Robot in a China Shop: Using Reinforcement Learning for Location-Specific Navigation Behaviour

Bian Xihan and Oscar Mendez and Simon Hadfield

***Abstract*— Robots need to be able to work in multiple different environments. Even when performing similar tasks, different behaviour should be deployed to best fit the current environment. In this paper, We propose a new approach to navigation, where it is treated as a multi-task learning problem. This enables the robot to learn to behave differently in visual navigation tasks for different environments while also learning shared expertise across environments. We evaluated our approach in both simulated environments as well as real-world data. Our method allows our system to converge with a 26% reduction in training time, while also increasing accuracy.**

I. Introduction

Potential applications of modern robotics are becoming wider as the technology evolves. We are giving robots more tasks in more locations and allowing them to face more difficult challenges. Robotics has moved from vacuuming a room to delivering take-out food, from working the assembly line to managing an entire factory. As their functionality grows, so does the variety of their potential work environments. Currently, robots are designed to operate in a single class of work environment. It is feasible to treat a house as a single environment(although this is a simplification). However, it is no longer possible when the environment expands to an entire city. This limitation of the environment should be addressed promptly.

To relieve robots from the limitation of a singular environment, it is only natural to look at human behaviour for guidance. When entering a different environment, we often behave differently, and this behaviour change is not caused by the task at hand, but rather by the environment itself. When we walk into a china shop, we are slow and careful to avoid collisions as opposed to walking down an empty hallway which emphasises speed rather than precision. Robots need this ability to understand the environment they are in and change their behaviour accordingly. Additionally, we, as humans, compartmentalize our knowledge and memory so we can effectively work with only a small portion of them rather than everything all at once. We use this idea as further motivation for our multi-task learning approach.

In this paper, we define the combination of these abilities to be the problem of multi-environment navigation: A robot, operated by a single artificial intelligence model, should have the ability to recognize and navigate through multiple different types of environment. To approach this problem, we use visual navigation, as visual information (compared to other sensory inputs) will be most effective in learning the difference between different environments. In this paper, we focus on navigation from visual sensors,

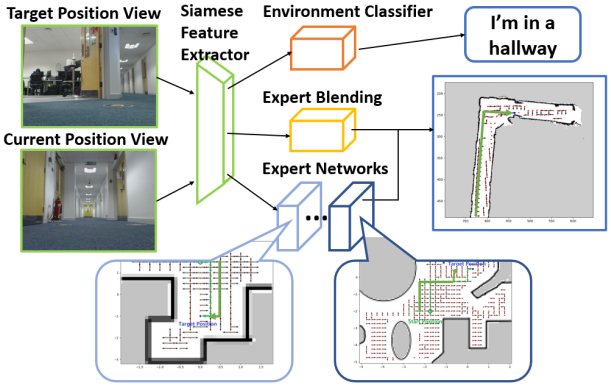


Fig. 1: The target position view and current position view are given to the agent, the environment classifier encourages the network to distinguish between environments in earlier layers, while multiple expert networks' outputs are blended to produce the final policy

and propose a new network architecture: Multi-environment Reinforcement-Learning in Navigation (MERLIN).

Visual navigation is a very active field of research, but most previous work focusses on improving navigation accuracy in a singular environment. In this work, we provide a new architecture where visual navigation in multiple environments can be achieved. We utilize a Siamese network feature extractor and multiple expert networks with attentive gating, combined with a special classification branch to encourage the network in recognizing the difference between environments. We find our model outperforms the state-of-the-art visual navigation models. Additionally, by encouraging the classification of environments, we are also able to achieve better results in learning performance, accelerating learning in multi-environment navigation.

In summary, we define a new problem of multi-environment navigation and propose a new model for approaching this problem. The main contributions of the paper are as follows:

1) Define the problem of multi-environment navigation.
2) Propose a new multi-task learning approach to visual navigation.
3) Propose the inclusion of an intermediate environment classification loss to accelerate learning.

II. Related Work

A. Visual Navigation

Visual Navigation relies on vision as the primary source of information for navigation, compared to other navigation

methods which rely on distance sensors. In map-based approaches for visual navigation, visual input is used mainly for landmark tracking. The algorithms detect landmarks on the camera input and track them in the following frames to determine the position of the robot [5, 7]. Another approach is to let the robot explore the environment and instead of a map, the robot builds a feature representation model of the environment [16].

In more recent studies, mapless and Artificial Intelligence (AI)-based navigation is becoming more popular. Mapless approaches (also known as visual odometry) [17], include two main solutions: Optical-flow and Appearance-based matching. Optical-flow-based solutions estimate the motion of objects by tracking the motion of features throughout a sequence of visual input [6,13]. Appearance-based matching solutions rely on prior knowledge of stored images of the environment [9]. The robot will try to match the current view with the stored images to locate and navigate [3, 22]. This idea of feature tracking to localization and navigation has been fundamental for AI-based localization and navigation. Notably, the work of Kendall in PoseNet [8], which uses a convolutional network for real-time camera relocalization where the model is trained on labelled images of the environments.

We approached the problem of multi-environment navigation through target-driven navigation, which is a branch of appearance-based matching solutions. This approach employs the use of deep Reinforcement Learning (RL) which does not require supervised training for landmarks or features. The work of Zhu et al. [22] presents a state-of-the-art (SOTA) target-driven visual navigation solution. The model they proposed is an actor-critic model which has a policy function of both the goal and the current state as input.

B. Reinforcement Learning

Schulman's 2015 work in Trust Region Policy Optimization (TRPO) is a classic RL algorithm that utilizes a different approach [19]. Trust region methods use a model function to estimate the objective function, and by optimizing the model function, perform better actions. The size of the policy update is constrained to monotonically decrease to ensure convergence. Schulman's later work in the Proximal Policy Optimization Algorithms (Policy Optimization Algorithm (PPO) and PPO2) improved upon TRPO [20]. These algorithms use a "surrogate" objective function to determine the next action while interacting with the environment. It assumes that with similar state input, the agent should take similar action, which lowers the rate and necessity of re-sampling. Instead of constricting the model functions, PPO applies a penalty to policies that differ from the objective function.

Another RL algorithm with a combined approach is the work of Mnih et al. in the Actor-Critic (AC) series of techniques [14]. These algorithms combine the Q-learning and policy gradient by creating a policy-based actor that chooses actions and a value-based critic which scores the actions. This allows the handling of both discrete and continuous problems, as well as updating more regularly for better learning efficiency. In the more advanced version: Asynchronous Advantage Actor-Critic (A3C), the algorithm allows asynchronously update to and get updated by the main policy by multiple agents, this advancement greatly increased computation and sampling efficiency, allows for faster training given the growing computing power. The multi-agent support also enabled RL in multi-task learning, allowing one agent to learn multiple tasks simultaneously. Due to its asynchronous nature, it is an ideal fit for multi-task learning by allowing agents to learn multiple tasks simultaneously, and update to a single main policy.

C. Multi-Task Learning

In multi-task reinforcement learning, numerous works have shown the capability of a single agent to perform at an expert level in multiple Atari games using deep Q-learning [2, 15]. In the RL environment, the model makes no assumption of the relatedness of tasks, which enabled many different approaches such as policy representation for each task and regionalized policy clustering, [11] which employs a hierarchical Bayesian approach to model the distribution over Gaussian process temporal-difference value functions for each task, and A3C-based deep reinforcement learning approach. The challenges these approaches all face are negative learning and scalability. Negative learning refers to when agents 'forget' previously learned knowledge while learning a new task. The most popular solution to negative learning is through the use of a gating mechanism, where the network is only allowed to update part of itself during the training for each task. This idea is first proposed by Rusu et al. in the work of Progressive Neural Network (PNN) [18], where the network freezes itself and add new resources for the new task. In the training process of the new task, the network stops the updating of its current weights, then widens all the nodes to provide new resources for learning the new knowledge. This approach does provide a good solution to the problem of negative learning. However, it fails in scalability, due to increasing learnable parameters and poor sampling efficiency with respect to the increasing complexity of tasks. To improve sampling efficiency, the work of Andrychowicz et al. [1] and Landolfi et al. [12] propose the use of a memory bank or model based approach during sampling. For parameter size, a common solution to this is using model compression techniques for data efficiency, such as the work of Teh et al. in Distral [21], where the network shares a distilled policy that captures the common behaviours across all tasks and allows workers to solve their own task while staying close to the main policy.

In this study, we develop a multi-branch gated network somewhat similar to PNN [18] but with soft blending and lifelong training of all branches training simultaneously on all tasks, instead of sequential training and freezing weights. This framework makes no assumption on the relatedness of tasks and mitigated negative learning effect through attention based soft gating.

III. METHODOLOGY

The objective of the agent is to be able to navigate in multiple types of environments. The agent can navigate the environment through actions: step forward, step backward, turn right and turn left (90 degrees). Given a target, the agent will only be given its current observation I_O and a view of the target I_t . A single agent should be capable of solving a set of similar navigation tasks placed in different environments by training a policy $\pi(a_t|s_t, \tau)$ and value function $V(s_t, \tau)$ for each task, while maximizing the reward for each task.

To tackle the specific problem of multi environment/task visual navigation, we propose a new architecture: MERLIN. Our architecture can be largely divided into 4 major components as shown in figure 2: A Siamese feature extractor, n sub-networks, an attention network and an environment classifier network. In the following section, each of these components will be described in turn.

A. Siamese Feature Extractor and Joint State Embedding

The network has 2 branches joined near the input to form a Siamese network. The expected functionality of the Siamese feature extractor is to capture the input state through feature extractors as well as to identify commonly useful information for all tasks and the feature characteristics to identify different tasks.

The Siamese feature extractor takes input in the form of a vector of the 2 images: the current agent observation I_C and the current target I_T. Each image will be fed into a different branch of the Siamese feature extractor network. Both the target state input and the current state input are first put through convolutional feature extractors E_f. The feature extractors share the same weights W_S between branches, and the output of each branch will then go through a normalization function N before concatenating into a state embedding. For the current state observation $O_C = (I_T, I_C)$ that contains the feature information from both input images, the features $F(s)$ is provided through the following equation:

$$F(s) = \{N(E_f(I_T|W_S)), N(E_f(I_C|W_S))\} \tag{1}$$

The Siamese feature extractors are updated by losses flowing through both the RC network branch and the sub-network/attentive network branch. This means the features are required to simultaneously be effective at solving the RL navigation task, and capable of distinguishing different categories of the environment.

B. Task-Specific Expert Policy Sub-Networks

The n sub-networks serve as the expert networks that learn the knowledge and skill to solve a specific task. The number n is determined by a range of factors including the number of tasks, the similarity between each task, the similarity between each environment, etc. In this work, the number n equals the number of environments. This can be considered as an optimization between requirements and availability of resources for the agent. The optimizer will learn to ignore excess sub-networks when given more resources than required as shown in the work of Bram [4].

The specific architecture of the sub-networks can be altered to fit the scenario. It is possible to have a variety of different expert networks that would work better for different tasks or task settings combined within the same agent. As we are working on the subject of visual navigation, the sub-network architecture in this study is designed for visual information processing and navigation tasks: A softmax layer maps the last hidden layer of each network to an A dimensional vector to produce action probabilities A_i and a linear layer outputs the value function V_i for each expert network i with a specific policy π_i:

$$A_i = softmax(\pi_i(E_i(F(s)|W_i)|W_A) \tag{2}$$

$$V_i = E_i(F(s)|W_i) \cdot W_v \tag{3}$$

Where E_i represents the RL network used for the actor-critic branch. If the action space is different between tasks, the size of A should be the largest action space size of all tasks.

C. Attentive Task Allocation and Soft Blending Network

The attentive task allocation network first takes the output and recognizes the corresponding expertise required by different tasks. While each expert sub-network produces a policy function, the attentive network assigns a distribution weight W_τ to these policy functions according to the estimated relevance of expertise.

$$W_\tau = softmax\left(\{Att(F(s)|W_{Att_i})|i \in \{0..n\}\}\right) \tag{4}$$

The softmax normalizes the weights W_{Att_i} to sum to 1. The final policy is then based on the dot product of the attention weights against the experts' action distributions:

$$\pi(\alpha|s) = \sum_{i=0}^{n} W_{\tau_i} \cdot A_i \tag{5}$$

where W_{τ_i} is the W_τ for the ith expert network. This policy will determine the action taken by the agent during each timestep. The attention weight W_τ is also used to compute a value function for reinforcement learning branches of the network. For the expert values $V_\tau = V_0, V_1 ... V_n$, the combined value function is:

$$V_{rl}(s) = \sum_{i=0}^{n} W_{\tau_i} \cdot V_i(s) \tag{6}$$

This value loss is used to update the expert networks, the attentive task allocation network, and the Siamese feature extractor, but not the RC networks.

D. Environment Classifier

Finally we propose an additional environment classifier, or Room Classifier (RC), network branch. This serves as a regularization that encourages the feature extractor layers to preserve the information that helps identify the tasks. The RC network updates only itself and the feature extractor layers with a loss function depending only on the current state:

$$L_{rc}(s) = 1 - \mathcal{RC}(F(s)|W_{rc}) \cdot \mathcal{M}_\tau \tag{7}$$

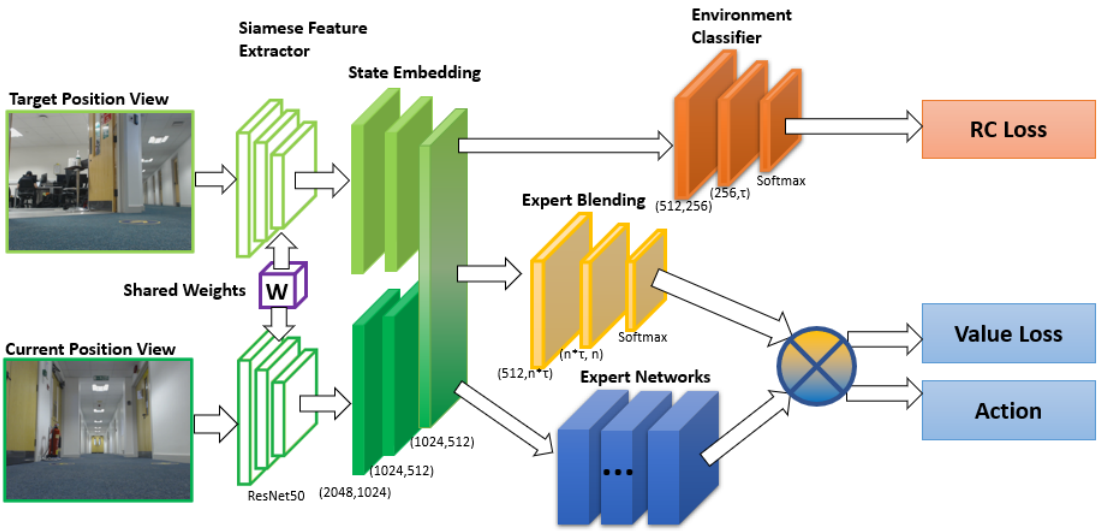


Fig. 2: Our architecture can be divided into 4 major components: A Siamese feature extractor (green), n sub-expert-networks (blue), attention network (yellow), and a environment classifier network (orange).

with $\mathcal{M}_\tau$ being a one-hot vector containing zeros everywhere, except the entry corresponding to the ground-truth task label which is 1. The RC loss function will be independent of the RL loss function, updating only the RC network and the Siamese feature extractor.

In this work, the RC network is trained through supervised classification methods, it may be possible in the future to extend the capability of the RC network to recognize a new task that the agents have never seen using unsupervised techniques.

IV. Experiments and Results

To evaluate our approach, we experimented with our agent's ability to perform visual navigation tasks in both simulated and real environments. The agent will be trained in multiple different themed environments simultaneously. The RL algorithm used in the experiments is a multi-thread A3C. The network backbone for the sub-networks are the same as the SOTA model [22]. The reward is inversely related to the length of the path taken by the agent to reach the target position: Negative reward accumulates with the path length l at a rate of p_{stp} per step. The agent also takes an additional penalty p_{cr} for the number n_{cr} when agent hitting obstacles. Each episode is limited to t_{lm} timesteps to deter reward hacking and avoid the agent getting stuck. Reaching the target position before this limit will result in a small positive reward r_{ter}. Exceeding this limit will result in ending the episode and a large negative reward p_{ter}. The final reward R is given by:

$$R = -p_{stp} * l - p_{cr} * n_{cr} + (r_{ter} | l \leq t_{lm}) - (p_{ter} | l > t_{lm}) \quad (8)$$

The agent will receive an image of the target position I_T and the current position's view I_C. In simulation a small amount of random noise is added to the current position before sampling I_C to ensure generalization.

Our agent is implemented with Pytorch and trained on Nvidia Geforce GPU servers. The evaluation is done in several experiments both in simulation and the real-world.

A. Experiments

We prepared both simulated and real-world environments for the experiments. To create the simulated environments, we used the 3D simulation environment AI2-THOR [10]. AI2-THOR is a 3D simulation program used for machine learning. The simulated environment consist of themed rooms such as living room, bathroom, kitchen, etc. We created the regularly sampled environments by allowing the agent to move forward and backwards as well as turning 90 degrees on a square grid. The agent will be dropped randomly into the environment and given a random target which is reachable from the starting position. The square grid $g_i \in G$ has a grid size of α with g_i being a sample from SO(2) (i.e. comprising of x, y, θ) The target is described through the view of the agent I_T at the target position $g_T = G(x_T, y_T, \theta_T)$. The current position $g_C = G(x_C, y_C, \theta_C)$ of the agent is given through a view of the agent at its current position I_C. To ensure the generalization capability of the agent and simulate the navigation error of an actual robot, the current view is sampled randomly near each grid point. The random sample position is selected by adding Gaussian noise proportional to the grid size $\Delta g \sim \mathcal{N}(0, (\alpha * \rho)^2)$ to the x and y coordinate of the original state position. Resulting in the view at $g_{sample} = G(x + \Delta x, y + \Delta y, \theta)$ as the actual input to the agent.

The agent will then try to find the shortest path to reach the target position. In our experiments, 4 environments are trained simultaneously, each using the same amount of computing resources, the average episode length and percentage of successful runs across all 4 environments are used to evaluate their performance.

To train the agent in the real-world environments, we collected real-world data using a Turtlebot from various locations with different themes such as a hallway, living room, common room, etc. As the robot already has drifting

Sim Dataset	Env1	Env2	Env3	Env4	Ep. Length	Avg. reach goal	RC Accuracy	Converge Step
MERLIN	15.83	**17.32**	**8.74**	**7.71**	**12.38**	**99.50**	99.61	23K
Expert1	**13.14**	200.00	199.22	197.64	152.31	25.40	0.00	6K
Expert2	196.85	24.48	197.62	194.54	153.20	26.00	0.00	10K
Expert3	200.00	199.21	20.25	195.38	153.53	24.60	0.00	5K
Expert4	198.44	199.20	197.68	8.73	150.82	25.70	0.00	7K
Joint Expert	15.53	21.32	11.49	8.96	14.30	99.30	0.00	31K

TABLE I: Navigation task step count and success percentage in the simulated dataset

Real Dataset	Env5	Env6	Env7	Ep. Length	Avg. reach goal	RC Accuracy	Converge Step
MERLIN	**10.82**	40.27	**25.04**	**25.35**	**94.60**	98.93	19K
Expert1	13.80	194.72	198.80	135.65	33.50	0.00	4K
Expert2	199.40	32.31	198.81	143.39	30.50	0.00	20K
Expert3	198.80	198.82	31.31	142.87	31.00	0.00	6K
Joint Expert	18.05	**21.94**	38.86	26.28	94.10	0.00	25K

TABLE II: Navigation task step count and success percentage in the real-world dataset

errors, there's no random sampling used in the real-world datasets. Using these images, we produced gridded real-world environments similar to the simulated environments which can be trained off-line. A total of 3 different real-world environments were used for training: A residential living room, a university common room, and an office corridor. They represent 3 types of spaces: a small enclosed space with few obstacles, a large open space with many obstacles, and a narrow enclosed space without obstacles. The agent is expected to behave differently in each of these types of space.

In the last section of the experiments, we improve on this and demonstrate our model directly on a live robot. To achieve this, a Turtlebot is used to navigate in the previously trained environments. A video showing this experiment can be found at this link.

B. Baseline comparison

We first compare our agent with the SOTA target-driven navigation model [22] and perform an ablation study of our technique. The SOTA model is trained in different environments both separately (referred to as Expert1 to Expert4) and concurrently (referred to as "Joint Expert"). All models are trained until they converge to over 99% success rate, the average episode length and the number of time steps taken to converge are recorded.

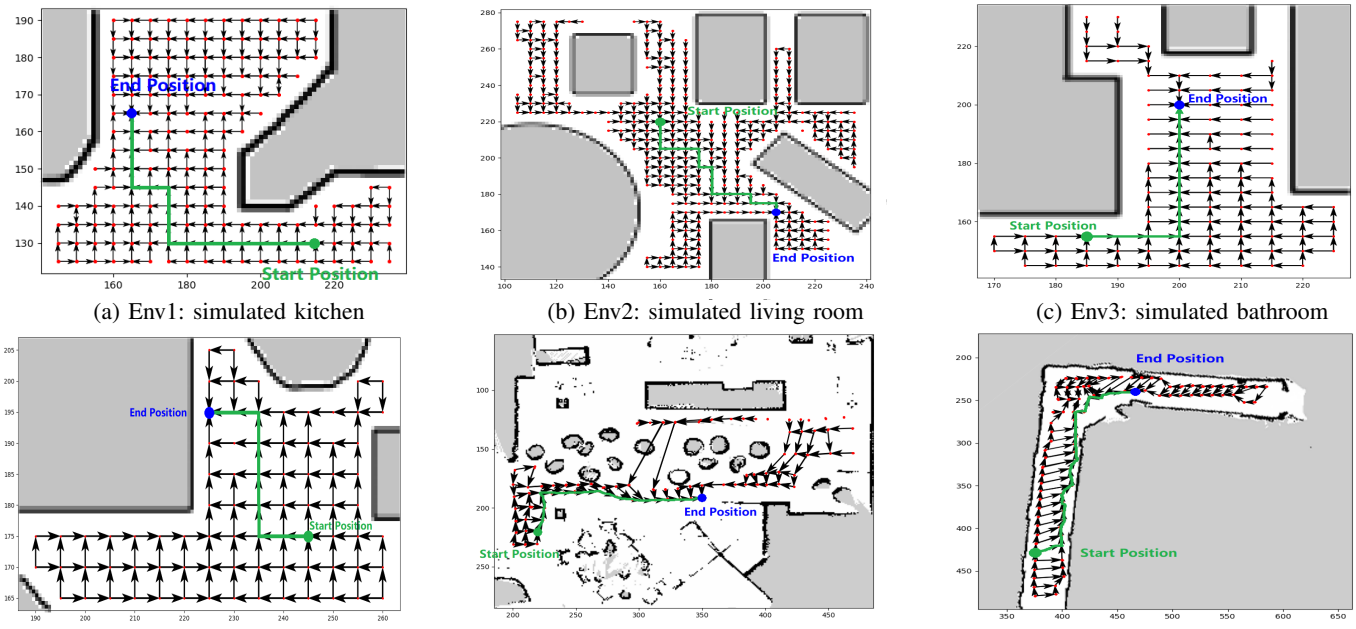
As shown in both Table I and II, the separately trained SOTA models show an inability to perform navigation task in any environment other than the one it was last trained on. The jointly trained baseline can complete the tasks in all different environments, but it requires a 30% longer training time and has a lower performance than our proposed technique. Additionally, our agent is able to out-perform the specialist experts in most of their corresponding environments. This suggests that there is a sharing of expertise between environments, which can take advantage of our expertise-blending approach. In the simulated kitchen, as shown in Figure 3b and Table I, it is a larger environment with more grid points compared to other environments. The joint expert performs more poorly in this environment compared to its performance in the rest of the environments. A similar pattern can be observed in the real-world results as shown in Table II, the joint expert has a particularly low performance in the largest environment Env7. It appears that the improvements offered by the proposed approach scales with the number and the size of the environments. The MERLIN model also has a 24%-26% faster converge speed. This indicates that when the number of learnable parameters increases, it is possible that our approach will still be able to converge on more difficult tasks when the joint expert cannot.

C. Qualitative Multi-environment behaviours

In Figure 3, we provide examples of the behaviours of the agent in different environments. The agent has different behaviour when dropped to a random position in each environment. The vector fields are formed by examining the trajectories from all possible starting positions to the target, and the green trajectory shows one complete example trajectory. In open spaces such as the simulated living room 3b and the simulated kitchen 3a, the agent tends to make strides and turns for localization. In narrow spaces such as the simulated bathroom 3c the agent would prioritize moving away from walls. In the real corridor 3f the agent would have much less turning actions during the narrow hallway but shows turning behaviours in the middle section where the space is relatively open.

D. Generalization and Noise Resilience

In the third experiment, the time limit is tightened for completing the episode, and the noise ratio for current view sampling is increased. The sampling method is also changed from Gaussian noise to a uniform noise scaling with the grid size to increase difficulty. A simple size of 50 is used for each noise level, and the agent will perform 1000 episodes across the 4 simulated environments. As shown in Figure 4, MERLIN outperforms the joint expert consistently and maintains more than 80% success rate until the noise level reaches 100% of the grid size. A drop in performance occurs around 50% noise level. This is due to the possible sampling positions of each grid starting to overlap with each other. At 100% noise level, the sampling position can drift to another state's position. These results indicate the MERLIN agent has a good generalization ability within each environment and is not over-fitting to the training environment. This also indicates that an agent trained off-line in a gridded version of a real-world dataset could potentially be transferred to operate in the real world.



(a) Env1: simulated kitchen (b) Env2: simulated living room (c) Env3: simulated bathroom

(d) Env4: simulated tiny bathroom (e) Env6: real common room (f) Env7: real corridor

Fig. 3: Agent makes large strides before turning in open spaces, and avoiding walls when in narrow spaces

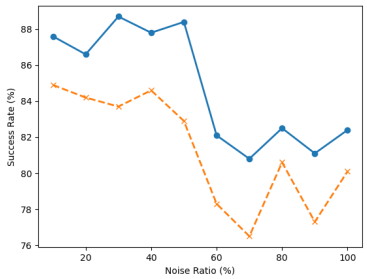


Fig. 4: Success Rate drops with increasing noise ratio in current view sampling. The blue line indicates MERLIN's performance, the orange line indicates the joint expert.

E. Live Demonstration

In the last experiment, the live demo shows the Turtlebot performs a target-driven navigation task in the real-world location of Env7 (real corridor) using the MERLIN model (Link to video). The robot successfully completed the task with only a few missteps outside the optimal path, likely caused by lagging and drifting errors. We can also observe the robot making a straight line in the hallway, but turning and making strides in the relatively open area in the middle. Compared to the gridded simulated environments, the robot spends more time turning for localization and attempted correction to drifting errors. As the model is trained in discrete environments, it has a natural deficiency in handling inaccurate turning angles, this could potentially be improved by generalizing over rotation during training. The model's solution to this is to keep turning until it finds a recognizable direction. However, this strategy has difficulties under excessive lagging and drifting error.

V. CONCLUSIONS

We have introduced the multi-environment navigation problem in the field of robotic navigation and proposed a multi-task deep reinforcement learning framework to approach this problem through visual navigation. Overall, the MERLIN model outperforms the SOTA model in the multi-environment target-driven navigation tasks in both performance and training speed. Interestingly, MERLIN also outperforms specialist single-environment expert networks even on their own training environment. We observed different behaviours depending on the surroundings in both the simulated environments and real-world environments. We also demonstrated the model's ability in operating in the real world even when trained off-line in discrete environments.

It is foreseeable that upcoming robots will require more multi-tasking capability than navigation in multiple different environments. They may also need adaptive skills for undertaking various non-navigation tasks in a variety of locations. Mimicking humans ability to adapt to environments is going to be vital for robots and provides a great challenge for the field of robotics and artificial intelligence. As for future work, it may prove useful to focus on the robot's capability to transition smoothly between expert networks over time. Another important topic is allowing the robot to adjust its behaviour to adapt to different network resource allocation, in relevance to lifelong learning. As well as the robot's ability of understanding a complex task and break them down to multiple simpler, more manageable tasks each with an independent expert network.

ACKNOWLEDGMENT

This work was partially supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant agreement EP/S035761/1 and Innovate UK Autonomous Valet Parking Project (Grant No 104273).

REFERENCES

[1] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Neural Information Processing Systems(NIPS), 2017.

[2] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In International Conference on Machine Learning, pages 507–517. PMLR, 2020.

[3] Shi Bai, Fanfei Chen, and Brendan Englot. Toward autonomous mapping and exploration for mobile robots through deep supervised learning. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2379–2384. IEEE, 2017.

[4] Timo Bräm, Gino Brunner, Oliver Richter, and Roger Wattenhofer. Attentive multi-task deep reinforcement learning. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 134–149. Springer, 2019.

[5] Masayoshi Hashima, Fumi Hasegawa, Shinji Kanda, Tsugito Maruyama, and Takashi Uchiyama. Localization and obstacle detection for robots for carrying food trays. In 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), volume 1, pages 345–351. IEEE, 1997.

[6] Berthold KP Horn and Brian G Schunck. Determining optical flow. Artificial intelligence, 17(1-3):185–203, 1981.

[7] Mansur Kabuka and A Arenas. Position verification of a mobile robot using standard pattern. IEEE Journal on Robotics and Automation, 3(6):505–516, 1987.

[8] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In 2015 Proceedings of the IEEE international conference on computer vision (ICCV), pages 2938–2946, 2015.

[9] Ye-Hoon Kim, Jun-Ik Jang, and Sojung Yun. End-to-end deep learning for autonomous navigation of mobile robot. In 2018 IEEE International Conference on Consumer Electronics (ICCE), pages 1–6. IEEE, 2018.

[10] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. arXiv preprint arXiv:1712.05474, 2017.

[11] Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In Neural Information Processing Systems(NIPS), 2016.

[12] Nicholas C Landolfi, Garrett Thomas, and Tengyu Ma. A model-based approach for sample-efficient multi-task reinforcement learning. arXiv preprint arXiv:1907.04964, 2019.

[13] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In Imaging Understanding Workshop, pages 121–130. Vancouver, British Columbia, 1981.

[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937. PMLR, 2016.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[16] Hans P Moravec. The stanford cart and the cmu rover. Proceedings of the IEEE, 71(7):872–884, 1983.

[17] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), volume 1, pages I–I. Ieee, 2004.

[18] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. arXiv preprint arXiv:1606.04671, 2016.

[19] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In International conference on machine learning, pages 1889–1897. PMLR, 2015.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[21] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In Neural Information Processing Systems(NIPS), 2017.

[22] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In 2017 IEEE international conference on robotics and automation (ICRA), pages 3357–3364. IEEE, 2017.