NavRep: Unsupervised Representations for Reinforcement Learning of Robot Navigation in Dynamic Human Environments

Daniel Dugas, Juan Nieto, Roland Siegwart and Jen Jen Chung

Abstract— Robot navigation is a task where reinforcement learning approaches are still unable to compete with traditional path planning. State-of-the-art methods differ in small ways, and do not all provide reproducible, openly available implementations. This makes comparing methods a challenge. Recent research has shown that unsupervised learning methods can scale impressively, and be leveraged to solve difficult problems. In this work, we design ways in which unsupervised learning can be used to assist reinforcement learning for robot navigation. We train two end-to-end, and 18 unsupervised-learning-based architectures, and compare them, along with existing approaches, in unseen test cases. We demonstrate our approach working on a real life robot. Our results show that unsupervised learning methods are competitive with end-to-end methods. We also highlight the importance of various components such as input representation, predictive unsupervised learning, and latent features. We make all our models publicly available, as well as training and testing environments, and tools[1]. This release also includes OpenAI-gym-compatible environments designed to emulate the training conditions described by other papers, with as much fidelity as possible. Our hope is that this helps in bringing together the field of RL for robot navigation, and allows meaningful comparisons across state-of-the-art methods.

I. INTRODUCTION

Robot navigation in complex environments and in the presence of humans is a challenging problem due to complexity in human behavior and the unpredictability of unstructured environments (see Fig. 1). Though popular planning approaches work well in many scenarios, edge cases complex enough to cause planning failures abound, and the engineering work which is necessary to address one edge case does not necessarily translate to another. As a result, designing a planner that is able to perform in any environment is still a monumental engineering task.

Outside of robot navigation, reinforcement learning (RL) approaches have led data-driven controllers to reach expert and even super-human performance at game-play tasks [1]. Data-driven methods, at least in theory, present a potential of unlimited scale, where a planner can be improved indefinitely so long as more data can be gathered. For this reason, several research efforts have focused on the use of data driven-methods for robot navigation [2–4]. Yet, the question remains of whether similar performance can be achieved for the problem of navigation among humans, and how.

[1]https://github.com/ethz-asl/navrep
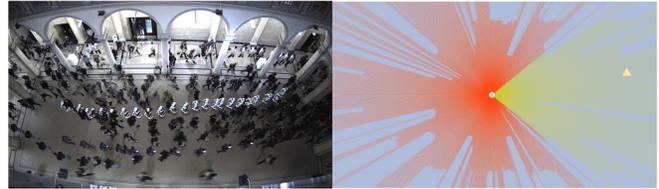
Fig. 1. Timelapse of an example target environment for autonomous navigation, snapshot of the corresponding LiDAR measurements on the right

a) b) c) d)
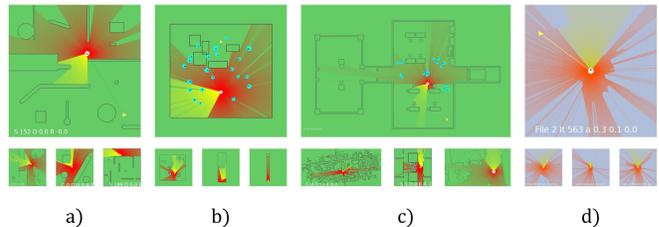
Fig. 2. We make NavRepSim openly available, a simulator which contains efficient, gym-compatible environments for end-to-end navigation, allowing all to reproduce scenes from (a) [2, 3], (b) CADRL and SOADRL [5–7], CrowdMove [4] (not shown), (c) IAN [8], and (d) real data.

When planning in the presence of humans, it is often necessary to provide estimates for the position and state of humans around the robot [5–7, 9, 10]. To make planning tractable, these estimates are typically reduced to position, velocity, and radius of the whole human. Subtleties in how individual legs move in LiDAR, or full body cues in 3D/camera measurements, are therefore lost. However, with end-to-end learning, since the data-driven model has access to the raw sensor input, it could theoretically infer actions based on such subtle information modalities.

For end-to-end learning, having to train the entire perception component of a policy from sparse rewards can be inefficient. An alternative is to use features from perception models which have been extensively trained with supervised training. This however, requires large quantities of labeled data. Using unsupervised training, it is possible to learn rich encodings from unlabeled data. These rich encodings can be shared across tasks. In addition, these unsupervised models can be used to generate synthetic reproductions of the world, which can assist or replace simulation. The question is, using current methods, can we train unsupervised representations which are practical for robot navigation?

We address this through the following contributions:

- We provide an open-source simulation environment, trained models, and useful tools for future end-to-end navigation benchmarking (see Fig. 2).
- We design several unsupervised learning architectures for the task of robot navigation among humans.

- We compare existing approaches to each other and to ours, in seen and unseen environments, and demonstrate our approach on a real robot.

II. Related Work

Several state-of-the-art RL methods for robot navigation make use of exact pedestrian positions in the input: CADRL and following works [5, 9, 10] trained an RL-based policy to avoid simulated agents with no static obstacles, though real obstacles are handled out-of-policy in real testing. SARL [6] target the same domain as CADRL, with an architecture that focuses on pairwise interactions in order to model a useful value function. SOADRL [7] extends the SARL architecture to also take into accounts static obstacles, in some cases with a limited field of view.

End-to-end methods avoid the need for detection and tracking of humans by using sensor data as input directly. Pfeiffer et al. [2] explore using end-to-end supervised learning for robot navigation given 2D LiDAR measurements. Example trajectories are taken from an expert planner. In [3] imitation learning is used, followed by end-to-end RL, allowing for better performance and harder navigation problems. Both [2, 3] have a static environment, with no dynamic agents. CrowdMove [4] shows a robot navigating in a crowded space using an end-to-end RL policy. The learning agent is trained to solve a joint planning problem, where multiple instances of itself all simultaneously navigate past each other and obstacles to reach their goal.

Ha and Schmidhuber [11] raise the question of whether unsupervised learning can assist end-to-end RL. Their world models approach is able to beat state-of-the-art performance on the CarRacing openAI gym environment by splitting the end-to-end task into an unsupervised latent feature learning task and a control policy learning from latent features RL task. A stated potential of this approach is the ability to use the unsupervised model to replace simulation, and they demonstrate a proof of concept for this idea on the Doom gym environment. Piergiovanni et al. [12] applies this "dream" idea to real-world robot navigation, in the limited scope of navigating to or avoiding an object based on camera images. GameGAN [13] pushes the neural simulation concept further by approximating the pac-man game through unsupervised learning of a generative adversarial network (GAN) architecture. Recent work [14–16], also demonstrated that the unsupervised learning of sequences can deliver impressive results simply through increase in scale. They demonstrate state-of-the-art performance in various natural language processing tasks, using their approach.

III. Navigation Simulation Environment

The NavRepSim environment is designed with RL applications in mind. It aims to apply to any range-based sensor navigation problem. The goal of open-sourcing this simulator is to make it easier for anyone to reproduce state-of-the-art solutions for learning-based navigation.

The simulator offers 23 fixed maps as well as the possibility to procedurally generate maps containing random
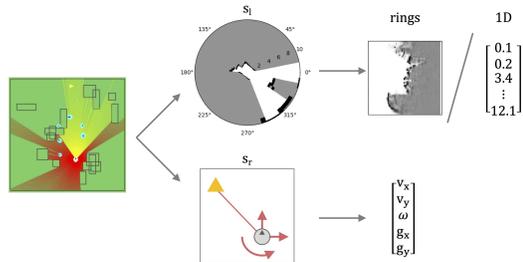
Fig. 3. The state representation obtained from the world (left) combines both the LiDAR observation (s_l), and goal/velocity observation (s_r). The LiDAR observation is either represented as a 64×64 matrix of occupancy probability (rings) or a 1080 vector of distances (1D).

polygons (see Fig. 2 for examples of the available maps). The offered maps were designed to imitate those used in prior work. Note that pre-made gym environments are made available to reproduce the specific training domains of other paper. We also created several new maps from real LiDAR data. Overall, we provide environments that range from simplified and highly synthetic to more realistic maps containing real-world sensor noise.

In addition to having different maps, the learning environments can also vary in terms of the simulated robot(s), the simulated human agents as well as the learning curriculum. The simulated robot can be either holonomic or differential-drive. The inertial physics of the robot base can be simulated with high fidelity or simplified as instantaneous velocities. Further, collisions can either lead to damage (reflected as a negative reward) or instant episode termination. Users can also specify the number of robots in the environment, since joint training of several robots in the same simulation is also possible. The number and behavior of the simulated human agents can also be adjusted. Agents in the environment may be static or dynamic, they may be rendered as moving legs or ellipses, and they can be controlled using different policies (constant velocity, ORCA [17], global planning). Finally, we offer two learning curricula. The first uses a constant episode set-up, picking different maps and agent locations for every episode, while the second varies the environment difficulty (more human agents and more obstacles in the case of procedurally generated maps) based on the policy's success.

Care is taken to maintain simulation efficiency, with most environments running at more than 100 iterations per second in a single thread (on a laptop, Intel Core i7-6600U CPU @ 2.60GHz x 4). For comparison, the CarRacing gym environment used in [11] runs at the same speed on the same machine.

IV. Learning-based Navigation

A core purpose of this work is to study the possible variants of learning-based robot navigation. The basic setup common across many state-of-the-art methods is a ground robot equipped with a range sensor (often a 2D planar LiDAR). The state space used in the learning formulation is derived from the incoming range data. As discussed in Section II, several works also assume that live pedestrian tracks are available to the robot, however in our work we do not
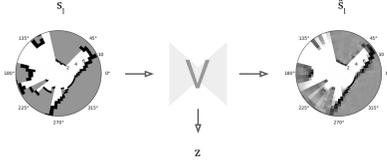
Fig. 4. Visualization of the V auto-encoding module's inputs and outputs. The LiDAR observation s_l and reconstruction $\hat{s}_l$ can be either in the rings or 1D representation.

make this assumption. The robot action space is continuous and is represented as a vector in $\mathbb{R}^3$, $\mathbf{a} = [v_x, v_y, \omega]$. Each of $v_x, v_y, \omega \in [-1, 1]$ correspond to the robot's translational and rotational velocity in its base-link frame. Here we consider a holonomic robot, however the work can be extended to diff-drive robots by simply requiring $v_y = 0$.

The scalar reward function for time-step t is $r^t = r_s^t + r_c^t + r_d^t + r_p^t$. The first term, r_s^t, is the success reward, which is 100 if the goal is reached and 0 otherwise. The collision reward, r_c^t, is -25 if the robot collides and 0 otherwise. The danger reward, r_d^t, is equal to -1 if the robot is closer than 0.2m to an obstacle or agent, and 0 otherwise. Finally, r_p^t is the progress reward, which is equal to the difference between the previous distance to goal and new distance to goal.

A. State space variants

The state representation s^t is a combination of the LiDAR observations $\boldsymbol{s_l^t}$ and the robot velocity and goal position in the robot frame $\boldsymbol{s_r^t}$ (see Fig. 3). In this work, we consider two methods for representing LiDAR measurements and compare their effects when used as inputs to a learned navigation policy. The first representation is simply a 1-dimensional vector of the 1080 raw distance values from the LiDAR normalized to $[0, 1]$. We refer to this as the **1D** representation. We also consider a probabilistic representation for the input LiDAR data. This representation maps the space around the robot as a polar grid ($\mathbb{R}^{64 \times 64}$), with evenly distributed discrete angular sections and exponentially distributed discrete radius intervals, leading to a greater resolution closer to the robot compared to regions that are farther away. Given the LiDAR scan, each grid cell is assigned a value $p \in \{0, 0.5, 1\}$ for being unoccupied, unknown or occupied, respectively. We refer to this two-dimensional probabilistic LiDAR representation as the **rings** representation.

B. NavRep: Unsupervised representations for navigation

In contrast to end-to-end learning approaches, NavRep exploits unsupervised representations. Following the approach of [11], we make use of three main modules,

- **V** (LiDAR encoding): a variational auto-encoder (VAE) trained to reconstruct the LiDAR state (Fig. 4),
- **M** (prediction): a sequence-to-sequence model trained to predict future state (Fig. 5),
- **C** (controller): a 2-layer fully connected network (FCN) trained to navigate using s_r and the latent representations of V and M, as input. Its output is the probability of taking action a.

For brevity, we refer to the latent encoding of V as z, and the latent encoding of M as h.
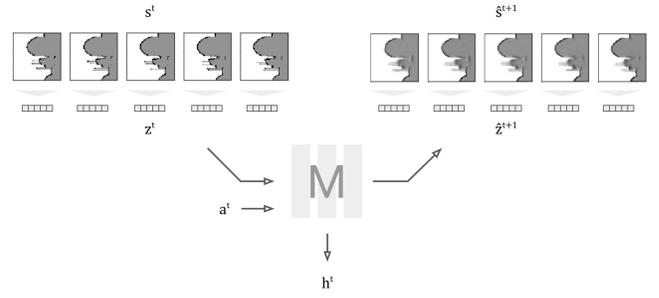
Fig. 5. Visualization of the M prediction module's inputs and outputs. The z latent features for each observation in the sequence are first encoded by the V module. These are passed to M along with the sequence of actions a. Each predicted $\hat{z}^{t+1}$ in the sequence is then decoded by V.
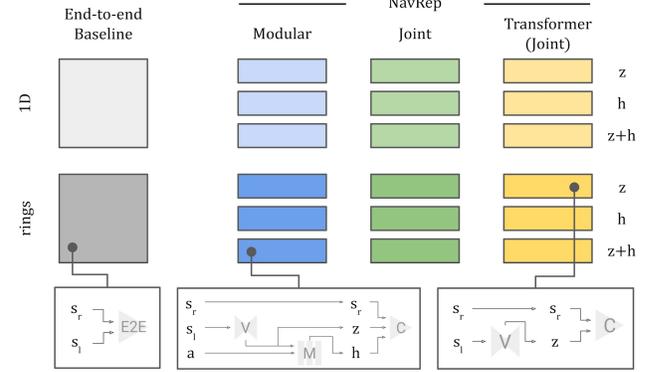
Fig. 6. Overview of the architectures trained in this work. For all approaches there are 2 LiDAR input variants. In addition, the 3 unsupervised-learning-based approaches each have 3 latent feature variants. Simplified schema of the components are detailed for 3 arbitrarily selected variants.

1) Modular training: From this set of modules, we offer our first NavRep learning variant. The V, M, and C modules are all trained separately, an LSTM is used for M, as in [11].

2) Joint training: Our second NavRep learning variant is designed to provide insight into the effect of training V and M jointly or separately. In [11], the authors first train V to minimize reconstruction loss before training M to minimize the z prediction loss while keeping the V network weights fixed. Here we compare their **modular** training regime with one that trains V and M **jointly**. An LSTM is used for M.

3) Transformer architecture: Rather than an LSTM, we use the attention-based **Transformer** architecture [18] in M to generate predictions of the future LiDAR encodings.

4) Input features to C: We consider the effect of the inputs used by the controller to compute the robot action, splitting each NavRep variant into three. We compare between using **z**, **h** or **[z, h]**. An overview of all the learning architecture and training variants that we study is provided in Fig. 6.

C. Network implementation and training

1) Hyperparameters: All V modules are composed of a 4-layer convolutional neural network (CNN) encoder, followed by 2-layer FCN, and 4-layer CNN decoding blocks. z latent features are of size 32, h latent features are of size 64. The LSTM-based M module has 512-cells. The Transformer-based M module is composed of 8 causal-self-attention blocks with 8 attention heads each. The modular V and M modules have respectively 4.3 million and 1.3 million

trainable parameters. The V+M transformer model has 4.8 million trainable parameters. More details available in the open-source implementation.

2) V: was trained using minibatch gradient descent, with a typical VAE loss,

$$L_V = L_{rec} + KL(z^t),$$

where $KL(z^t)$ is the Kullback-Leibler divergence between the latent feature distribution parameterized by the encoder and the prior latent feature distribution. The reconstruction loss L_{rec} is the mean squared error between input and output,

$$L_{rec} = H(\hat{s}^t, s^t), \tag{1}$$

where H denotes the binary cross entropy, s^t denotes the observed state (LiDAR representation) at time t in the sequence.

3) M: was trained on batches of sequences. We define the loss function as the latent prediction loss,

$$L_M = H(\hat{z}^{t+1}, z^{t+1}) + H(\hat{s_r}^{t+1}, s_r^{t+1}). \tag{2}$$

This adds the binary cross entropy loss between the predicted ($\hat{z}^{t+1}$) and the true (z^{t+1}) latent features to that of the predicted ($\hat{s_r}^{t+1}$) and the true (s_r^{t+1}) goal-velocity states.

4) Joint V+M: was trained with a loss function composed of the sum of the reconstruction and prediction losses from (1) and (2). However in this case, for the rings variant the prediction loss is

$$L_{V+M} = H(\hat{s}^{t+1}, s^{t+1}),$$

the binary cross entropy loss between the reconstructed state prediction $\hat{s}^{t+1} = V(\hat{z}^{t+1})$ and true next state s^{t+1}. For the 1D LiDAR variant we used the mean squared error instead of the cross-entropy loss as the predictions are not distributions, but direct values.

5) C: was trained using the proximal policy optimization (PPO) algorithm [19, 20]. Each C module was trained up to 3 times with varying random seeds in order to mitigate initialization sensitivity in the results. Training was stopped after 60 Million training steps.

6) Training data: The V and M modules were trained with data extracted from the same environment in which the C module was later trained. To generate training sequences, the ORCA [17] policy was used to control the robot, and the number of agents in the scene was increased to ensure sufficient observations of dynamic objects and prevent imbalance in the dataset. A simple model of leg movement was used to render dynamic agents as moving legs in the generated LiDAR data and a time-step of 0.2s was used for updating the agents' positions.

The C module itself was trained in the SOADRL-like environment (shown in Fig. 2), with randomly generated polygons and agents (every episode is unique). A bounding obstacle had to be added to prevent the policy from learning a simple risk-averse heuristic which consists in going around the entire area. During training, actions for the C module were limited to x-y movement, with 0 rotation. However, the robot's initial angular position was randomized for each episode.

Curriculum learning was used to improve the training of the policy: every failed episode lead to a lower number of
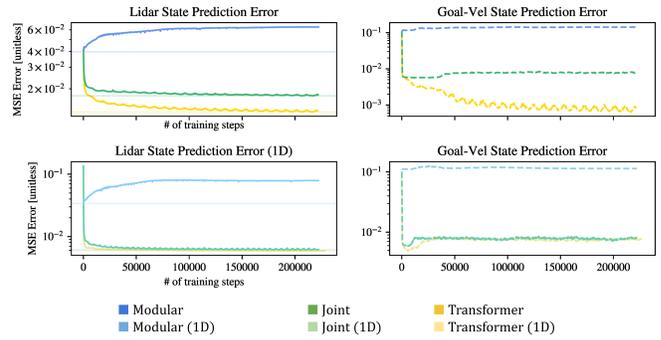
Fig. 7. Comparison of the *worldmodel error* for the joint, modular and transformer architectures, for their (top) rings LiDAR state representation variant, and ((bottom) 1D variant. This error is calculated and shown separately for the LiDAR state s_l (left) and goal-velocity state s_r (right).

polygon obstacles and agents, conversely, every successful episode lead to an increase. The maximum number of agents and polygons was capped to 5 agents and 10 obstacles.

D. End-to-end learning baselines

As a baseline, we implement a typical end-to-end RL approach. This approach attempts to learn a navigation policy $\pi(a, s)$ predicting action probabilities directly from LiDAR inputs.

Two variants of end-to-end learning are implemented: the first taking the 1-dimensional LiDAR representation as the state input, and the second taking 2-dimensional rings representation as the state input. For the value and policy model, we use a CNN for feature extraction, followed by fully connected layers. To make comparisons fair, the same CNN architecture and dimensions are used as in the V and C models described in the following subsection.

V. EXPERIMENTS AND RESULTS

A. Worldmodel error for unsupervised learning architectures

The six joint, modular and transformer architectures (Fig. 6) have differing loss functions, which makes it difficult to compare their losses during training. To this end, we define the "worldmodel error", a next-step prediction error term that allows meaningful comparison across architectures,

$$E_{worldmodel} = MSE(\hat{s}^{t+1}, s^{t+1}).$$

However, this error function does not produce comparable results for the rings vs. 1d LiDAR inputs, as they do not have the same dimensions and scale. Instead, these comparisons are shown separately in Fig. 7. The worldmodel errors shown here are calculated for the training sets, which compares each architecture's ability to learn in an unsupervised manner. It can also be understood as the accuracy of 'Dreams' generated by each architecture. In our results (Fig. 7) modular outperforms joint training, and Transformer outperforms LSTM.

B. Controller training performance

Though the training environment features curriculum learning through adaptive difficulty, a separate validation environment with fixed difficulty is used to quantify the absolute progress of each model every 100'000 training steps. This validation is similar to the training environment, except
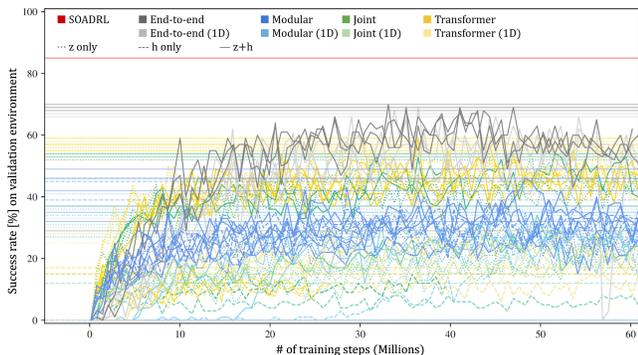
Fig. 8. 3 training sessions were performed for each of the two end-to-end and 18 NavRep architectures to account for the influence of network initialization on the results (legend as in Fig. 7). Horizontal lines mark the maximum success rate of each curve. The SOADRL success rate allows for comparison with methods that use exact human locations as input.

that it remains on the maximum difficulty, i.e. maximum number of agents and obstacles, and that it consists of 100 episodes with arbitrary but deterministic initializations. Success rates (number of validation runs in which the robot reaches the goal without collision) are shown in Fig. 8.

1) Discussion: Looking at the modular, joint and transformer models, it seems that the worldmodel error and learned controller performance are correlated. This suggests that better reconstruction and prediction performance in the V and M modules leads to latent features which are more useful for the control task. Nevertheless, it can also be seen that the end-to-end approaches perform better in the validation tasks than the NavRep controllers. Finally, none of the models are able to surpass the performance of SOADRL, which has access to exact positions of pedestrians.

C. Test performance

Each model was tested in 27 unseen scenarios (3 maps, 9 scenarios each). At least 300 episodes were run in each scenario, for a total of 10'000 test episodes per model. These scenarios are designed to represent a wide range of possible real-life situations that can help to identify specific issues with the learned controller, such as global planning, agent avoidance and crowd aversion. For each map, the first 6 scenarios are taken from [8], and the remaining 3 are (7) a trivial scenario where the robot is close to the goal with no obstacle in between, (8) an easy scenario with a close goal, and a single obstacle between robot and goal, and (9) a typical dynamic avoidance task where all agents are placed on a small circle with goals on the opposite side and no obstacles in between. The scenario definitions and selection was fixed before any testing took place to avoid bias.

1) Discussion: In general it can be seen that the end-to-end models perform suffer a greater drop in performance from training to testing, whereas the NavRep models have more consistent performance. In particular, the 1D LiDAR end-to-end model fails completely in the realistic map even though it is a top performer during training. In addition, when investigating the low performance of SOADRL, we saw that most failures were due to crashes into obstacles and not agents.
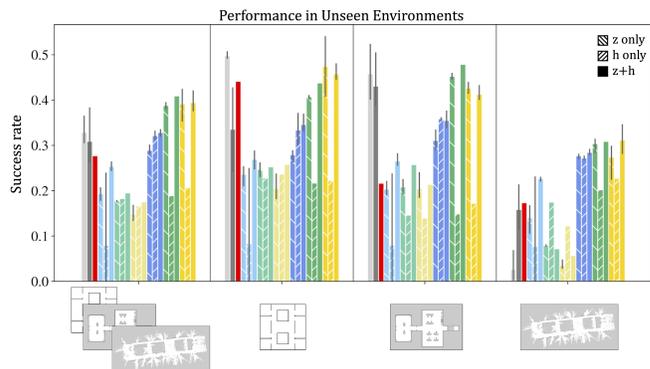
Fig. 9. Mean performance of each model in the testing environments. Legend as in Figs 7, 8, with error bars denoting the minimum and maximum score across random seeds. Overall performance is shown (left), as well as per-map performance from the simple, complex, and realistic maps.
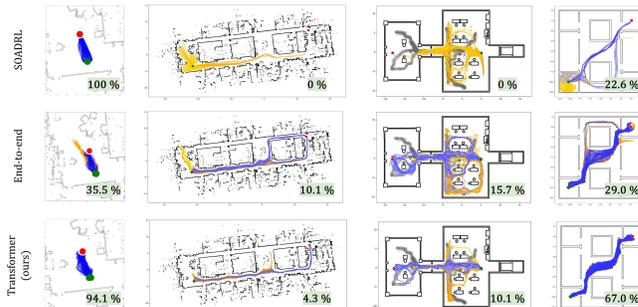
Fig. 10. Comparison of trajectories executed by the best performing models during testing in unseen environment. From left to right, the scenarios are 1) simplest scenario in realistic map, 2) global planning task in realistic map, 3) full planning task (global, static, dynamic avoidance) in complex map, 4) global planning in simple map. Orange trajectories are trajectories which resulted in failure to reach the goal, through collision or time-out. Blue trajectories are successful trajectories. Grey circles indicate the positions of dynamic agents. Overlaid is the percentage of successful trajectories.

In the same test environment, a traditional planner based on the timed elastic band (TEB) approach from the widely-used ROS navigation stack achieves 76% success rate.

The low success rates of all models on long-range navigation scenarios (Fig. 10) indicates that global navigation is a challenge for RL-based methods. We also see that several works [5, 7, 10] avoid this problem by dealing with local navigation only. Looking at other failures, we see that in several cases, the end-to-end and unsupervised models don't learn to proceed slowly in the presence of "danger" (other dynamic agents). Instead they move as fast as possible, or are stuck when available space becomes narrow. This can perhaps be improved through reward design. Finally, in the challenging dense static crowd scenario, though most attempts fail, a few are able to succeed by avoiding the dense area, which implies taking a significantly longer route. As shown in [8], this is a challenging situation even for traditional planners.

D. Analysis of learning variants

1) Impact of rings: Rings encodes the space around the robot as values between 0 and 1 in a pseudo-probabilistic representation of occupancy. It seems that allowing the NavRep model to infer from and predict occupancy uncertainties explicitly makes its task easier than inferring directly from

the raw LiDAR scan. In addition, rings provides a higher-level abstraction for the CNN to operate in, meaning that it requires less depth, while scaling the spatial resolution information to focus on regions close to the robot. This is in line with how sensors typically work and also matches the way that, in obstacle avoidance, more attention should usually be paid to close objects. The difference in resolution sensitivity between the 1D and rings representations could also explain the dramatic performance drop observed in the realistic map for the end-to-end model. Local features in the LiDAR representations for the simple vs. realistic maps vary more in the 1D case than in the rings. Moreover, rings more easily allows using the same model with different sensors of varying resolution.

2) Joint vs modular training: Previous work [11] recommends modular training, that is, training V, M and C separately, and to use both z and h as inputs to the C module. However, in this work we find that jointly trained V+M models provide superior performance in almost all tasks. The computational cost of joint training is furthermore not significantly greater than for modular training, though memory usage does increase due to having to store sequences of rich observations rather than compressed latent features.

3) Latent feature usefulness: When looking at model performances both in seen and unseen environments, our results do not support the idea that using both z and h latent features is advantageous. Though in some cases models trained with $z + h$ features performed slightly better, this improvement is not significant when compared to the variance between random seeds. In our experience, the performance cost of including h latent features in the C training regime is great: training a $z + h$ C model takes approximately 9 times longer than for an equivalent z-only C model. h-only models have it worst, with low test performance and slow training times.

E. Real-robot trials

We took our Transformer architecture (rings, z-only) and implemented it on a Pepper robot (Fig. 11). Our robot has a quasi-holonomic base and two $270°$ LiDARs merged together to form a 1080-vector combined range scan. The robot was given several waypoints in a space mixed with people and obstacles, and tasked to patrol between waypoints.

The learned controller achieved 100% success in the real environment, reaching its goal for all trajectories. However, we observed issues which show a need for improvement. (i) Smoothness: the commands selected by the controller are aggressive, leading to jerky motion. (ii) Reluctance to go into tight spaces: when faced with narrow but traversable areas, the controller oscillates around the entrance but eventually makes it through. (iii) Getting stuck: we observed the planner spend several seconds oscillating in front of an obstacle before finally passing on one side.

These issues appear to be in part due to the following causes: (i) Inertia in the dynamics of the robot is not modelled in the training environment, which causes sub-optimal control in the policy. (ii) Sensor noise is not present in the simulated environment. As a result, when it occurs
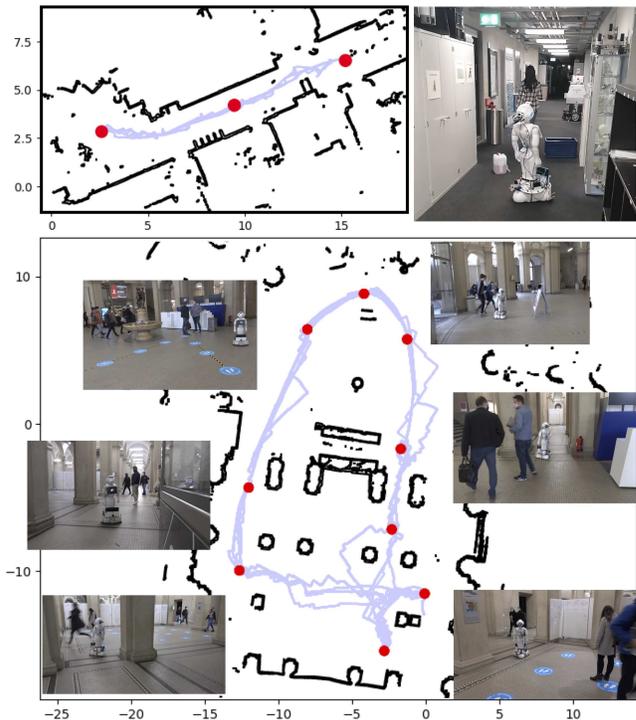
Fig. 11. We test the proposed method on a real robot, where the learned policy is repeatedly able to avoid obstacles and people and reach its goals. Goals are shown in red, successful trajectories in blue, images taken during the experiment are displayed next to the locations they were taken in. Each sequence was run three times in a row.

in the real scenario, the controller can react unpredictably. We surmise that there are further unknown reasons. Such "unknown unknown" effects could possibly be addressed via continual learning by refining the model in the real world.

VI. CONCLUSION

Training robust sensor-to-control policies for robot navigation remains a challenging task. NavRep methods do not dominate over end-to-end methods, but we do see interesting properties and trade-offs between the two. Though less suited to learning specific tasks, unsupervised representations display several benefits, such as more consistent performance across general tasks, modularity, and the potential to generate "Dream" environments [11], which could assist or replace simulation. The popularity of these methods is additionally propelled by the continued increase in compute and memory availability. Moving forward we expect to see larger unsupervised models with richer modalities, that create more convincing approximations of reality and provide more useful and discerning features.

Nevertheless, navigation-RL still has a long way to go, should one aim to replace traditional planners. Among the current issues are global planning, sim-2-real constraints, and safety. Potential avenues include more rigorous exploration of reward functions, more realistic training environments, and continual learning. Given that standardized gym-like environments could help harmonize future research, we hope that by making NavRep and NavRepSim publicly available and easy to use, progress in this domain can be accelerated and to improve collaboration between researchers.

REFERENCES

[1] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," *arXiv preprint arXiv:2003.13350*, 2020.

[2] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *2017 ieee international conference on robotics and automation (icra)*. IEEE, 2017, pp. 1527–1533.

[3] 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, 2018.

[4] T. Fan, P. Long, W. Liu, and J. Pan, "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," *The International Journal of Robotics Research*, p. 0278364920916531, 2020.

[5] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 285–292.

[6] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6015–6022.

[7] L. Q. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot navigation in crowded environments using deep reinforcement learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, in press.

[8] D. Dugas, J. Nieto, R. Siegwart, and J. J. Chung, "Ian: Multi-behavior navigation planning for robots in real, crowded environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, in press.

[9] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1343–1350.

[10] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

[11] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems*, 2018, pp. 2450–2462.

[12] A. Piergiovanni, A. Wu, and M. S. Ryoo, "Learning real-world robot policies by dreaming," *arXiv preprint arXiv:1805.07813*, 2018.

[13] S. W. Kim, Y. Zhou, J. Philion, A. Torralba, and S. Fidler, "Learning to simulate dynamic environments with gamegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1231–1240.

[14] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training."

[15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[16] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[17] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 1928–1935.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[20] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert *et al.*, "Stable baselines," *GitHub repository*, 2018.