reward that is equal to the amount of path followed so far. The agent is able to observe the pose and velocity of all vehicles but not their intentions. It can also observes it own target path. The intentions of exo-vehicles are modelled by [33].

Due to motion noises and changing traffic conditions, a macro action size of 3 is used in this environment. A list of speed and steer commands (i.e. turn-and-go curves) are used to represent macro actions in this environment. The maximum speed, steer and acceleration of the target vehicle are constrained to be $6m/s$, 15° and $3m/s^2$, and that of other exo-vehicles are constrained to be $4m/s$, 15° and $2m/s^2$. During training or testing, all vehicles are spawned randomly on the map with their own target paths to traverse across an intersection.

4) The Puck Push Environment: In this environment (see Figure 3d), a holonomic circular robot is controlled by the agent to push a round puck to a desired location. The motion of the robot cause sliding motion of the puck in the 2D workspace. The sliding motion is modelled by $\theta' = \theta e^{\mu d}$ where θ is the direction of the puck in robot's frame, d is the distance robot moved and μ is the sliding rate coefficient. Part of the workspace is covered up such that the agent receives no information about any objects that's within the covers. When outside of the cover, the agent receives noisy observations with 90% chance, and no observations otherwise. Each step costs -0.1 penalty and a reward of 100 is given when the robot successfully delivers the puck and -100 penalty is given when the robot exhausts allowed step sizes (100) in an episode. The key challenge is to navigate around the puck to re-push it when the puck slides away from the agent.

A macro action size of 5 is chosen for this environment with each action parameterised by 2D Bezier curves. For each episode, the goal region is drawn randomly at the far right of the map and other object's positions are fixed as shown in the Figure.

B. Experimental Setup

All learning methods are implemented using Python, while the planner and simulator are implemented using C++. For fair comparisons, we choose the same critic architecture and planner (MACRO-DESPOT) used in [1]. We also use the same number of LSTM stack sizes (i.e. 2) in RMAG for all environments. The impact of different stack sizes is further studied in Section IV-D. Online learning is adopted where the collected experiences is stored in a replay buffer. 16 sub-processes running in parallel on an Intel-i7 are used to simulate and collect experiences. To facilitate the learning speed, the replay buffer is stored on the graphics card with a maximum size of 100,000 samples. When the max size is reached, the new sample will be used to replace the oldest one. All trainings are performed on a single NVIDIA 4080. Upon training, 256 samples will be drawn from the replay buffer uniformly and these samples form a single batch to train the networks. To ensure there is a rich data set in the beginning, 10,000 random simulations are collected with randomly generated macro actions. For all the environments, the initial learning rate is set to $1e-4$ and Adam optimiser is used. At test time, the critic is discarded and 500,000 random simulations are ran to obtain benchmark results. Since our domains are continuous and objects are drawn uniformly, we expect most of scenarios are novel to the agent during evaluation.

C. Experimental Results

The learning plots of the different methods and performance comparisons of these methods on the four testing scenarios are presented in Fig. 4. We noticed sometimes RMAG is unstable during training. The issue mainly come from the addition of recurrences in the model [29], [34]. Once trained, the mean (and standard errors) of the evaluation results are presented in Table I. In all four scenarios, RMAG substantially outperforms other methods, in terms of expected total reward and other qualitative measures, such as success rate.

1) The Light Dark Environment: Figure 4a displays the training results of the three learning methods. Since this task is the simplest in terms of environment complexity among the four scenarios, the training plots are similar to each other. However, RMAG generalise better during evaluation as shown in Table I. Our method performs at least 15% better than other methods in terms of cumulative rewards and at least 7% better in terms of success rate.

2) The Navigation 2D Environment: The high fluctuation of rewards in training is likely due to the complex dynamics of the environment. Based on Table I, MAGIC and MAE are able to reach the goal when the beacons are uniformly scattered across the map, however, these two methods are often stuck when the nearest beacon is far away from the initial position or there is a large gap between the beacons. RMAG, on the other hand, can easily navigate under more uncertain situations and precisely use nearby beacons to localise itself before finding its way to the target. Hence, it performs roughly three times better than MAGIC while minimising the chance of entering danger zones.

3) The Drive Hard Environment: Three methods exhibit similar learning plots as shown in Figure 4c. Since the macro action size is small, compared to other environments and potential errors can be constantly checked at each layer of search, adding memory module in the model do not significantly impact the quality of macro actions, therefore, RMAG only outperforms other methods by a small portion as shown in Table I. Nevertheless, this demonstrates that

TABLE II: Ablation study on the number of LSTM layers of RMAG for each environment

	Light Dark		Navigation 2D		Drive Hard		Puck Push	
# LSTM Layers (D)	Cumulative Rewards	Success Rate %	Cumulative Rewards	Success Rate %	Cumulative Rewards	Success Rate %	Cumulative Rewards	Success Rate %
0 (MAE)	80.9 (1.7)	92.2 (0.85)	7.42 (2.3)	83.4 (1.2)	59.34 (2.6)	78.1 (1.3)	92.2 (0.97)	97.7 (0.47)
2	78.3 (1.8)	90.1 (0.91)	16.0 (2.2)	89.9 (0.99)	65.6 (2.4)	82.6 (1.3)	92.5 (0.95)	97.8 (0.46)
4	77.6 (1.9)	90.6 (0.92)	16.0 (2.2)	88.9 (0.99)	61.0 (2.4)	80.6 (1.3)	51.9 (2.6)	78.9 (1.3)
6	83.8 (1.5)	93.7 (0.77)	18.1 (2.3)	86.8 (1.1)	61.8 (2.7)	76.7 (1.2)	75.9 (1.9)	90.3 (1.00)
8	58.1 (2.5)	81.2 (1.2)	21.6 (2.1)	90.4 (0.93)	41.2 (3.1)	64.5 (1.5)	58.7 (2.5)	82.3 (1.2)

RMAG has a robust learning capability even in situations that do not heavily require long-short term memory.

4) The Puck Push Environment: Since the puck gets stuck when it is near the wall and its position is unknown when hidden under the cover, the agent needs to plan its action based on previous sequence of actions and considers interacting with the puck carefully such that the puck can be delivered easily in future steps. Therefore, an LSTM in RMAG greatly benefits the learning of the agent as seen in Figure 4d. RMAG not only learns faster than other methods, but also reaches more rewards during evaluation. As a result, it performs superior than other methods with a 97.8% success rate.

D. Ablation Study

The depth of the LSTM channel in RMAG influence its performance in different environments. Table II displays the benchmark results of RMAG with different LSTM depths for each environment while keeping other parameters the same. Note that 0 stack size refers to the MAE architecture. For Light Dark and Puck Push, the MAE structure performed nearly as good as the best results, which shows the importance of building a latent representation for the inference of macro actions. However, MAE did not outperform RMAG in any of the environments, as the recurrence module is able to progressively build more quality macro action sets in a given situation. Furthermore, greater stack size works better in Light Dark and Navigation 2D environment as shown in Table II. This is linked to the fact that these environments use a bigger macro action length comparing to the Drive Hard and Puck Push environment (see Table I), which require greater model complexities.

We further list architecture complexities in terms of number of parameters, parameter memory size and multiply accumulates (MACs) conditioned on a fixed input size in Table III. RMAG with stack size less than 6 has fewer parameters and consume less memory and computation resources comparing to MAGIC. Since MAE is a simplified version of RMAG, it hence enjoys the lightest architecture.

TABLE III: Model Complexities Conditioned on Fixed Input Sizes

	Input Dimension	**# of Param**	**Memory (MB)**	**MACs (M)**
MAGIC	(1x3, 1x300)	1,804,600	7.22	1.80
MAE	(1x3, 1x300)	371,192	1.48	0.37
RMAG-2	(1x3, 1x300)	387,032	1.55	0.91
RMAG-4	(1x3, 1x300)	403,928	1.62	1.45
RMAG-6	(1x3, 1x300)	420,824	1.68	1.99
RMAG-8	(1x3, 1x300)	437,720	1.75	2.53

V. Conclusions

In this paper, we propose a candidate macro-actions generator that can exploit environment information, based on RNN. This architecture is then combined with the critic and planner components of MAGIC to provide an progressive improvement of the set of macro-actions for POMDP planning. Comparison with state-of-the-art, including MAGIC, on four different long horizon planning tasks with various difficulties indicate that the proposed architecture outperforms existing method and architecture in solving problems with long planning horizon. It also enjoys a lighter computational complexity while maintaining a high generality comparing to the generator used in MAGIC. Further understanding on how environment information can be exploited to construct simple learning components that help improve planning scalability could be very useful.

References

[1] Y. Lee, P. Cai, and D. Hsu, "Magic: Learning macro-actions for online pomdp planning," *arXiv preprint arXiv:2011.03813*, 2020.

[2] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[3] H. Kurniawati, "Partially observable markov decision processes and robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 253–277, 2022.

[4] G. Theocharous and L. Kaelbling, "Approximate planning in pomdps with macro-actions," *Advances in Neural Information Processing Systems*, vol. 16, pp. 775–782, 2003.

[5] R. He, E. Brunskill, and N. Roy, "PUMA: Planning under uncertainty with macro-actions," in *AAAI*, 2010.

[6] Z. W. Lim, D. Hsu, and W. S. Lee, "Monte carlo value iteration with macro-actions." in *NIPS*, 2011, pp. 1287–1295.

[7] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *IJRR*, vol. 30, no. 3, pp. 308–323, 2011.

[8] A.-A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato, "Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements," *IJRR*, vol. 33, no. 2, pp. 268–304, 2014.

[9] G. Flaspohler, N. A. Roy, and J. W. Fisher III, "Belief-dependent macro-action discovery in pomdps using the value of information," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 108–11 118, 2020.

[10] A. Plaat, W. Kosters, and M. Preuss, "Model-based deep reinforcement learning for high-dimensional problems, a survey," *arXiv preprint arXiv:2008.05598*, 2020.

[11] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[12] Y. Zhu, A. Joshi, P. Stone, and Y. Zhu, "Viola: Imitation learning for vision-based manipulation with object proposal priors," *arXiv preprint arXiv:2210.11339*, 2022.

[13] J. Oh, S. Singh, and H. Lee, "Value prediction network," *Advances in neural information processing systems*, vol. 30, 2017.

[14] 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, pp. 484–489, 2016.

[15] X. Chen, H. Dai, Y. Li, X. Gao, and L. Song, "Learning to stop while learning to predict," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1520–1530.

[16] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[17] E. J. Sondik, "The optimal control of partially observable markov processes," Ph.D. dissertation, Stanford University, 1971.

[18] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," 2001.

[19] M. Stolle and D. Precup, "Learning options in reinforcement learning," in *International Symposium on abstraction, reformulation, and approximation*. Springer, 2002, pp. 212–223.

[20] S. Mannor, I. Menache, A. Hoze, and U. Klein, "Dynamic abstraction in reinforcement learning via clustering," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 71.

[21] T. A. Mann, S. Mannor, and D. Precup, "Approximate value iteration with temporally extended actions," *Journal of Artificial Intelligence Research*, vol. 53, pp. 375–438, 2015.

[22] G. Konidaris, "Constructing abstraction hierarchies using a skill-symbol loop," in *IJCAI: proceedings of the conference*, vol. 2016. NIH Public Access, 2016, p. 1648.

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

[24] P. Karkus, D. Hsu, and W. S. Lee, "Qmdp-net: Deep learning for planning under partial observability," *Advances in neural information processing systems*, vol. 30, 2017.

[25] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[26] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *NeurIPS*, vol. 31, 2018.

[27] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," *Advances in neural information processing systems*, vol. 28, 2015.

[28] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *ICRA*. IEEE, 2017, pp. 2786–2793.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[30] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[31] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[32] Z. Sunberg and M. Kochenderfer, "Online algorithms for pomdps with continuous state, action, and observation spaces," in *ICAPS*, vol. 28, 2018, pp. 259–263.

[33] Y. Luo, P. Cai, D. Hsu, and W. S. Lee, "Gamma: A general agent motion prediction model for autonomous driving," *arXiv preprint arXiv:1906.01566*, 2019.

[34] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.