Learning with Stochastic Guidance for Robot Navigation

Linhai Xie[1], Yishu Miao[1], Sen Wang[2], Phil Blunsom[1],
Zhihua Wang[1], Changhao Cheng[1], Andrew Markham[1] and Niki Trigoni[1]

***Abstract*—Due to the sparse rewards and high degree of environment variation, reinforcement learning approaches such as Deep Deterministic Policy Gradient (DDPG) are plagued by issues of high variance when applied in complex real world environments. We present a new framework for overcoming these issues by incorporating a stochastic switch, allowing an agent to choose between high and low variance policies. The stochastic switch can be jointly trained with the original DDPG in the same framework. In this paper, we demonstrate the power of the framework in a navigation task, where the robot can dynamically choose to learn through exploration, or to use the output of a heuristic controller as guidance. Instead of starting from completely random actions, the navigation capability of a robot can be quickly bootstrapped by several simple independent controllers. The experimental results show that with the aid of stochastic guidance we are able to effectively and efficiently train DDPG navigation policies and achieve significantly better performance than state-of-the-art baseline models.**

***Index Terms*—Deep reinforcement learning(DRL), Deep deterministic policy gradient(DDPG), REINFORCE, Robot navigation.**

I. Introduction

DEEP Reinforcement Learning (DRL) has been shown to be highly effective at mastering complex simulations and artificial tasks, e.g. playing Atari games [1] and Go [2]. However, DRL's poor sample complexity has limited its application to real world tasks, such as navigating a robot to a target position without crashing into obstacles.

Deep Deterministic Policy Gradient (DDPG) [3] is an actor-critic algorithm that is suitable for such continuous control tasks in principle, but in practice the cost of exploration in complex navigation environments can prove prohibitive.

Since an agent must stochastically explore a long sequence of states during each training episode, high variance becomes the main bottleneck that hinders DDPG from learning effective DRL models. In order to mitigate this issue, conventional architectures generally require a huge number of learning samples, resulting in high computational and environmental costs. In this paper, we propose a new framework that allows an agent to stochastically switch between high variance controllers (e.g. DDPG), and low variance controllers (e.g. simple deterministic controllers), effectively allowing the DDPG component to be quickly bootstrapped instead of starting from completely random moves.

[1] Department of Computer Science, University of Oxford, Oxford OX1 3QD, United Kingdom {firstname.lastname} @cs.ox.ac.uk

[2] School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh EH14 4AS, United Kingdom s.wang@hw.ac.uk

Manuscript received April 19, 2005; revised August 26, 2015.

Intuitively, learning is usually easier to carry out under guidance from other heuristics. The independent controllers here act as the guidance that are introduced for learning better DDPG policies. In our case, the agent still maintains an independent DDPG module that learns navigation by exploring the environment, but is able to dynamically switch between learning from exploration or learning from the heuristic controllers. Here, the switching mechanism is constructed as a stochastic function updated by the REINFORCE learning signal [4] to maximise total reward. Meanwhile, the DDPG component is learned by employing the action selected by the stochastic switch, rather than directly using the output action generated by its policy network. Therefore, the switching mechanism helps DDPG avoid trivial explorations during the early training process, and learns to balance between exploration and heuristic guidance. More interestingly, once trained, the DDPG component can be used in isolation from the other controllers, in which case the switch is turned off and the navigation is carried out solely by the DDPG component. Similar to the idea of imitation learning [5], [6], the DDPG component is able to learn from the demonstrations given by the guidance, which is a Proportional-integral-derivative (PID) controller and an obstacle avoidance (OA) controller in our case, and instantly generalise to new situations that PID and OA could not handle. Those guidance can be considered as a positive bias for reducing the variance of gradient estimators, and the model is able to remove this bias after benefiting from it.

For quantitative evaluation, we firstly compare our model with the incorporation of the stochastic switch to the vanilla DDPG baseline and deterministic benchmarks for demonstrating the benefits brought by bootstrapping with additional primitive controllers. Then the influence of using different independent controllers is investigated, which shows that the framework has strong generalisation ability and it is able to accumulate the benefits from different simple controllers. In addition, we propose three variants of the switch mechanism including a uniformly random switch, an argmax switch and a Thompson sampling switch for comparison. Finally, we show that the models can abandon the extra controllers when their usage rate declines below a threshold and are able to continue self-learning by only using the DDPG component. For qualitative evaluation, we test our model in a real world scenario: without further modification, the model trained in simulation is able to be directly transferred to carry out navigation tasks.

In summary, we propose a new framework that leverages

the heuristic knowledge provided by independent controllers to bootstrap deep reinforcement learning for robot navigation. Our experiments demonstrate that by incorporating stochastic guidance, we are able to effectively and efficiently train the DDPG navigation policies and achieve significantly better performance than baseline models.

Specifically our contributions are:

- We achieve near-oracle performance, with low training variance and an accelerated training rate relative to vanilla DDPG.
- We demonstrate how the stochastic switch dynamically learns which controller to choose based on the input and anticipated reward, outperforming switches which only learn from anticipated reward (e.g. Thomson sampling).
- We show that this technique is particularly important for sparse rewards, where vanilla DDPG is unable to learn how to complete an entire episode.

II. Problem Formulation and Assumptions

We can consider the local navigation problem as a decision making process [7] where the robot is required to avoid obstacles and reach a target position. At time $t \in [0, T]$ the robot takes an action $a_t \in \mathcal{A}$ according to the observation x_t. After executing the action, the robot receives a reward r_t given by the environment according to the reward function and then transits to the next observation x_{t+1}. The goal of this decision making procedure is to reach a maximum discounted accumulative future reward $R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau$, where γ is the discount factor.

The reward function r_t at time t is defined as:

$$r_t = \begin{cases} R_{crash}, & \text{if robot crashes} \\ R_{reach}, & \text{if robot reaches the goal} \\ (d_{t-1} - d_t)\cos(\omega_t) - C, & \text{otherwise} \end{cases} \tag{1}$$

where R_{crash} is a large penalty for collision, R_{reach} is a positive reward for reaching the goal, d_{t-1} and d_t denote the distances between the robot and the goal at two consecutive time steps $t-1$ and t, ω_t represents the rotational speed of the robot at time t, and C is a constant time penalty which encourages the robot to approach the goal quickly.

III. Model

The proposed model consists of three parts: perception, control and stochastic switch, as shown in Fig.1. At each time step, the perception part processes an observation and generates a corresponding input representation. Then different controllers can propose candidate actions based on the input representation. Finally, the stochastic switch determines which one of the actions to be carried out.

A. Perception

At each time step t, the robot observes the state of the world x_t, which includes a stack of current and historical geometric observations, its linear and angular velocities and a destination. The geometric observations, which give distances to surrounding objects (depth images or laser scans), are processed by a convolutional neural network to produce a compressed input representation. It is then concatenated with the robot velocities and destination and input to both the control and stochastic switch blocks. We assume that the robot can localise itself to obtain the relative position of the destination in robot coordinate frame, for example using wi-fi localisation [8].

B. Control

1) Action: With the observation x_t at time t, the robot takes an action $a_t = (a_t^v, a_t^\omega) \in \mathcal{A}$, where a_t^v and a_t^ω respectively denotes the expected linear and rotational velocity at time t. It obtains a reward r_t given by the environment and transitions to the next observation x_{t+1}. The goal of our model is to reach a maximum discounted accumulative future reward R_t. In this work, the actions can be determined by the independent controllers and DDPG, establishing a set of candidate actions for the stochastic switch to choose.

2) Independent Controllers: Two independent controllers aid DDPG to learn reasonable policies, especially in the initial training phase. One is a proportional-integral-derivative (PID) controller with proportional term [9], which derives action from the relative position of the destination $[x_{local}, y_{local}]$ in robot coordinate frame as:

$$a_t = K_p \cdot [x_{local}, y_{local}]^T, \tag{2}$$

where K_p is the coefficient for the proportional term. PID controller is one of the most widely used and successful control mechanisms. However, without considering geometric observation, it does not have an obstacle avoidance capability.

The other one is a simple obstacle avoidance (OA) algorithm which can drive the robot without collision. It uses geometric observations to detect and avoid nearby obstacles by controlling the heading direction (rotational speed) of the robot:

$$|a^\omega| = \begin{cases} a_{max}^\omega \cdot \frac{|d_o - \beta|}{\beta}, & d_o < \beta \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

where d_o is the distance to the closest obstacle, a_{max}^ω represents the largest rotational speed and β indicates a pre-defined minimum safety distance. In the case where the distance between the robot and an object is less than the safety distance, i.e., $d_o < \beta$, the robot will rotate to avoid collision. These two controllers complement one another to provide candidate actions for stochastic switch. Note that the OA only produces a^ω, while the selected a^v is provided by the DDPG controller.

Note that these are just two simple exemplar controllers, and it is possible to incorporate more either in number or sophistication.

3) DDPG: The main controller of this framework is DDPG, which is an actor-critic approach in DRL [3] that simultaneously learns the policy and the action-state value (Q-value) to assess the learnt policy. Although the policy network and the critic network of the DDPG share the same input representation from the perception, the policy network predicts the action, while the critic network estimates the Q-value for current state-action pair. In the learning mechanism of critic network, given the policy π which maps states to actions a_t, the expected return

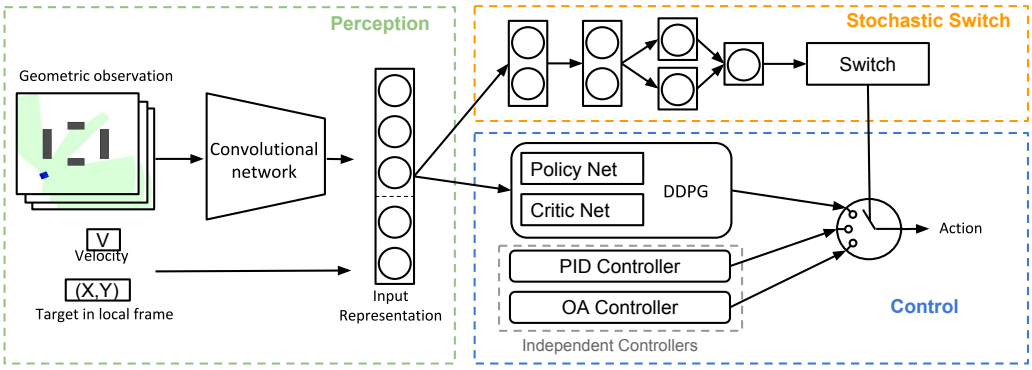


Fig. 1: The architecture of the proposed framework. It consists of three sections namely: perception, control and stochastic switch. The perception section processes an observation and generates a corresponding input representation. The control section contains a standard DDPG controller and a number of independent controllers - in this case, a Proportional-integral–derivative (PID) controller and an Obstacle Avoidance (OA) controller. The stochastic switch determines which controllers' action should be used out for navigation.

is $Q^\pi(x_t, a_t) = \mathbb{E}[R_t|x_t, a_t, \pi]$, which can be calculated with the Bellman equation [10]:

$$Q^\pi(x_t, a_t) = \mathbb{E}[r_t + \gamma \mathbb{E}[Q^\pi(x_{t+1}, a_{t+1})|x_t, a_t, \pi]].$$

If the policy is deterministic, we can define $a_t = \mu(x_t)$ and the inner expectation can be avoided. Since the outer expectation is independent of policy μ, it becomes an off-policy learning. Then, the objective is to minimise the temporal difference (TD) error:

$$\begin{aligned} L(\theta^Q) &= \mathbb{E}_{x_t, a_t, r_t, x_{t+1}}[(y - Q(x_t, a_t; \theta^Q))^2], \\ y &= r_t + \gamma Q(x_{t+1}, a_{t+1}; \theta^Q) \end{aligned} \tag{4}$$

where θ^Q is a parameter of the critic network. To update the critic network by temporal difference learning [11], all learning samples stored in the replay buffer are formulated as (x_t, a_t, r_t, x_{t+1}).

The policy network is parameterised by θ^μ. During training, the gradients are estimated by applying chain rules to the objective function (expected reward) $J(\theta^\mu)$ w.r.t the parameters θ^μ. Generally, in DDPG, the parameters are updated by the gradients computed based on the actions produced by the policy network. However, in our case, we introduce a stochastic switch for choosing the final action from a set of actions proposed by all controllers. Hence, the networks are updated by the action finally selected by the switch network at each time step. It can be the a_t produced by the policy network of DDPG as well as the actions from the heuristic controllers, depending on different situations and the learnt switching strategy.

C. Stochastic Switch

The PID controller, OA algorithm and DDPG are three independent sources that produce candidate actions for the switch network to (optimally) select. The switch network is a stochastic deep neural network which consists of a parameterisation network and a multinomial distribution. The switch effectively learns which controller to choose, based on the anticipated reward and conditioned on the input representation. It is trained end-to-end with the other deep learning components e.g. DDPG and the optional input representation network.

1) Stick-breaking: Conventionally, a softmax layer can be employed to provide the parameter θ for the multinomial distribution. Here, instead, we apply **stick-breaking construction** [12], [13], [14], which is alternative to softmax.

The intuition is to introduce a bias that encourages more usage of the deep reinforcement learning algorithm, such as DDPG in our case. Since our framework is designed to train a robust DDPG component that benefits from the stochastic guidance, we expect it to be used more often than others in this framework so that we are able to get rid of the simple independent controllers after a certain period of training. It basically transforms the modeling of multinomial probability parameters into the modeling of the logits of binomial probability parameters.

We firstly define $\alpha = f_s(x_t|\theta^s)$ as the unscaled logits from the fully connected layers of switching network $f_s(\cdot|\theta^s)$ given the input representation x_t and θ^s as the parameter of the switch network. Then the binomial logits η can be obtained with $\eta = \text{sigmoid}(\alpha)$. Notice that $\eta = [\eta_1, \eta_2]^T$ and $\alpha = [\alpha_1, \alpha_2]^T$ are all 2 dimensional vectors since we have 3 controllers in our case.

Therefore, the multinomial probability parameters $\xi = [\xi_1, \xi_2, \xi_3]$ can be generated with the stick breaking function $f_{\text{SB}}(\eta)$ by two breaks:

$$\begin{aligned} \xi_1 &= \eta_1 \\ \xi_2 &= \eta_2(1 - \eta_1) \\ \xi_3 &= (1 - \eta_2)(1 - \eta_1), \end{aligned} \tag{5}$$

where the f_{SB} can be generalised to more breaks $\xi_k = \eta_k \prod_{i=1}^{k-1}(1 - \eta_i)$ $(1 < k < K)$ if there are K controllers.

Conditioned on the current observation x_t, we are able to construct the stochastic switch policy $s_t \sim \pi^s(s_t|x_t; \theta^s)$ as:

$$\xi = f_{\text{SB}}(\text{sigmoid}(f_s(x_t|\theta^s))) \tag{6}$$

$$s_t \sim \text{multinomial}(\xi). \tag{7}$$

At each time step t, the stochastic switch samples a decision s_t and ξ_1, ξ_2, ξ_3 corresponds to DDPG, PID and OA. Then, according to the decision s_t, the critic network of DDPG takes the final action $a_{s_t} \in \{a_{\text{DDPG}}, a_{\text{PID}}, a_{\text{OA}}\}$ as input and updates the networks accordingly. Meanwhile, the stochastic switch is updated by the REINFORCE learning signal so that the

switch network is able to dynamically choose to learn through exploration (DDPG), or it can choose to use the output of a heuristic controller (PID or OA) as guidance by observing the environment.

2) REINFORCE Algorithm: Since the gradients cannot be directly back-propagated through the discrete samples, we employ REINFORCE algorithm [4] to construct the gradient estimator for the switch network, where the goal is to maximise the total reward R under the switching policy $\pi^s(s_t|x_t;\theta^s)$. Thus the objective function is:

$$\mathcal{H} = \mathbb{E}_{p(S;\theta^s)}[R] = \mathbb{E}_{p(S;\theta^s)}\Big[\sum_{\tau=1}^{T}\gamma^{\tau-1}r_\tau\Big], \tag{8}$$

where S is a sequence of decisions $s_1, s_2, ..., s_T$ in an episode, $s_t \sim \pi^s(s_t|x_t;\theta^s)$ is the decision sample at each time step t, and $p(S;\theta^s) = \prod_{t=1}^{T}\pi^s(s_t|x_t;\theta^s)$ is the probability of generating the current decision sequence. Hence, the gradients can be estimated as follows:

$$\begin{aligned}\frac{\partial\mathcal{H}}{\partial\theta^s} &= \mathbb{E}_{p(S;\theta^s)}\Big[\frac{\partial}{\partial\theta^s}\log p(S;\theta^s)R\Big] \\ &\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\frac{\partial}{\partial\theta^s}\log\pi^s(s_t^n|x_t^n;\theta^s)R^n\end{aligned} \tag{9}$$

where N is the number of sampled episodes, T_n is the length of the episode n, and R^n is the total reward of the episode. It indicates a Monte Carlo based unbiased gradient estimation for updating the switch network.

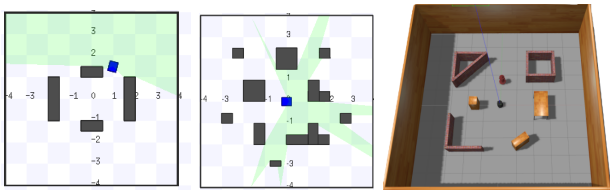
3) Variance Reduction: Since the REINFORCE gradient estimator also suffers from the high variance issue, we introduce two control variates [15] for alleviating the problem: a centred learning signal b^c (moving average) and an input dependent control variate $b(x)$ respectively. Here, we simply build an MLP (multilayer perceptron) to implement the $b(x)$ conditioned on input x. During training, the two control variates are learned by minimising the expectation: $\mathbb{E}_{p(S;\theta^s)}[R - b^c - b(x))]^2$, which is and the gradients are derived as,

$$\frac{\partial\mathcal{H}}{\partial\theta^s} \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_n}\frac{\partial}{\partial\theta^s}\log\pi^s(s_t^n|x_t^n;\theta^s)(R^n - b^c - b(x_t)).$$

The introduction of stochastic switch can be considered as an inductive bias for learning to navigate with better action samples. Updated by REINFORCE, the stochastic switch is able to sense the environment, avoid trivial explorations and select better actions for learning DDPG policies. In addition, as the independent controllers are incorporated via the stochastic switch, the negative influence of the introduced biases from the heuristics is limited. The independent controllers can be explicitly disabled once DDPG is sufficiently trained, although this naturally happens to a large degree anyway.

D. Algorithm

The brief algorithm of training DDPG with our stochastic guidance (SGuidance) is demonstrated in Algorithm 1. Since DDPG is an off-policy approach, a replay buffer R is applied to store all the transitions (x_t, a_t, r_t, x_{t+1}). And a batch of transition is sampled at every training step to update DDPG. In contrast, the learning of switching policy with REINFORCE is an on-policy process, thus we only save the current trajectory of switching (x_t, s_t, r_t), where $t = 1, 2, ...T$ to calculate the gradients for updating the switching network, including the control variate, at the end of each episode.



(a) *ROS Stage simple* (b) *ROS Stage complex* (c) *ROS Gazebo*

Fig. 2: **(a)** The 4 grey rectangles are obstacles and the blue square represents the robot. A sparse laser is mounted on the robot and its detecting area is illustrated as the green area. **(b)** It shows a more complex environment simulated by *ROS Stage*. **(c)** *ROS Gazebo* is also used for training a model that can be transferred to real-world environment. Turtlebot 2 (a platform for ground robots) is employed as the mobile platform equipped with a depth camera.

Algorithm 1 SGuidance

procedure TRAINING
  Initialize switching network $f_s(x_t|\theta^s)$ and buffer R_s.
  Initialize DDPG network $Q(x, a|\theta^Q)$ and $\pi(x|\theta^\pi)$.
  Initialize the target network of DDPG.
  Initialize replay buffer R and exploration noise ϵ.
  for episode=1, M **do**
    Reset the environment.
    Initialise replay buffer R_s.
    Obtain the initial observation x_t.
    for step = 1, T **do**
      Sample switch $s_t = f_s(x_t|\theta^s) \in \{0, 1, 2\}$.
      Sample $a_t^0 = \pi(x_t|\theta^\pi) + \epsilon$.
      Get a_t^1 from PID controller.
      Get a_t^2 from OA controller.
      Execute $a_t = a_t^{s_t}$ and obtain r_t, x_{t+1}.
      Store transition (x_t, a_t, r_t, x_{t+1}) in R.
      Sample a batch of transitions from R.
      Update the actor and critic network of DDPG.
      Update the target network of DDPG.
      Store switching trajectory (x_t, s_t, r_t) in R_s.
    end for
    Update switching network with trajectory in R_s.
  end for
end procedure

IV. EXPERIMENTS

A. Training Environments and Settings

The proposed framework is trained in two different simulators. The first one is a light-weight simulator, *ROS Stage*[1] (Fig. 2(a) and Fig. 2(b)), in which a large amount of repetitive experiments are conducted for showing the learning curve, demonstrating the improvements brought by stochastic

[1]http://wiki.ros.org/stage

guidance, and comparing to other baseline models. In this simulator, we mount the mobile robot with a laser scanner to provide the geometric information of surroundings. Hence, the convolutional neural network (in Fig. 1) is not used in this case, and the laser scans are directly concatenated with the other observations as input representation. By accelerating the simulation time, we obtain the **quantitative evaluation** through a lot of repetitive experiments in *ROS Stage*. Note that the simple environment as shown in Fig. 2(a) is our default training environment and experiments in the complicated environment are carried out in Sec. IV-B6.

The other one, *ROS Gazebo*[2] (Fig.2(c)), contains a physical engine and can accurately simulate the dynamics of the mobile robot. Thus the model trained in *ROS Gazebo* is directly applied to real world scenario to **qualitatively evaluate** the navigation performance, but it has a larger computational overhead compared to *ROS Stage*. Here, depth images are utilised to observe surroundings, therefore a 3-layer convolutional network (the filters are [4,4,3,8], [4,4,8,16] and [4,4,16,32] respectively) is constructed to provide input representations based on depth images.

In each training episode the robot starts at the origin point (centre point) with a random heading direction and the destination is randomised within the area beyond obstacles. When the robot collides with an obstacle or reaches the destination, the current episode terminates. The action control frequency is 5Hz and the switching frequency is 1Hz. The lower switching frequency is to facilitate the learning of the switching policy as long episodic samples result in higher variance of the estimated gradients by REINFORCE. For all the experiments carried out in *ROS Stage*, the training process lasts for 100k steps and is repeated for 5 times. The averaged learning curves as well as the variance[3] are illustrated for demonstrating the performance.

Regarding the hyper-parameters, the hidden layers of critic network and actor network contain 100 ReLU units in each layer, while the output layer of actor network applies tanh and sigmoid respectively for rotational and linear velocity. When updating DDPG parameters, 32 learning samples are randomly sampled from a rank based prioritised experience replay [16] as a training batch, and the learning rate for the actor network, the critic network and the stochastic switch are 10^{-4}, 10^{-3} and 10^{-3} respectively, and the rest follows [3].

B. Navigation in Simulated Environment

Here, we discuss the experiments carried out in *ROS Stage* for quantitative evaluation. The default settings for applying stochastic switch with DDPG is to learn with the guidance from both PID and OA controllers where REINFORCE algorithm and Control Variates(CV) are utilised.

1) Reinforcement Learning with Stochastic Guidance: Fig.3 compares the models for demonstrating the benefits brought by learning with stochastic switch. **SGuidance** is our model with stochastic switch that dynamically choose the action from the candidates proposed by the controllers of DDPG, PID and OA. As shown in Fig.3, **SGuidance** achieves significantly better performance than the **DDPG** baseline. Meanwhile, **DDPG** suffers from the high variance issue according to the wide transparent area around the learning curve, while **SGuidance** is much more stable. This is due to the high complexity of the environment that leads to the highly variant learning samples provided by DDPG, which might lead to trivial explorations. In addition, the stochastic gradient estimator of DDPG applies biased approximation which makes it difficult to guarantee the convergence and stability. By contrast, **SGuidance** is able to benefit from the heuristic simple controllers since the beginning of training procedure instead of starting from completely random moves.

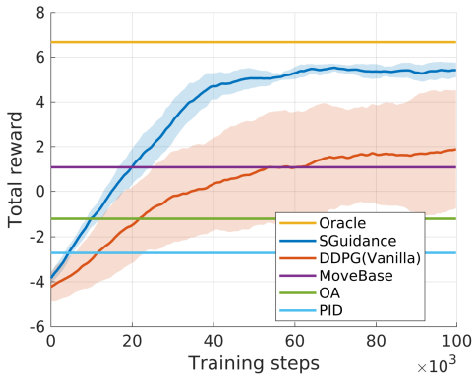


Fig. 3: The total reward achieved by our models and other baseline models as comparison. The curve represents the average value of 5 repetitive training procedures and the transparent area indicates the variance of the results.

In this experiment, we also plot the rewards of **MoveBase** (without map) and **Oracle** (**MoveBase** with map) for comparison. **MoveBase** package[4] is a widely used motion planner for mobile robot navigation and is implemented in the ROS package named Navigation Stage. It consists of a local planner [17], [18] and a global planner (implemented by Dijkstra or A* algorithm). The global planner generates an optimal path from the origin to the destination on the global map of the environment, and the local planner dynamically avoids the newly detected obstacles while moving along the optimal path. Hence, we call the **MoveBase** with map **Oracle** in this experiment. As shown in Fig.3, **DDPG** is able to obtain comparable performance to **MoveBase**. **SGuidance**, however, significantly surpasses the deterministic **MoveBase** model. Even without the access to the global map, **SGuidance** has shown its strong ability to navigate in the environment by just using the geometric information. In addition, we plot the performance of two simple heuristic controllers (OA and PID) for reference. Basically, the simple deterministic controllers cannot be applied independently for carrying out navigation task (the accumulative rewards are both under 0). However, when incorporated with DDPG via the stochastic switch, they contribute significantly towards alleviating the high variance issue encountered with vanilla DDPG.

2) Using Different Independent Controllers: This experiment shows the impact of different independent controllers

[2]http://wiki.ros.org/gazebo_ros_pkgs

[3]Note that the variance mentioned here is the variance of the smoothed learning curves.

[4]http://wiki.ros.org/move_base

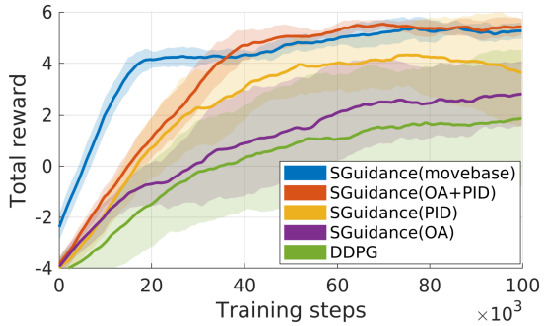


Fig. 4: The smoothed total reward obtained by incorporating different heuristic controllers with DDPG. **SGuidance(PID+OA)** utilises both PID and OA controllers while **SGuidance(OA)** and **SGuidance(PID)** only adopt one of them respectively. **SGuidance(movebase)** is guided by movebase and **DDPG(Vanilla)** is the baseline DDPG without heuristic guidance.

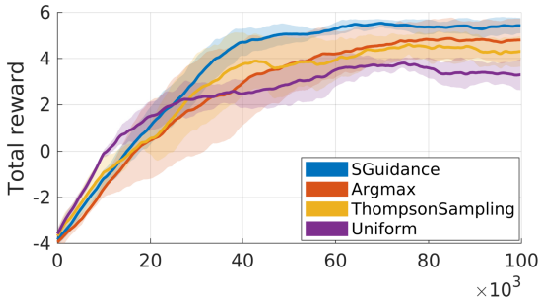
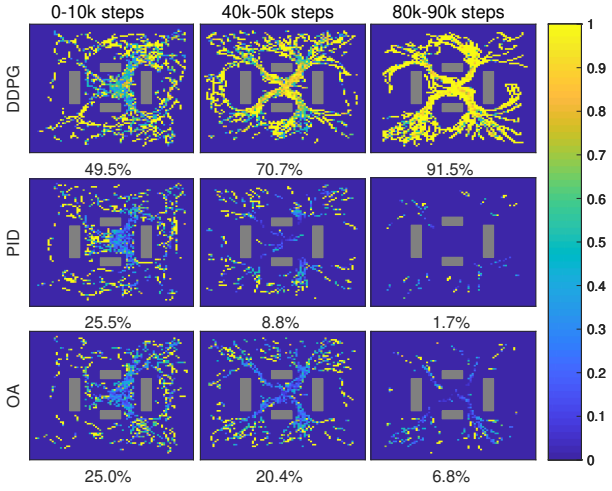


Fig. 5: The accumulated reward obtained by learning with different switching mechanism. **Stochastic** represents our default stochastic switch settings. **Argmax** and **Uniform** are the proposed argmax switch and uniformly distributed switch respectively. **ThompsonSampling** is the Thompson Sampling we implemented with Gaussian prior.

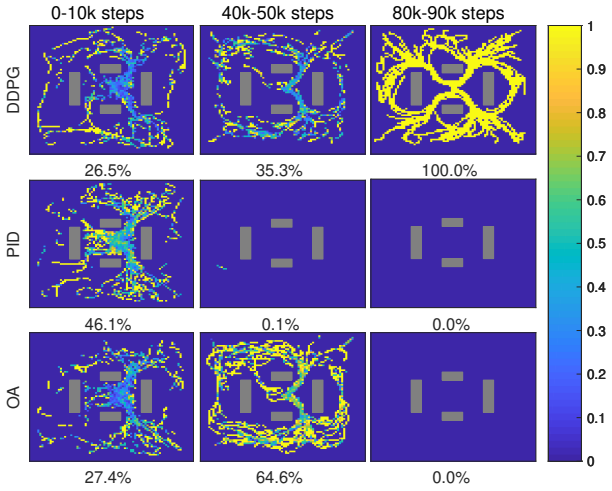
towards training performance.

As illustrated in Fig.4, **SGuidance (PID + OA)** achieves the best performance when compared to the DDPG with only PID or OA and the DDPG without any independent controllers. It demonstrates that the contribution of the stochastic switch is greatly enhanced by adding more controllers, which yields more stable learning curves and better navigation performance. Interestingly, the PID controller brings more benefits than the OA controller in this context, and their benefits could be accumulated with the help of the stochastic switch. Additionally, when movebase is utilised as the independent controller alongside DDPG, there is an obvious improvement on learning speed, especially in the early stage of training. These results show that the proposed framework enables us to incorporate a large range of different controllers, from naive to more sophisticated ones. This is a massive advantage over imitation learning which requires a good demonstration to learn from.

3) Using Different Switching Mechanism: Fig. 5 compares the stochastic switch to other switching variants, including argmax switch, a uniformly random switch and a switch based on Thompson Sampling [19]. The uniform switch assigns uniformed fixed probability to DDPG, PID and OA controllers, while the argmax switch applies biased argmax output instead of stochastically drawing samples from the stochastic switch network. Thompson sampling [19] is implemented with a Gaussian prior distribution and estimates the posterior of the total reward when deploying each controller. As illustrated in



(a) The switching policy of SGuidance



(b) The switching policy of Thompson Sampling

Fig. 6: Above figures illustrate the spatial density of using each controller (DDPG, PID or OA) in three different learning periods. Each row represents the employment of a single controller and each column indicates the early (0-10k steps), middle (40k-50k steps) or late (80k-90k steps) training stage. Note that all grey areas are obstacles.

Fig. 5, **SGuidance** has the best performance while **Uniform** is the worst. Note however, that **Uniform** actually learns the fastest in the first 20k steps. **ThompsonSampling** has a comparable performance to **Argmax**. The former learns slightly faster mainly due to the stochasticity and keeps seeking a potentially better switching policy. Later **Argmax** outperforms **ThompsonSampling** and **Uniform** as it selects controllers based on current observations while both **ThompsonSampling** and **Uniform** do not. When comparing **Argmax** with **SGuidance**, it has much bigger variance on the total reward. This is because **Argmax** is a biased sampler and the introduced bias in turn damages its final performance since there is less exploration.

4) Further Comparison with Thompson Sampling: To understand the benefits obtained by learning a situation-aware switching policy, we compare the usage of different controllers between **SGuidance** and **ThompsonSampling** during the training in Fig. 6(a) and Fig. 6(b). They demonstrate the spatial

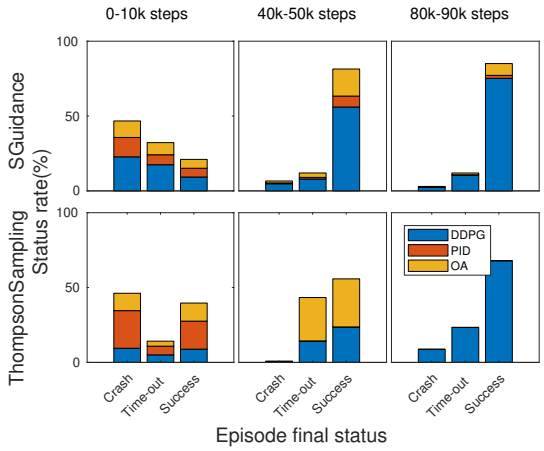


Fig. 7: Above figures display the rate of reaching different final status in an episode when applying **SGuidance** and **ThompsonSampling** respectively. Each column represents a training period and different segments in each bar indicates the usage ratio of different controllers within this period.

density of controller usage, where each row represents the employment of a single controller and each column indicates the early (10k-20k steps), middle (50k-60k steps) or late (90k-100k steps) training period. Note that the training is based on the *ROS Stage simple* environment in Fig. 2(a) and in each episode the robot is initialised in the centre and the destination is randomised.

Fig. 6(a) and Fig. 6(b) show the main differences between the switching policy of **SGuidance** and **ThompsonSampling**. **SGuidance** explores the strengths and weaknesses of each controller in different situations while **ThompsonSampling** does not. Notice in the second row of Fig. 6(a) that **SGuidance** gradually abandons candidate actions from PID controller when it is still surrounded by the obstacles and only uses it when approaching the target. The OA controller (third row) is occasionally adopted to avoid obstacles. However, in 6(b), it is shown that the usage of each controller does not depend on the position in the map. In fact, **ThompsonSampling** completely abandons PID and OA controller eventually. Since it can only estimate the overall performance of the controller in an episode, it will easily converge to DDPG when it is confident enough about the superiority of DDPG after a long training period without considering the context.

Fig. 7 plots the rate of reaching different final states and the average usage of each controller in an episode when applying **SGuidance** and **ThompsonSampling** respectively. It further demonstrates the difference between their learnt switching policies in terms of situation dependence from a more statistical perspective. In each sub-figure, three columns represent three final states, corresponding to different situations, i.e. crashing with obstacles (**Crash**), timing out (**Time-out**) or successfully reaching the destination (**Success**). For **ThompsonSampling**, the ratio among the usages of all controllers always remain the same in different situations, whilst for **SGuidance**, they are similar at first but then diverge over time.

5) Construction of Stochastic Switch Function: Since we have introduced a bias, which can be considered as a preference, of selecting different controllers with stick-breaking, this part

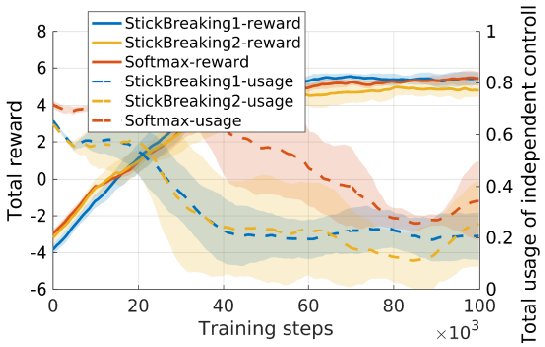


Fig. 8: Discussion on different stochastic switch function. The left y-axis shows the total reward of all the methods, and the right y-axis shows the total usage of the independent controllers.

will investigate the effect of setting different orders for PID and OA controller in stick-breaking. In Fig. 8, the **StickBreaking1** (DDPG, PID, OA) represents the function we applied in the paper and the **StickBreaking2** (DDPG, OA, PID) used an alternative order of the independent controllers. More specifically, according to Eq. 6, **StickBreaking1** and **StickBreaking2** both set η_1 with DDPG controller and give different order with PID and OA controller where η_2 is assigned with the PID controller in **StickBreaking1** but with OA controller in **StickBreaking2**. As shown in the figure, **Softmax** is able to achieve almost adequate performance compared to **StickBreaking1** in terms of the total reward. However, according to the total usage of independent controllers, the DDPG component is being less used in **Softmax** than **StickBreaking1** and **StickBreaking2**. Although the two stick breaking functions have similar total usage of independent controllers, **StickBreaking2** performs slightly worse than **StickBreaking1**, which shows that the order of independent controllers has a small effect on the performance. Hence, the softmax function is a safe choice to construct the stochastic switch function. However, the prior knowledge about the performance of simple controllers could be used to benefit the learning in stick breaking construction. For instance, Fig. 4 shows PID brings more benefit than OA when incorporating with stochastic switch, and Fig. 8 also shows the **StickBreaking1** performs slightly better than **StickBreaking2**.

6) Complex Environment and Simple Reward: In this experiment, we test **SGuidance** in a more cluttered environment as displayed in Fig. 2(b). Through our experiments, we found that although using the dense reward function can accelerate training in the simple environment, it makes the actor network converge to a suboptimal policy in the complex one. As mentioned in [20], the greedy behaviour learnt from the dense reward makes the robot approach the destination as quickly as possible without keeping a reasonably safe distance to the obstacles, resulting in a higher probability of collisions. Hence we use Eq. 10 as the reward function here, making the network converge slower but at a safer policy.

$$r_t^{sparse} = \begin{cases} R_{crash}, & \text{if robot crashes} \\ R_{reach}, & \text{if robot reaches the goal} \\ -C, & \text{otherwise} \end{cases} \quad (10)$$

The first two rows of Fig. 9 provide the trend of obtaining

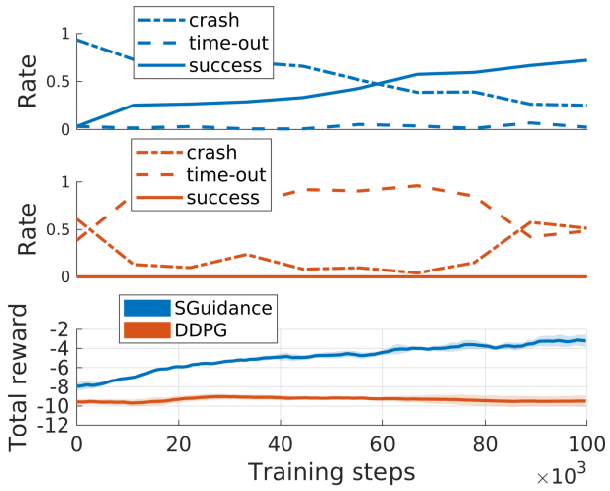


Fig. 9: The figure illustrates the experimental results on the complex environment as shown in Fig. 2(b) with sparse rewards. The robot only receives a reward when crashing or reaching the destination, besides a time penalty at each step. The first two rows represent the rate of getting different ending status (crash, time-out or success) of each navigation episode when deploying the guidance or not. The total reward of each episodes is also displayed in the last row.

different ending status of each episode through the training period with/without the stochastic guidance respectively.

As shown in Fig. 9, with the growing performance of the DDPG controller, **SGuidance** reaches the destination more frequently. However, the naive **DDPG** without any guidance never explores to reach the destination safely.

7) Inference without Guidance: This experiment explores whether the trained DDPG policy is able to independently carry out navigation with all the heuristic controllers turned off after it has been trained with stochastic guidance. We firstly train the DDPG with **SGuidance** and compare the inference performance of keeping or turning-off the stochastic guidance. The experiment is carried out in the *ROS stage simple* environment with reward defined in Eq. 1. As shown in Table I, the overall performance only has a slight decrease after switching-off the guidance in testing.

Interestingly, when we carry out the same test in the *ROS stage complex* environment, turning-off the guidance even improves the success rate (**SR**) and reduces the crash rate (**CR**). This is because **SGuidance**, which is similar to **ThompsonSampling** at this point, keeps exploring potentially better options. However, this exploration also occasionally leads to erroneous actions especially in the late training period or the testing phase and is counter productive. This applies more to the complex environment as a single wrong action may lead to a crash.

8) Learning with Different γ: In this experiment we investigate how different values of the reward discount factor γ affect the switching policy learnt by **SGuidance**. As shown in Fig. 10, with a smaller γ, **SGuidance** samples the DDPG less frequently since it becomes a more greedy controller and has a worse long-term performance. As a consequence, when the policy in DDPG cannot reach the destination, **SGuidance** prefers the OA controller which, at least, can avoid collisions, i.e. large instant negative rewards.

Guidance	Policy-U	PID-U	OA-U	SR	TR	CR	AR
Test in Simple World with Dense Reward							
Keep	88.1%	3.5%	8.4%	92%	7%	2%	5.01
Turn-off	100%	0%	0%	90%	4%	6%	4.91
Test in Complex World with Sparse Reward							
Keep	81.7%	2.5%	15.8%	81%	14%	5%	-2.29
Turn-off	100%	0%	0%	83%	16%	1%	-2.34

TABLE I: Comparing the performance between keeping/turning-off the guidance after training in terms of the usage of three controllers (**Policy-U**, **PID-U**, **OA-U**), the rate of the final state of each episode (**SR**: success rate, **TR**: time-out rate, **CR**: carsh rate) and also the average total reward of each episode (**AR**).

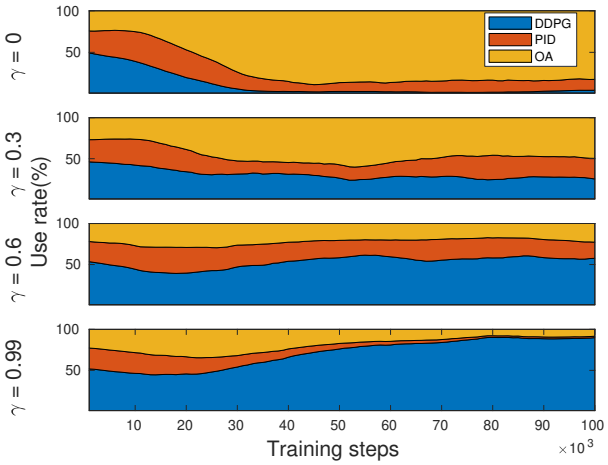


Fig. 10: The use rate of each controller by **SGuidance** with different discount factor γ. Each figure draws the use rate of three controllers with a different γ.

C. Navigation in Real World Environment

In this experiment, we qualitatively analyse the performance of our model applied in real world environments. The model is trained in a simulated world built by *ROS Gazebo* (Fig. 2(c)), and directly transferred into the real world scenario without any fine tuning in order to verify the effectiveness and strong generalisation of the model. The learning curves, which are illustrated in Fig. 11(a), show that **SGuidance** can outperform naive **DDPG**.

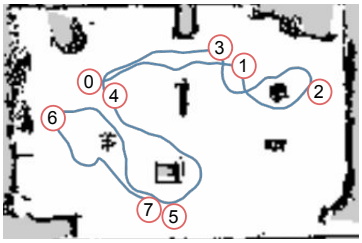
A Turtlebot 2 robot mounted with a Kinetic depth camera is used as the mobile platform. Unlike the observation from laser scanner which is simulated in *ROS Stage*, the dimension of state space for using a depth camera is dramatically increased. Therefore, a 3-layer convolutional neural network is employed (as in Fig.1) to provide geometric representations. Other inputs, i.e. velocity and goal location, are concatenated with the geometric representation into a dense input representation.

Since the ground truth of the robot locations is not available in the real world environment, we apply the off-the-shelf AMCL ROS package[5] for providing the estimation of the robot location, and calculating the destination position in the local coordinate frame. In order to improve the localisation accuracy, we record the map of the environment with Gmapping ROS package[6]. It

[5]http://wiki.ros.org/amcl
[6]http://wiki.ros.org/gmapping

(a) Learning curves in Gazebo Simulator

(b) Real world environment



(c) Destinations and navigation trajectory

Fig. 11: **(a)** The total reward achieved by **SGuidance** and **DDPG** models in ROS Gazebo simulator as comparison. Note that since Gazebo simulates physics properties, the control policy is more difficult to learn and the training takes a longer time when compared to use ROS stage. **(b)** The real world scenario. A turtlebot is used as the mobile platform and several boxes are placed in the room as obstacles. **(c)** The room layout and obstacles are the black areas. The blue curve represents the trajectory of the robot and the goals are plotted with red circles where the number indicates the sequence.

is worth mentioning that this global map is not used by the navigation component of the model during training or testing. The obstacles are laid out in the room as illustrated in Fig. 11(b). The target of this experiment is to employ the learned policy and control the robot to reach several destinations successively without any collision. Fig.11(c) shows the trajectory of the robot (blue curve). Notice that the robot can smoothly avoid all the obstacles and reach each target successfully by only learning in simulation with the proposed stochastic guidance model.

V. Related Work

Many works have applied DRL on robotic problems, e.g. navigation [21], [22], [23], [24], [25], [26], [27] and manipulation [28], [29]. Since most of the robotics problems involve continuous control, policy based approaches such as policy gradient [30] or actor-critic method [31], e.g. DDPG [3], are widely used as the conventional approaches. Introducing positive bias is a common approach for alleviating the issue. [16] assigns higher weights to the data where the model has less confidence to improve the efficiency of sample usage. [32] leverage the concept of information gain when exploring new policies. Unlike above approaches where the bias are tightly merged into the models, our framework incorporates extra knowledge as stochastic guidance without imposing any change to the underlying approach.

Thompson Sampling [33] shares the similar spirit of our switching mechanism which learns to switch among different controllers. The difference is that, instead of explicitly calculating the posterior for updating in Thompson Sampling, our framework directly employs neural networks to construct the latent distributions which are trained jointly with the DDPG component by backpropagation. The advantage is that the switch function can be easily built and conditioned on all of the sensor inputs so that it chooses different controllers according to different contexts/conditions. In addition, the target of our framework is more focused on training a better DDPG component, which is able to benefit from the low-variance gradient estimator due to the better samples generated by the stochastic switch.

To effectively stabilise the estimated gradients from REINFORCE, we have applied variance reduction techniques. This is a common topic across different fields. Johnson et al. [34] proposed Stochastic Variance Reduced Gradient(SVRG) to reduce the variance of gradients estimation in vanilla stochastic gradient descent with control variates. In [35], another control variate is introduced based on Taylor expansion of the off-policy critic. It can effectively alleviate the high variance problem in policy gradient while keep its advantage of sample efficiency but is computationally expensive. Neural Variational Inference also suffers from high variance problem, several approaches have been proposed. One major approach is by applying delicately designed baselines [15], [36], [37], and another is based on continuous relaxation [38] when the latent variable is discrete.

In [39], Leonetti et al. investigated a low level integration of RL and external controllers where the RL algorithm only explores with feasible actions provided by the planner, these heuristics can not be discarded, both for training and testing. Therefore, the performance of the learner very depends on, if not limited by, the capability of the heuristics. By contrast, in our framework, DDPG can explore the full action space by itself alongside the guidance during the whole training process and can eventually work independently.

Yang et al. [27] proposed a hierarchical reinforcement learning based method. It learns basic skills with multiple independent actors and finally samples actions from the one with the highest Q-value estimation which is similar to the deterministic baseline 'argmax' mentioned in section IV-B3. From the experiments it is proved that our stochastic switch outperforms the deterministic counterpart. Furthermore, our goal is only to assist the training of DDPG where all the extra controllers will be discarded eventually.

The proposed framework is also related to imitation learning [40], [6]. In the imitating approaches, the demonstrator must be good enough to solve the problem. But our approach stays in a larger spectrum since it should be compatible with both simple heuristic and complex learn-able controllers. This is because, how to use these controllers is decided by the switching policy. It learns to use suitable controllers under different observations and these controllers might be completely useless in other cases. Furthermore, after obtaining a good DDPG component, all of the heuristics (or controllers) can be gotten rid of, and we are able to test in the environment just by the DDPG component without using any of the heuristics.

VI. Conclusion

This paper proposes a new framework for effectively incorporating heuristic knowledge to overcome the high variance

issue in learning DDPG. The experiments demonstrate that the stochastic switch allows an agent to balance the learning from exploration and heuristics, which significantly bootstraps the performance of navigation that surpasses state-of-the-art baseline models. More interestingly, the DDPG component remains independent and can be tested in isolation from other controllers. When transferring the policies in the real world, the robot is able to successfully carry out navigation tasks, which indicates the robustness and strong generalisation of our proposed framework.

ACKNOWLEDGMENT

This work was supported by EPSRC Mobile Robotics: Enabling a Pervasive Technology of the Future (grant No. EP/M019918/1).

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] 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.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2016.

[4] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[5] B. Piot, M. Geist, and O. Pietquin, "Bridging the gap between imitation learning and inverse reinforcement learning," *IEEE Trans. Neural Netw.*, vol. 28, no. 8, pp. 1814–1826, Aug 2017.

[6] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4423–4430, Oct 2018.

[7] R. Simmons and S. Koenig, "Probabilistic robot navigation in partially observable environments," in *International Joint Conference on Artificial Intelligence*, vol. 95, 1995, pp. 1080–1087.

[8] Y. Sun, M. Liu, and M. Q.-H. Meng, "WiFi signal strength-based robot indoor localization," in *IEEE International Conference on Information and Automation*. IEEE, 2014, pp. 250–256.

[9] K. J. Åström and T. Hägglund, *PID controllers: theory, design, and tuning*. Instrument society of America Research Triangle Park, NC, 1995, vol. 2.

[10] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[11] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[12] J. Sethuraman, "A constructive definition of dirichlet priors," *Stat. Sin.*, pp. 639–650, 1994.

[13] M. Khan, S. Mohamed, B. Marlin, and K. Murphy, "A stick-breaking likelihood for categorical data analysis with latent gaussian models," in *Artificial Intelligence and Statistics*, 2012, pp. 610–618.

[14] Y. Miao, E. Grefenstette, and P. Blunsom, "Discovering discrete latent topics with neural variational inference," in *International Conference on Machine Learning*, 2017, pp. 2410–2419.

[15] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," *arXiv preprint arXiv:1402.0030*, 2014.

[16] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[17] B. P. Gerkey and K. Konolige, "Planning and control in unstructured terrain," in *ICRA Workshop on Path Planning on Costmaps*, 2008.

[18] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[19] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen *et al.*, "A tutorial on thompson sampling," *Foundations and Trends® in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018.

[20] L. Xie, S. Wang, S. Rosa, A. Markham, and N. Trigoni, "Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6276–6283.

[21] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[22] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *IEEE International Conference on Robotics and Automation*. IEEE, 2017, pp. 3357–3364.

[23] F. Sadeghi and S. Levine, "(cad)2rl: Real single-image flight without a single real image," 2017.

[24] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards monocular vision based obstacle avoidance through deep reinforcement learning," *arXiv preprint arXiv:1706.09829*, 2017.

[25] X. Truong and T. D. Ngo, "Toward socially aware robot navigation in dynamic and crowded environments: A proactive social motion model," *IEEE Trans. Automat. Sci. Eng.*, vol. 14, no. 4, pp. 1743–1760, Oct 2017.

[26] C. Ye, N. H. C. Yung, and D. Wang, "A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance," *IEEE Trans. Syst., Man, Cybern. B*, vol. 33, no. 1, pp. 17–27, Feb 2003.

[27] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass, "Hierarchical deep reinforcement learning for continuous action control," *IEEE Trans. Neural Netw.*, vol. 29, no. 11, pp. 5174–5184, Nov 2018.

[28] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *IEEE International Conference on Robotics and Automation*. IEEE, 2017, pp. 3389–3396.

[29] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine, "Collective robot reinforcement learning with distributed asynchronous guided policy search," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2017, pp. 79–86.

[30] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, 2000, pp. 1057–1063.

[31] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Trans. Syst., Man, Cybern. C*, vol. 42, no. 6, pp. 1291–1307, Nov 2012.

[32] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," in *Advances in Neural Information Processing Systems*, 2016, pp. 1109–1117.

[33] M. J. Kim, "Thompson sampling for stochastic control: The finite parameter case," *IEEE Trans. Automat. Contr.*, vol. 62, no. 12, pp. 6415–6422, Dec 2017.

[34] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 2013, pp. 315–323.

[35] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," 2017.

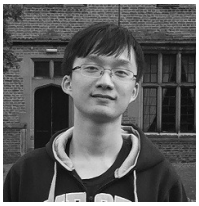
[36] M. T. R. AUEB and M. Lázaro-Gredilla, "Local expectation gradients for black box variational inference," in *Advances in Neural Information Processing Systems*, 2015, pp. 2638–2646.

[37] A. Mnih and D. Rezende, "Variational inference for monte carlo objectives," in *International Conference on Machine Learning*, 2016, pp. 2188–2196.

[38] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *Stat*, vol. 1050, p. 1, 2017.

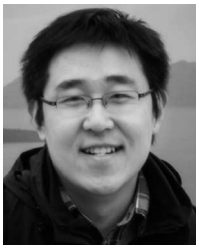
[39] M. Leonetti, L. Iocchi, and P. Stone, "A synthesis of automated planning and reinforcement learning for efficient, robust decision-making," *Artificial Intelligence*, vol. 241, pp. 103–130, 2016.

[40] Y. Duan, M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," *arXiv preprint arXiv:1703.07326*, 2017.



Linhai Xie is a DPhil student at the Department of Computer Science, University of Oxford. Before that, he obtained his BEng degree at National University of Defense Technology, China. His research is focused on learning based robot perception and autonomy, including robot navigation, reinforcement learning, deep learning, robotic vision and SLAM.

Yishu Miao is the founder & CEO of MO Intelligence Ltd, an AI startup for automating digital building twins. He received a DPhil degree in Computer Science from the University of Oxford, a Master in Data Mining from Tsinghua University. He worked as a research intern at DeepMind during the year of 2016 and 2017. He is a pioneer research scientist in deep learning, passionate entrepreneur, and experienced software engineer. He has years' experience on landing AI technologies in multiple different areas.



Sen Wang is an Assistant Professor in Robotics and Autonomous Systems with Heriot-Watt University and a faculty member of the Edinburgh Centre for Robotics. Previously, he was a postdoctoral researcher with the University of Oxford. His research focuses on robot perception and autonomy using probabilistic and learning approaches, especially autonomous navigation, robotic vision, SLAM and robot learning.

Phil Blunsom is originaly from Australia, where he obtained his PhD degree at the University of Melbourne under the supervision of Timothy Baldwin, Steven Bird and James Curran. He then came to the United Kingdom as a Research Fellow at the University of Edinburgh. There he worked on the application of machine learning techniques to machine translation with Miles Osborne. Since 2009 he has been at the University of Oxford, both in the Department of Computer Science and as a Fellow of St Hugh's College. Since 2014 he has been splitting his time between Oxford and DeepMind London, where he works with the Natural Language group.

Zhihua Wang is a DPhil student at the Department of Computer Science, University of Oxford. Before that, he obtained MPhil degree from University of Cambridge and BEng degree from University of Manchester. His research interests lie in Cyber-Physical systems, deep learning and intuitive physics.

Changhao Cheng is currently a PhD student in Department of Computer Science, University of Oxford. Before that, he obtained his MEng degree at National University of Defense Technology, China, and BEng degree at Tongji University, China. His research interests include machine learning, robotics and cyber physical systems. He is working on machine (deep) learning to process time-series sensor data for localization, navigation, mapping and perception, in support of robots, mobile devices, self-driving vehicles and Internet of Things (IoT).

Andrew Markham is an associate professor in the Department of Computer Science, at the University of Oxford. He received his BSc (Hons)(2004) degree and PhD(2008) degree both from the University of Cape Town, South Africa. He broadly works within the area of cyberphysical systems and machine learning, with application to a number of interdisciplinary areas such as wildlife tracking, earthquake monitoring and industrial safety. Current research interests include data driven techniques for positioning, scene reconstruction and sensor processing.

Niki Trigoni is a Professor at the Oxford University Department of Computer Science and a fellow of Kellogg College. She obtained her DPhil degree at the University of Cambridge (2001), became a postdoctoral researcher at Cornell University (2002-2004), and a Lecturer at Birkbeck College (2004-2007). At Oxford, she is currently Director of the EPSRC Centre for Doctoral Training on Autonomous Intelligent Machines and Systems, a program that combines machine learning, robotics, sensor systems and verification/control. She also leads the Cyber Physical Systems Group, which is focusing on intelligent and autonomous sensor systems with applications in positioning, healthcare, environmental monitoring and smart cities. The group's research ranges from novel sensor modalities and low level signal processing to high level inference and learning.