SnapNav: Learning Mapless Visual Navigation with Sparse Directional Guidance and Visual Reference

Linhai Xie[1,2], Andrew Markham[1] and Niki Trigoni[1]

***Abstract*— Learning-based visual navigation still remains a challenging problem in robotics, with two overarching issues: how to transfer the learnt policy to unseen scenarios, and how to deploy the system on real robots. In this paper, we propose a deep neural network based visual navigation system, SnapNav. Unlike map-based navigation or Visual-Teach-and-Repeat (VT&R), SnapNav only receives a few snapshots of the environment combined with directional guidance to allow it to execute the navigation task. Additionally, SnapNav can be easily deployed on real robots due to a two-level hierarchy: a high level commander that provides directional commands and a low level controller that provides real-time control and obstacle avoidance. This also allows us to effectively use simulated and real data to train the different layers of the hierarchy, facilitating robust control. Extensive experimental results show that SnapNav achieves a highly autonomous navigation ability compared to baseline models, enabling sparse, map-less navigation in previously unseen environments.**

I. Introduction

"Where is the reception?"

"Go straight and turn left when you see the exit."

The above conversation is a very common and efficient human interaction when querying the route to an unknown destination. This kind of instructions that consist of a sequential action paired with a visual reference are widely used not only in navigation but also in various kinds of activities such as reading the user guide for a new product. This type of navigational instructions have two prominent characteristics. The first is the direct conjunction between the action and the visual observation spaces which provides specific guidance about what to do and where to do it. The second notable feature is the inherent sparsity as complex instructions can be distilled into a few key actions at visually important waypoints or cues, relying on the innate capacity of humans to navigate between these points. Together, this yields accurate instructions that can be efficiently communicated.

This observation triggers an interesting question: can robots mimic those human behaviours to navigate in a completely unknown environment when supplied with very sparse guidance? Such a robot system will be highly efficient at communication and possess a strong generalisation ability towards unfamiliar scenarios, which will help contribute towards establishing an intelligent multi-agent society.

In this paper, we present a deep neural network based system, SnapNav, as a practical solution to mapless visual navigation in unknown environments. Firstly, it can navigate in an unknown environment with only a few pieces of guidance. As shown in Fig. 1 the guidance consists of a snapshot image and the desired action (turn left, turn right, stop) before each turning or termination point along the path. Secondly, the navigation system is designed with a two-level hierarchy for fully benefiting from the training data from different domains, i.e. appearance or depth observations, simulation or reality, and straightforwardly deploying the learnt policy on real robots.

Our contributions are summarised as follows:

- We propose a novel visual navigation system which enables the robot to navigate in unknown environments with very sparse guidance.
- A two-level architecture of the network is proposed and trained with different learning mechanisms to easily transfer the learnt policy from simulation to real world.
- We introduce a novel self-supervised training approach with multiple learning signals to obtain robust guidance from the high-level commander.
- We show that the trained network can be used to navigate in real-world experiments.

[1]Authors are with Department of Computer Science,University of Oxford, Oxford OX1 3QD, United Kingdom, `first name.last name@cs.ox.ac.uk`

[2]Linhai Xie is also with Department of Mechatronic Engineering and Automation, National University of Defense Technology, Changsha 410073, China

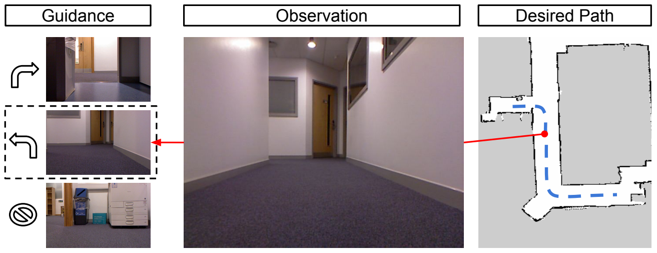

Fig. 1: An example of the visual navigation task with sparse directional guidance. The robot automatically selects from the provided guidance based on current observation.

II. Related Work

Deep learning has recently been broadly applied in robot navigation [1]–[3] and learning-based mapless visual navigation has demonstrated remarkable performance in complex environments. Many works propose to learn a shortest path strategy by encoding the environmental information into the parameters of deep neural network [4]–[6]. Although these techniques exhibit robust navigation capabilities, these agents are actually overfitting the training environment and cannot apply the learnt experience to previously unseen scenarios. Another branch of mapless navigation approaches is local navigation [7]–[9] where the prerequisite of a known relative goal position largely restricts its real world applications to ones where accurate location exists.

A. Visual teach and repeat

Recently, researchers have introduced learning agents that can follow a demonstrated path [10]–[13] which is similar to the traditional Visual teach and repeat (VT&R) [14] in robotics. This group of solutions require simple descriptions of the environment when navigating in an unknown environment, e.g. raw camera image sequences, or additionally, the labelled actions, instead of a pre-defined map. Among them only [11], [13] can be deployed on real robots. However the former is trained with a large amount of manually labelled real-world data from an omni-camera while the latter requires an explicit localisation of the demonstrated image in the sequence, largely discounting the practicality. Furthermore, reliance on a long video stream hinders efficient communication between agents.

B. Language based visual navigation

Natural language instructions, as a form of extremely sparse guidance, is also introduced in visual navigation [15], [16]. Although it imitates human behaviour and can produce a more autonomous agent, the difficulty of visual language grounding, i.e. associating the perception from two completely different modalities, together with the ambiguity of the natural language itself, limits the performance of language guided navigation. The agent mentioned in [16] which takes a list of thumbnail images of the street view in guidance is similar to our SnapNav but only learns high level navigation strategies in a simulated toolkit named StreetNav.

III. Task Description

The task we are investigating in this paper is similar to StreetNav in [16] which can be termed as visual path following with sparse guidance. Although there are some similarities between these works, we focus on a more robotic oriented perspective. In particular, we consider the challenges of real-time perception and control of an autonomous robot, and take the robot out of pure simulation into reality.

A. Task Decomposition

Although it is straightforward to solve the entire task with a single policy network as in [10], [16], it is non-trivial to deploy those systems to a real robot. On the one hand, due to the absence of a simulator that can simultaneously render realistic camera data and precisely capture the dynamics of robots, command and control strategies that are deployable in reality cannot be learnt with a single simulator. On the other hand, manually labelling real world data is labour expensive [11] and limitations on training samples rarely leads to a robust control policy due to the lack of exploration. However, by decomposing problem into a high level **commander** and a low level autonomous **controller**, we can learn sub-policies with separate kinds and sources of data to optimize each sub-task. More concretely, since the visual matching of current observations and snapshots in guidance heavily relies on the appearance of the observed objects, the command strategy demands data with high visual fidelity but can largely ignore the robot dynamics. Conversely, the control policy only focuses on the geometry of the surroundings for obstacle avoidance, robot dynamics and occasional commands/high-level actions. Therefore this policy can be more easily trained in a robot simulator where the robot dynamics as well as depth observations are finely modelled. The robot can also be exposed to a wide range of arbitrary world setups, allowing for more robust local navigation. Each sub-task will be formulated more specifically in the following parts.

B. Command Sub-Task

The commander C is supplied with n pairs of guidance $\{\mathbf{G}_i = (\mathbf{S}_i, m_i) | i = [1, 2, ..., n]\}$ where $\mathbf{S}_i$ and m_i represent snapshots and guidance commands respectively. Each snapshot $\mathbf{S}_i$ in the guidance records a first-person-view RGB image of the area where the robot should either alter direction or stop at a particular point. Since the robot is only given the order to vary direction or whilst terminating, there are only three types of commands in guidance, $m_i \in \{$“`Turn right`”, “`Turn left`”, “`Stop`”$\}$, with the implicit action being to carry on in a straight path.

Then with all pieces of guidance and the current image observation $\mathbf{O}_t$ from an on-board camera, the commander C predicts a high level command $c_t \in \{$“`Turn right`”, “`Go forward`”, “`Turn left`”, “`Stop`”$\}$ at each time step t as $c_t = C(\mathbf{O}_t, \mathbf{G}, \mathbf{h}_c)$ where $\mathbf{h}_c$ is the hidden state of the GRU cell in commander.

C. Control Sub-Task

The control sub-task is formalised as a Markov Desision Process (MDP). At time $t \in [1, T]$ the robot takes an action $\mathbf{a}_t \in \mathscr{A}$ according to the observation $\mathbf{X}_t$. 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 $\mathbf{X}_{t+1}$. The goal of this MDP 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.

More specifically, the action is the control signal of the robot, $\mathbf{a}_t = (a_t^v, a_t^\omega) \in \mathscr{A}$, where a_t^v and a_t^ω respectively denotes the expected linear and rotational velocity at time t. The observation $\mathbf{X}_t$ is a first-person-view depth image which can be directly accessed in a simulator, e.g. *ROS Gazebo*[1], or estimated from an RGB image with an off-the-shelf estimator [17] in the real world. The reward function r_t at time t is defined as:

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

where R_{crash} is a 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 (the next turning point or the final destination) at two consecutive time steps $t-1$ and t.

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

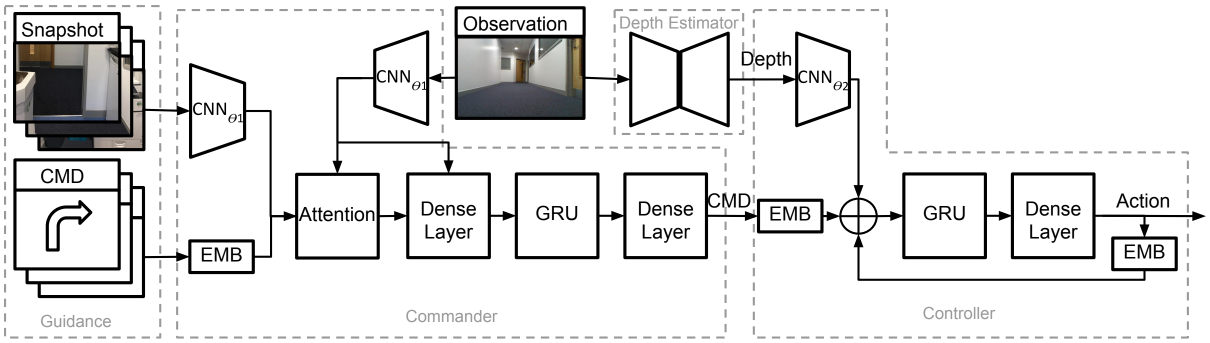

Fig. 2: The network architecture of SnapNav which consists of two modules. The commander firstly attends to a particular guidance instruction by finding the correct match with the current observation. It then publishes a high level command. The controller then predicts a low level robot action given the command, the estimated depth image and the previous predicted action. Note that commands are represented with the abbreviation "CMD", "EMB" denotes a linear embedding layer and $\bigoplus$ is the concatenation operation.

IV. Network Architecture

A. *Attention-Based Commander*

To accomplish the command sub-task, the commander is designed with convolutional neural network (CNN) layers and linear embedding (EMB) layers to process raw inputs, followed by a hard attention component and a recurrent policy network.

As illustrated in Fig. 2, image inputs to commander are firstly encoded by 5 convolutional layers, whereas commands are input to a linear embedding layer where the guiding snapshot $\mathbf{S}_i$, command m_i and the current observation $\mathbf{O}_t$ are transformed to vectors respectively as $\mathbf{v}_i^S = \text{CNN}_{\theta_1}(\mathbf{S}_i)$, $\mathbf{v}_i^m = \text{EMB}(m_i)$, $\mathbf{v}_t^O = \text{CNN}_{\theta_1}(\mathbf{S}_t)$. Since both the snapshots and current observation are the same type of data, the CNNs for encoding them have the same parameters θ_1.

Given vectorised inputs $\mathbf{v}_i^S$, $\mathbf{v}_i^m$ and $\mathbf{v}_t^O$, the goal is to choose the guidance which contains the most relevant snapshot w.r.t. the current observation. Intuitively, due to the sparsity of the guidance, only highly related snapshots should be matched for the command strategy at each time step. Thus, hard attention is preferable to soft attention which simply sums over all guidance instructions with different weights.

The hard attention is usually modelled with a discrete stochastic layer which is non-differentiable and thus has to be optimised with gradient estimation methods other than conventional backpropagation [18]. Fortunately, as shown in [16], [19], an alternative is to adopt a generalisation of the max-pooling operator to choose the optimal guidance instruction. This bypasses the non-differentiable problem:

$$(\mathbf{v}_{i^*}^S, \mathbf{v}_{i^*}^m) = \underset{(\mathbf{v}_i^S, \mathbf{v}_i^m)}{\text{argmax}}[\text{softmax}(-||\mathbf{v}_i^S - \mathbf{v}_t^O||_2)]. \quad (2)$$

It results in a sub-differentiable model and can be combined with other gradient-based models. Our experiments later prove that the attention performance can be improved by a large margin either by combining other gradient estimators such as the REINFORCE algorithm [20] with backpropagation or training with auxiliary tasks i.e. metric learning. Then the attended snapshot vector together with the embedded command is processed by a dense layer and concatenated with the encoded observation. Finally, a recurrent network is used to predict the attended command when the selected snapshot and the current observation are similar enough. If there is low similarity, the default "`Go forward`" command is issued.

B. *Controller*

Given the command from the commander $c_t = C(\mathbf{O}_t, \mathbf{G}, \mathbf{h}_c)$, the last action $\mathbf{a}_{t-1}$ and the depth image provided by the simulator or a depth prediction network $\mathbf{X}_t = D(\mathbf{O}_t)$, the controller outputs the best action to navigate through the environment, based on its trained policy. Similar to the commander, the raw depth image is encoded with a CNN $\mathbf{v}_t^X = \text{CNN}_{\theta_2}(\mathbf{X}_t)$, the command is also embedded as $\mathbf{v}_t^c = \text{EMB}(c_t)$ which are then used to predict the action by the controller $\mathbf{a}_t = \pi(\mathbf{v}_t^X, \mathbf{v}_t^c, \mathbf{h}_\pi, \mathbf{a}_{t-1})$ where $\mathbf{h}_\pi$ is the recurrent hidden state in the controller and θ_2 represents the parameters of the depth encoding CNN.

The recurrent network is of importance in the controller as it is required to decide precisely when to carry out the command for turning left or right. This is because the high level commands are aligned to maximal visual matches, which may not be precisely aligned to the desired turning point. This decoupling yields higher levels of autonomy, as the low level controller decides when it is best to turn.

V. Training

In this section we introduce our training mechanisms respectively for the controller and commander.

A. *Self-Supervised Commander Training Labels*

Rather than relying on manually annotated video, in this section we present a novel, self-supervised technique to create pseudo-labels. Given a raw video, the optical flow between subsequent frames is firstly estimated with FlowNet [21] and then segmented where optical flow is high i.e. likely turning points. This assumes that images collected along a straight, continuous trajectory are highly similar, whereas images around a corner show high disparity. Next, a snapshot is randomly sampled near the end of each segment and, together with k ($k = 20$ in this paper) nearest frames, is labelled with a random command drawn from {"`Turn`

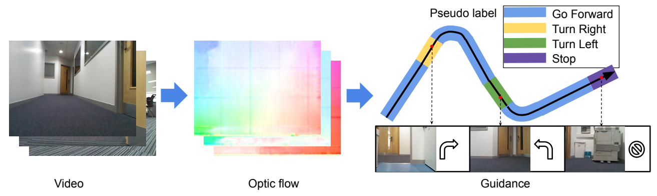

Fig. 3: The videos are firstly segmented based on the estimated optic flow. Then the pseudo labels of direction varying commands (yellow and green sections) are assigned before the agent actually makes the turn and is important to be labelled randomly instead of according to the ground-truth. The "`Stop`" command is only labelled at the end part of the video (the purple part) and the other frames are set to "`go forward`" as the default command (the blue areas).

`right`", "`Turn left`", "`Stop`"}. It is worth noting that the command of turning left or right before each turning point is not labelled according to the real actions taken by the agent during data collection but assigned randomly. This is the key to preventing the commander network from overfitting to small real-world datasets. Finally, the remaining frames in the sequence are all labelled with the command "`Go Forward`" as the default prediction of the commander.

Note that only a small dataset with 5k real sequential images are collected for fine-tuning the commander network. Before that, we initialise the network with 100k sequential images collected from *ROS Gazebo* with a standard navigation package *ROS Navigation*[2] automatically.

B. Training the commander

Given the labelled data, the commander can be optimised with several different learning signals, which can be used separately or jointly. We discuss the relative merits of each approach below.

a) ***Command Loss:*** The basic approach is to minimise the cross-entropy loss between the probability distribution of predicted commands $p \in (0,1)^M$ and the pseudo command labels $y \in \{0,1\}^M$ as:

$$Loss_{cmd} = -\frac{1}{T}\sum_{t=0}^{T}\sum_{m=0}^{M} y_t^m \log(p_t^m) \tag{3}$$

where T and M represent the length of the sequence and the number of categories of the command given a sequence of data. This relies on the sub-differentiable property of argmax$(\cdot)$ as mentioned in [16] and can be optimised using standard back-propagation algorithms.

b) ***Learning Attention Policy with REINFORCE:*** Our novel insight in this paper is to use the REINFORCE [20] algorithm to estimate gradients for learning a better attention policy which, due to the sub-differentiable argmax function, does not train well. The framework of REINFORCE algorithm usually models the policy learning process as an MDP where the agent is the attention layer in commander (it is independent from the MDP for control policy learning). Note that the attention layer does not contain any trainable parameter, therefore the attention policy $\pi(u_t|\mathbf{G},\mathbf{O}_t;\theta_1)$ completely relies on the CNN encoder that is parameterised by θ_1. Given the guidance $\mathbf{G}$ and the sequential observations $\mathbf{O}_t$, the attention policy induces a distribution over possible interaction sequences $s_{1:T} = \mathbf{O}_1, u_1, ..., \mathbf{O}_T, u_T$ where u_t and $\mathbf{O}_t$ are the attended location of snapshots and the observation at time step t. The target is to maximise the reward accumulated along an interaction sequence $s_{1:T}$ as $J(\theta_1) = \mathbb{E}_{p(s_{1:T};\theta_1)}[\sum_{t=1}^{T}\gamma^{T-t}r_t]$. Note that $p(s_{1:T})$ depends on the attention policy and the reward r_t at each time is proportional to the command prediction accuracy. The gradients are approximated using Monte-Carlo with N learning samples:

$$\begin{aligned}\nabla_{\theta_1} J &= \sum_{t=1}^{T}\mathbb{E}_{p(s_{1:T};\theta_1)}[\nabla_{\theta_1}\log\pi(u_t|\mathbf{G},\mathbf{O}_t;\theta_1)R_t] \\ &\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T}\nabla_{\theta_1}\log\pi(u_t^n|\mathbf{G}^n,\mathbf{O}_t^n;\theta_1)R_t^n.\end{aligned} \tag{4}$$

c) ***Metric Learning:*** As a further alternative to the REINFORCE learning signal, metric learning [22] can also be applied to explicitly improve the image encoding, by forcing the output embeddings to lie within a metric space. This can be considered as an auxiliary task alongside command prediction. Similar to [22], the triplet loss is adopted for metric learning. As each video is segmented whilst generating the pseudo command labels, an anchor image can be sampled from one of the segments. Then neighbouring images can be defined as positive (similar) images whilst all the images in other segments are labelled as negative (dissimilar) ones. Hence, after being encoded by the CNN, a triplet, which contains an anchor image vector $\mathbf{v}^a$, k^+ positive image vectors $\mathbf{v}^+$ and k^- negative image vectors $\mathbf{v}^-$, can be randomly generated from each raw image sequence and its loss function is formulated as follows:

$$Loss_{metric} = [\frac{1}{k^+}\sum_{i=1}^{k^+} l_2(\mathbf{v}_i^+ - \mathbf{v}^a) + \sigma - \frac{1}{k^-}\sum_{j=1}^{k^-} l_2(\mathbf{v}_j^- - \mathbf{v}^a)]_+, \tag{5}$$

where $[\cdot]_+$ is the hinge function, $l_2(\cdot)$ denotes the Euclidean distance, σ is a margin which is set to 1 and k^+ and k^- are set beneath 20 to ensure the the high similarity in positive images and a balanced proportion between positive and negative samples. This loss can be jointly minimised with the command loss through gradient back-propagation, therefore, it is more convenient and simple compared with utilising REINFORCE algorithm.

C. Reinforcement Learning for Control Policy

The control policy is purely learnt with DRL in virtual environments. To enhance the generalisation ability of the controller and decrease the possibility for the agent to memorise the environment, the geometry of the training environment is randomised every 40 episodes and in each episode the desired path for the robot to follow is also randomly generated as shown in Fig. 4. Then, for a robust transfer from a simulated robot to the real one, we use a

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

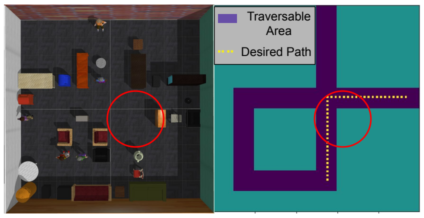

Fig. 4: An example of randomised environments in *ROS Gazebo* and its map. An oracle commander is designed accordingly to give the robot correct commands before it reaches the turning and termination point. The red circle highlights a challenging situation where the intersection is not fully constrained with obstacles which can easily confuse the robot.

finely modelled Turtlebot2 robot [3] in both the virtual and real world which is equipped with a Microsoft Kinect [4] to capture both depth and RGB images. Note that SnapNav can use either the groundtruth depth provided by the kinect or the estimated one, compatible to the system only with a monocular camera.

We test the learning of control policy with several different algorithms, e.g. DDPG [23], RDPG [24] and DRQN [25] and finally choose DRQN which exhibits the best performance in our task. The low-level controller is firstly trained with the oracle commander which always gives the agent correct commands according to the robot position in the desired path for 1M training steps and later fine-tuned with the noisy predicted commands from the learnt command policy for 0.5M training steps. It shows a constantly improved overall performance with different commanders in Table. I. The training is carried out on a single GTX970 GPU and each run takes about 20 hours where the control frequency is 5 Hz and the simulator is 4x times faster than the real world.

VI. Experiments

We carry out a model ablation study and real world tests to evaluate the optimality of the proposed model for SnapNav, and its generalisation ability to real-world scenarios.

A. Model Ablation Study in Virtual World

Each model in the ablation study is tested in random environments similar to Fig. 4 with 1000 independent runs and the success rate (SR) of reaching the final destination is used as the metric.

*a) **Attention policy learning**:* We compare several different models and learning signals for training the commander. The most basic model does not utilise the attention mechanism and simply sums over all the encoded guidance for command prediction. We term this **AllSum**. Then, since soft attention is reported to have similar performance as **AllSum** in [16], we consider the hard attention model as described in Sec IV-A which is purely optimised with the command loss learning signal and is termed as **HardAtt**. Next, we add the REINFORCE (**REINFORCE**) or metric learning (**Metric**) learning signal. We also combine three learning signals together (**Combined**).

[3]https://www.turtlebot.com/turtlebot2/
[4]https://en.wikipedia.org/wiki/Kinect

	DRQN	Finetuned DRQN
AllSum	50.2%	54.4%
HardAtt	60%	61.6%
REINFORCE	**72.7%**	74.2%
Metric	70.2%	73.6%
Combined	71.2%	**74.9%**
Oracle	85%	-%

TABLE I: Success Rate (SR%) with different commanders as well as using the DRQN puerly initialised with the Oracle commander or further fine-tuned with each commander.

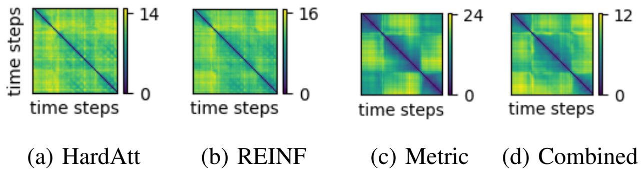

Fig. 5: The Euclidean distance matrix of the encoded images in a sampled video with two direction changing actions. Note that REINF indicates the REINFORCE algorithm.

From the results shown in Table I, we can clearly see that commanders that employ the attention mechanism significantly outperform **AllSum** which proves that attending to a specific snapshot is essential for the command prediction task. Compared with prior art, our introduction of either the REINFORCE algorithm or metric learning yield further, substantial gains.

To better understand the reasons for this difference, Fig. 5 illustrates the Euclidean distance matrix of a sampled video after being encoded by the commander. It is shown that compared with **HardAtt** and **REINFORCE**, **Metric** and **Combined** learn an encoder that clearly distinguishes images from different segments more clearly. Thus, this should make it easier to attend to the correct snapshot during navigation. However, according to Table I, the REINFORCE learning signal is more significant at improving the overall performance of the commander compared with the metric learning. This may be because that REINFORCE algorithm refines the attention policy directly according to the accuracy of the command prediction, whilst metric learning is a manually introduced bias in the encoding space which is not entirely related to the task of navigation.

*b) **Control policy learning**:* Selecting the suitable DRL approach to train the controller is of key importance for the entire system performance. From the learning curves shown in Fig. 6, we firstly prove the necessity of introducing the recurrent network into the controller model by comparing **DDPG** and **RDPG** where the latter is the recurrent version of the former. Because the command is assigned to the controller before the robot reaches the turning point, the recurrent network becomes important for memorising recent commands. Then we investigate the improvements brought

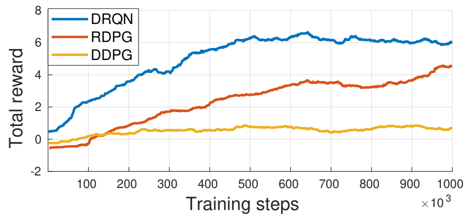

Fig. 6: The smoothed learning curves of DRQN, RDPG and DDPG respectively. Each algorithm is trained with 1 million steps in *ROS Gazebo* with the oracle commander.

	Success	TimeOut	Crash
DDPG	3.4%	31.8%	64.8%
RDPG	60.9%	36.6%	**2.5**%
DRQN	**85**%	**10.3**%	4.7%

TABLE II: Distribution of rewards over 1000 testing episodes for different controllers. Note that **Success** means the successful arrival of the final destination while **TimeOut** and **Crash** denote the situations where the robot cannot reach the destination within 300 steps or collides with an obstacle respectively.

by restricting the action and search space by exploiting a value-based approach **DRQN**. The continuous action space ($[0, 0.3]$ m/s for linear speed and $[-\frac{\pi}{6}, \frac{\pi}{6}]$ rad/s for angular speed) is discretised into three options, i.e. going straight forward with the maximum linear velocity and turning left or right with the maximum angular velocity and a slower linear velocity. Correspondingly, the Q-network only requires to estimate the Q-values of these options given the current observation and the hidden state of the recurrent network which substantially narrows down the search space and boosts the learning efficiency.

Table II looks deeper into the learnt policies. **DDPG** terminates most episodes with collisions as it attempts to maintain the expected total reward by minimising the probability of moving in an opposite directions to the destination, as this incurs an immediate negative reward. In contrast, its recurrent version (**RDPG**) achieves a much higher success rate due to the possibility of memorising the historic commands and heading to the correct directions accordingly at intersections. However the large exploration space for continuous action hinders the network from learning a stable and robust turning behaviour in highly randomised junctions, especially in the intersections with uncommon geometries as highlighted in Fig. 4. Different from **DDPG** and **DRPG**, **DRQN** explores in a discretised action space and shows significant performance gains. Therefore, we adopt **DRQN** as the final model for the low-level controller in real-world experiments.

B. Real-World Test and Comparison with Baselines

we evaluate the learnt policy in real-world scenarios to demonstrate its utility in real robotic scenarios. Fig. 7 demonstrates a example real-world test which qualitatively examines the obstacle avoidance ability and the robustness against the changing environments of SnapNav.

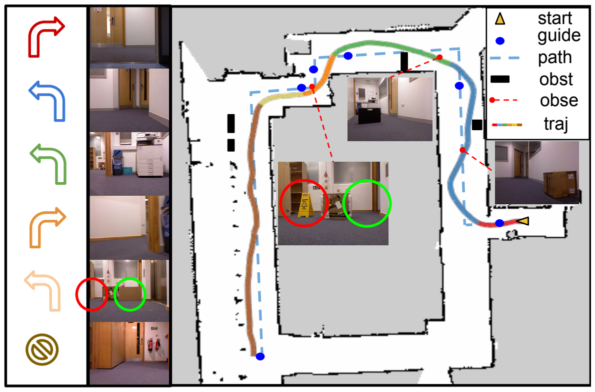

Fig. 7: An example real world test. The legends from top to bottom are: start position, guidance recording position, path in demonstrating phase, random obstacles in testing phase, robot observations and robot trajectory in testing. The attention location is illustrated by different colours in the robot trajectory which corresponds to the colour of actions in the guidance. The red and green circles emphasises the changing visual appearance and geometry of the environment between demonstrating and testing. More details are provided in the submitted video.

	SR	Dist(m)	ImgNum
SnapNav	**80**%	**24.29**	**6**
SimOnly	40%	20.19	**6**
VT&R	20%	18.32	326

TABLE III: Evaluations in real world environments. The metrics used are: success rate (**SR**), the average travelled distance before collisions or reaching the destination (**Dist**) and the number of images used as guidance (**ImgNum**). Each model is tested with 5 independent runs.

Furthermore, two baseline models are proposed for quantitative comparison as shown in Table III. **SimOnly** implements a similar model as [16] with additional depth observation and is purely trained in the simulator through DRL. The other one is the supervised variant of the deep **VT&R** model in [26] based on the limited real-world data. The former proves that SnapNav has an enhanced generalisation ability between virtual and real worlds, realised by the two-level hierarchy. The latter is a densely guided counterpart which verifies the effectiveness of navigating with sparse guidance and the restricted generalisation ability of supervised learning with limited samples.

VII. Conclusion

A practical solution is introduced in this paper for mapless navigation which can navigate in completely unknown environments with very sparse guidance. The experiments show its highly autonomous navigation performance with a strong generalisation ability from simulation to real world.

Acknowledgment

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

References

[1] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 5129–5136.

[2] C. Richter, W. Vega-Brown, and N. Roy, "Bayesian learning for safe high-speed navigation in unknown environments," in *Robotics Research*. Springer, 2018, pp. 325–341.

[3] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[4] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.

[5] 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 *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[6] P. Mirowski, M. Grimes, M. Malinowski, K. M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, A. Zisserman, R. Hadsell *et al.*, "Learning to navigate in cities without a map," in *Advances in Neural Information Processing Systems*, 2018, pp. 2419–2430.

[7] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 31–36.

[8] 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.

[9] O. Zhelo, J. Zhang, L. Tai, M. Liu, and W. Burgard, "Curiosity-driven exploration for mapless navigation with deep reinforcement learning," *arXiv preprint arXiv:1804.00456*, 2018.

[10] A. Kumar, S. Gupta, D. Fouhey, S. Levine, and J. Malik, "Visual memory for robust path following," in *Advances in Neural Information Processing Systems*, 2018, pp. 765–774.

[11] N. Hirose, F. Xia, R. Martín-Martín, A. Sadeghian, and S. Savarese, "Deep visual mpc-policy learning for navigation," *arXiv preprint arXiv:1903.02749*, 2019.

[12] T. Swedish and R. Raskar, "Deep visual teach and repeat on path networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1533–1542.

[13] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell, "Zero-shot visual imitation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 2050–2053.

[14] P. Furgale and T. D. Barfoot, "Visual teach and repeat for long-range rover autonomy," *Journal of Field Robotics*, vol. 27, no. 5, pp. 534–560, 2010.

[15] X. Wang, Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y.-F. Wang, W. Y. Wang, and L. Zhang, "Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6629–6638.

[16] K. M. Hermann, M. Malinowski, P. Mirowski, A. Banki-Horvath, K. Anderson, and R. Hadsell, "Learning to follow directions in street view," *arXiv preprint arXiv:1903.00401*, 2019.

[17] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," *arXiv preprint arXiv:1812.11941*, 2018.

[18] V. Mnih, N. Heess, A. Graves *et al.*, "Recurrent models of visual attention," in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[19] M. Malinowski, C. Doersch, A. Santoro, and P. Battaglia, "Learning visual question answering by bootstrapping hard attention," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 3–20.

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

[21] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2758–2766.

[22] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[23] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[24] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv preprint arXiv:1512.04455*, 2015.

[25] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 AAAI Fall Symposium Series*, 2015.

[26] A. Kumar, S. Gupta, and J. Malik, "Learning navigation subroutines by watching videos," *arXiv preprint arXiv:1905.12612*, 2019.