Real-to-Sim-to-Real: Learning Models for Homogeneous Multi-Agent Systems

Anton Gurevich*, Eran Bamani* and Avishai Sintov

***Abstract*— Training Reinforcement Learning (RL) policies for a robot requires an extensive amount of data recorded while interacting with the environment. Acquiring such a policy on a real robot is a tedious and time consuming task. It may also cause mechanical wear and pose danger. This is more challenging in a multi-agent system where individual data is required from each agent. While training in simulations is the common approach due to efficiency and low-cost, they rarely describe the real world. Consequently, policies trained in simulations and transferred to the real robot usually perform poorly. In this letter, we present a novel real-to-sim-to-real framework to bridge the reality gap for homogeneous multi-agent systems. First, we propose a novel deep neural-network architecture termed *Convolutional-Recurrent Network (CR-Net)* to simulate agents. CR-Net includes convolutional, recurrent and fully-connected layers aimed to capture the complex transition of an agent. Once trained with data from one agent, we show that the CR-Net can also accurately predict motion of other agents in the group. Second, we propose to invest a limited amount of real data from one agent in a generative model. Then, training the CR-Net with synthetic data sampled from the generative model is shown to be at least equivalent to real data. The generative model can also be disseminated along with open-source hardware for easier usage. Using the models, we build a simulation that is based solely on data from one agent. We show experiments on ground and underwater vehicles in which multi-agent RL policies are trained in the simulation and successfully transferred to the real-world.**

I. INTRODUCTION

Multi-Agent Reinforcement Learning (MARL) addresses the decision-making for multiple agents in a common environment working toward a common task [1], [2]. The agents simultaneously learn a policy by observing and interacting with the same environment. Each agent is working to maximize some individual reward or a shared one for all agents [3]. Learning for multi-agent systems has potential applications in coordinated search and rescue, autonomous vehicle routing [4] and wireless sensor networks [5]. In addition, MARL also applies to policy training for aerial, ground and underwater swarms [6].

Many different MARL algorithms were developed in recent years [7] while only few were tested on real robots [8], [9]. Training RL policies on real robots is a tedious and time consuming task. In addition, the robot must work for a very long time which may cause damage and wear. The problem is even more challenging in a multi-agent system where individual data is required from each agent. Training in simulation, on the other hand, is a compelling solution where the data is acquired at a lower cost [10]. Simulation-based learning provides a cost-effective way to collect data through interactions with the environment. Such approach, for instance, was used for obtaining control for an autonomous vehicle using a simulation with synthetic images [4]. A similar approach was used for autonomous soil excavation [11]. However, simulations rarely capture reality and the trained policies are usually poorly transferred [12]. This problem is even worse for open-source hardware such as for underactuated robotic hands [13] and mobile robots [14]. Such hardware is usually 3D printed accompanied by low-cost actuators which impose many fabrication uncertainties in, for example, friction, size, mass and compliance. Therefore, hand-crafting models for these systems is a challenging problem leading to the lack of a good simulator.

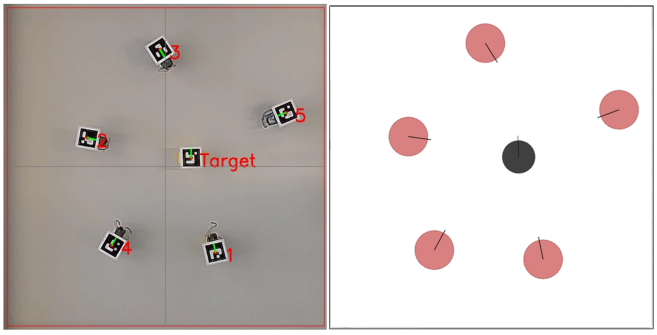


Fig. 1: (left) A group of micro-ground robots reaching a target with a trained reinforcement learning policy and (right) the data-based simulator used for training the policy.

Learning a policy solely from a simulation and deploying it to the real world is considered a hard challenge. This problem is commonly referred as the *reality gap* or *sim-to-real* (simulation to reality). When lacking a feasible simulator, the common approach to bridge the reality gap is to collect data from the real agent in order to generate a data-driven simulation. Such approach is often termed *real-to-sim* [15]. The training data is used to learn a *transition model* which maps a current state of the robot and a desired action to the next state. This is commonly done using a supervised learning method such as an Artificial Neural Network (ANN) or Gaussian Processes [16]. With such model, one can train RL policies using a nearly real simulator and transfer them to the agent. However, this approach yet requires a large amount of training data. In addition, a model trained on one agent may not fit a similar agent due to fabrication uncertainties. Hence, practitioners interested in sharing data or a trained model corresponding to some hardware (e.g., open-source hardware [13]) may not be able to also disseminate accuracy.

A. Gurevich, E. Bamani and A. Sintov are with the School of Mechanical Engineering, Tel-Aviv University, Israel. e-mail: {eranbamani, antong}@mail.tau.ac.il.

* These authors contributed equally.

In this work, we propose a *real-to-sim-to-real* framework to learn a robust, general and task-agnostic data-based simulation for an homogeneous multi-agent system to be deployed in the real world. We treat the multi-agent system as a set of single-agent forward models. This allows easy model learning and policy training that is suitable for various mobile robot group tasks. An example of such system and the corresponding data-based simulator is seen in Figure 1. The proposed framework illustrated in Figure 2 consists of two main components: a novel ANN architecture for learning a transition model and the generation of synthetic data to train it. We first propose the *Convolutional-Recurrent Network* (CR-Net) designed to learn state transitions. CR-Net is a novel ANN architecture which uses convolutional layers and Long Short Term Memory (LSTM) cells. We show that the CR-Net achieves high accuracy with data collected from a single agent and evaluated on others. Hence, the model can be learned on one agent while used individually in a multi-agent system.

In the second component of the framework, we propose to invest a limited amount of real data recorded from one agent to train a generative model. The generative model can provide an infinite amount of synthetic transition data. Hence, the required effort to collect data is reduced. We utilize the Generative Adversarial Network (GAN) [17]. Synthetic labeled data generated by the GAN with a much lower cost than real data would be used to train the CR-Net. The synthetic data can be used to supplement the real data or replace it completely. The CR-Net along with GAN provides an accurate model, that is at least equivalent to using real data, for similar agents with no further data collection effort.

The framework enables hardware, or open-source hardware in particular, to be accompanied with an already trained GAN. Therefore, open-source dissemination of the generative model rather than data would be easier and provide flexibility for the prospective user. The user will have a deployment-ready labeled data generator which can cope with fabrication uncertainties of open-source hardware. The generator can immediately provide synthetic data for a user to build a custom (e.g., one can choose which features in the state to used) and close-to-reality simulator. In the simulator, the same transition model could be applied individually to each agent in a multi-agent system. The simulator can be used for training a model-free or model-based RL policy to complete a shared task. Deployment of the policy on the real agents is then straight-forward without additional effort.

II. Related Work

Many approaches have been proposed for bridging the reality gap and to apply sim-to-real transfer. Early approach suggested adding noise to the simulation [18]. More recently, domain randomization or domain adaptation was proposed where various properties in an existing simulation are constantly varied [10]. Similarly, Peng et al. [12] proposed dynamics randomization to randomly sample dynamic properties (e.g., robot link mass, damping and friction) in the simulator during training. In such way, the policy is able to adapt to uncertainties that may emerge when transferring to the real system. Such approach, however, can be time-consuming since the policy must experience a large variance of dynamic possibilities. The work of Kasper et al. [19] used system identification rather than dynamics randomization to align a simulation with the real robot. A different approach used a complementary neural-network that is trained to predict the residuals between simulated and real trajectories [20]. All of these approaches require the formation of a physics-engine based simulation that is sufficiently close to the real system and environment.

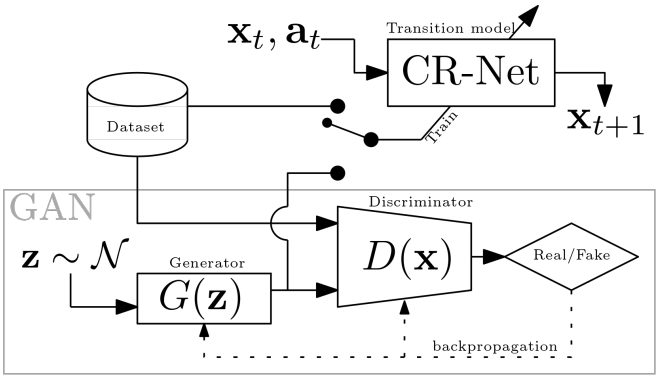


Fig. 2: The proposed CR-Net is a transition model trained to map current state and future action to the next state. The model can be trained using data from the real system or synthetic data from the generator of the GAN.

In this work, we explore the possibility of investing the recorded data in a generative model rather than directly to a regression model. In recent years, attention has been put in developing ANN models that can capture the complex distribution of some data and generate artificial data from the same distribution [21]. Some of the approaches include Variational Auto-Encoders (VAE), Generative Adversarial Network (GAN), Deep Convolutional GAN (DC-GAN), a fully connected and convolutional GAN (FCC-GAN), Gaussian Mixture Model GAN (GMM-GAN) and Cycle-GAN [22]. We focus in our work on GAN which is a powerful method to learn complex data distributions. GAN is mostly used for image processing [23] and visual perception [24] while also having other applications such as in health care [25] and motion planning [26].

III. Methods

A. Problem Formulation

Let $\mathbf{x} \in \mathcal{C}$ be the state of agent A where $\mathcal{C} \subset \mathbb{R}^n$. Further, $\mathbf{a} \in \mathcal{U}$ is an action exerted on the agent where $\mathcal{U} \subset \mathbb{R}^m$ is the action space. Assuming a Markov Decision Process (MDP), the motion of A is governed by the function $f : \mathcal{C} \times \mathcal{U} \rightarrow \mathcal{C}$ such that, given the current state $\mathbf{x}_i$ and action $\mathbf{a}_i$, the next state is given by $\mathbf{x}_{i+1} = f(\mathbf{x}_i, \mathbf{a}_i)$. An analytical formulation of transition model f is usually unavailable or inadequate due to fabrication inaccuracies and inherent uncertainties in the environment. Consequently, an hand-crafted model

f particularly tuned for agent A will, most likely, yield erroneous predictions for agent B. Therefore, we aim to learn a transition model $\tilde{f}$ trained over data from one agent that can be independently applied to each individual agent in a multi-agent system.

B. Data collection

Training data is collected by driving an agent in the state space with random actions. During motion, ground-truth data of trajectories is provided. Thus, the resulting data is a set of observed states and actions $\mathcal{P} = \{(\mathbf{x}_1, \mathbf{a}_1), \ldots, (\mathbf{x}_N, \mathbf{a}_N)\}$. The trajectories in $\mathcal{P}$ are processed to a set of inputs $(\mathbf{x}_i, \mathbf{a}_i)$ and corresponding labels of the next state $\mathbf{x}_{i+1}$. Hence, the training data for transition model $\mathbf{x}_{i+1} = \tilde{f}(\mathbf{x}_i, \mathbf{a}_i)$ is of the form $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{a}_i), (\mathbf{x}_{i+1})\}_{i=1}^{N-1}$. Note that in some systems, directly learning the next state can be difficult when the sampling frequency is high and the consecutive states are too similar. Therefore, it is common to learn the change from state $\mathbf{x}_i$ given an action $\mathbf{a}_i$ over the time step Δt. Hence, the training dataset will be of the form $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{a}_i), (\Delta\mathbf{x}_{i+1})\}_{i=1}^{N-1}$ where $\Delta\mathbf{x}_{i+1} = \mathbf{x}_{i+1} - \mathbf{x}_i$. The model would then predict

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \tilde{f}(\mathbf{x}_i, \mathbf{a}_i). \tag{1}$$

C. Generative Adversarial Network

A generative model is a statistical model that can generate new data from the distribution of a real dataset. Given a set of data inputs X and labels Y, a generative model would capture the joint probability $p(X, Y)$ and would be able to generate a new instance $(\mathbf{x}, \mathbf{y}) \sim p(X, Y)$. A Generative Adversarial Network (GAN) is a generative model based on deep neural-networks [27]. As reviewed in Section I, GAN is commonly used for image processing and to generate new image data. Nevertheless, we propose the use of GAN to generate synthetic one-dimensional samples that describe the motion of an agent.

GAN is comprised of two networks trained simultaneously: a *discriminator* and a *generator* networks as seen in Figure 2. The generator is trained to capture the distribution of the training data and to generate plausible data of the same distribution. The discriminator, on the other hand, is trained to distinguish between real data and fake data outputted by the generator. Hence, the discriminator is able to penalize the generator for producing implausible results. In contrast, the generator attempts to maximize the probability of the discriminator to incorrectly classify either real or fake data instances. Consequently, GAN is often considered as a two-player game where both generator and discriminator try to overcome each other. Each model has its own loss function. Let $\mathcal{Q} = \mathcal{C} \times \mathcal{U} \times \mathcal{C}$ be the joint space of a transition such that $(\mathbf{x}_i, \mathbf{a}_i, \mathbf{x}_{i+1}) \in \mathcal{Q}$. The generator function $G : \mathbb{R}^d \to \mathcal{Q}$ maps a d-dimensional noise vector to a joint transition vector where d is some tunable hyper-parameter. The noise vector $\mathbf{z} \in \mathbb{R}^d$ is randomly sampled from a prior normal distribution $\mathcal{N}$ and yields a generated state sample $G(\mathbf{z}) \in \mathcal{Q}$. The discriminator function $D : \mathcal{Q} \to [0, 1]$ takes a joint transition vector $\mathbf{q} \in \mathcal{Q}$ and tries to classify whether it came from the real dataset or artificially generated. The output is a single scalar denoting the certainty of the classification.

The discriminator input comes from two sources including real instances from distribution μ and fake ones created by the generator. Deriving the cross-entropy between the real and generated distributions, the loss function is defined by

$$V(D, G) = \mathbb{E}_{\mathbf{x}\sim\mu}[\log(D(\mathbf{x}))] + \\ + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}}[\log(1 - D(G(\mathbf{z})))] \tag{2}$$

where $\mathbb{E}$ denotes the expected value. While the training of the discriminator aims to maximize $V(D, G)$, i.e.,

$$L_D = \max_D V(D, G),$$

the generator requires the opposite by solving

$$L_G = \min_G V(D, G).$$

We note that the generator cannot directly minimize $\log(D(x))$ but only the second component in (2), i.e., $\log(1 - D(G(z)))$. In our work, we generate states and actions of an agent that resemble real recorded data. Therefore, the generator G would generate joint transition data $\{(\mathbf{x}_k, \mathbf{a}_k, \mathbf{x}_{k+1})\}_{k=1}^{M}$ from the distribution of $\mathcal{Q}$.

D. Long Short Term Memory

Long Short Term Memory (LSTM) is a class of Recurrent Neural-Network (RNN) aimed to learn sequential data [28]. RNN's utilize previous outputs as inputs while including hidden states. However, the standard RNN is usually not capable of handling long intervals where back-propagating errors tend to vanish or explode [29]. LSTM, on the other hand, is capable of learning long-term dependencies by utilizing memory about previous inputs for an extended time duration [30]. Along with an hidden state vector, LSTM maintains a cell state vector. At each time step, the process may choose to read from the cell vector, write to it or reset the cell using an explicit gating mechanism. Each cell unit has three gates of the same shape. The input gate controls whether the memory cell is to be updated or, in other words, which information will be stored. Forget gate decides whether to reset the memory cell and removes irrelevant information from the cell state. Similarly, output gate controls whether the information of the current cell state is made visible and adds useful information to the cell state. All gates commonly have a Sigmoid activation function. An additional hyperbolic activation function distributes the values flowing through the network and, therefore, prevents vanishing or exploding gradients. The LSTM is trained using recorded data sequences with back-propagation.

E. CR-Net

We propose the CR-Net model aimed to estimate transition function f of an agent. The model receives an $(n + m)$-dimensional input vector including the current state $\mathbf{x}_i$ and future action $\mathbf{a}_i$. CR-Net is composed of three parts as seen in Figure 3. The first part is a Convolutional Neural-Network (CNN). CNN is a class of artificial NN that contains one or more convolutional layers and is popular in tasks of image

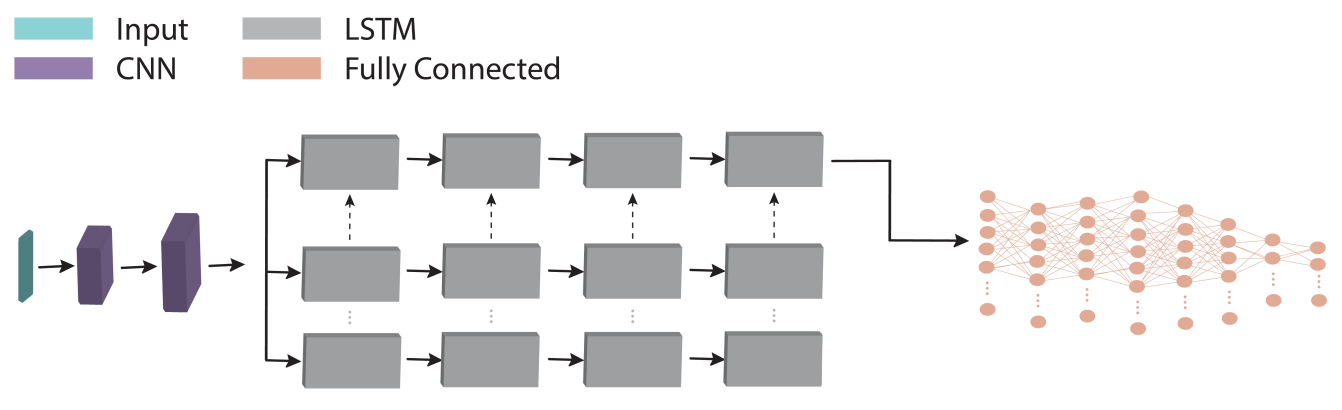


Fig. 3: The proposed CR-Net architecture to predict the next step of the agent.

classification and segmentation [31]. A salient advantage of CNN is in its ability to automatically capture important features without human supervision. Included filters allow the network to study different properties of the data and enable to recognize low and high order patterns. CNN usually works with unstructured data such as images. However, our model passes an input vector through n_c one-dimensional convolution layers, batch normalization and Leaky ReLU.

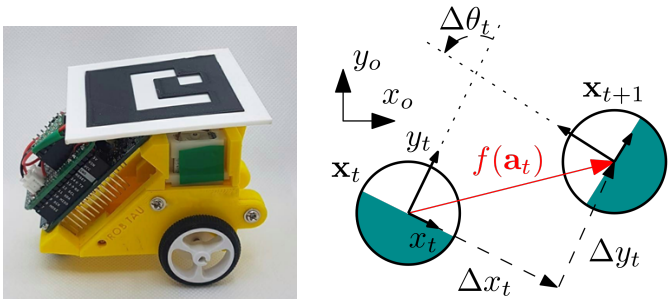
The second part of the model is an LSTM that receives an n_b-dimensional flattened vector from the CNN. While the features are non-sequential [32], yet the model is able to learn some causality between the features in the vector leading to the corresponding output label. The LSTM contains n_l recurrent layers and n_h hidden layers. The last part is a Fully-Connected NN (FC-NN) that receives the output of the CNN and LSTM. The FC-NN contains n_f hidden layers, batch normalization, dropout and an hyperbolic tangent (Tanh) activation function. The last layer contains a linear function to predict the next state $\mathbf{x}_{i+1}$. We note that the conventional use of FC-NN and sequential states for the LSTM are tested in the experimental section while yielding inferior performance. CR-Net can be trained with real data (i.e., *CR-Net-R*) or with synthetic data (i.e., *CR-Net-S*) sampled by the generator of the GAN.

IV. Experiments

In this section, we test and analyze the proposed methodology and framework. We experiment on groups of ground and underwater robots. The robots are fabricated through 3D printing and, therefore, their design is not of high accuracy. Furthermore, the robots use low-cost actuators such that two identical ones do not perform equivalently. Consequently, each agent may behave slightly different upon exerting actions. We show that using data from only one agent, the GAN and CR-Net can provide an accurate model for a multi-agent system. The models are later demonstrated on a MARL framework. Videos of the experiments can be seen in the supplementary material.

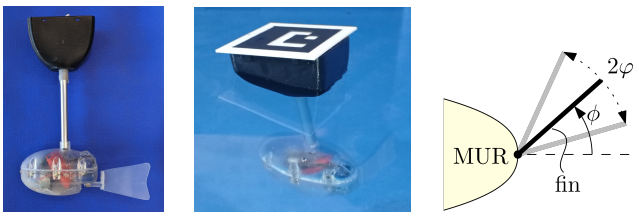
A. Systems

We test our approach on two micro-robot groups including ground and underwater vehicles. System CAD models and already trained GAN of the micro-ground robot are open-sourced[1]. Therefore, any user can build multiple robots and easily generate synthetic data for modeling them.



(a) (b)

Fig. 4: (a) Micro-Ground Robot (MGR) and (b) its state transition from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$.



(a) (b) (c)

Fig. 5: The (a) Micro-Underwater Robot (b) in the water and (c) a diagram of its tail-fin propulsion.

1) Micro-Ground Robot (MGR): MGR is a two wheel drive mobile robot seen in Figure 4a. The body of the MGR is fabricated by 3D printing with a Polylactic-Acid (PLA) filament. The robot size is $90 \times 84 \times 54$ mm and it weighs 130 grams. It is equipped with a motor for each wheel, a DRV8835 motor controller and an Arduino board. An action $\mathbf{a}_t \in \mathbb{R}^2$ of the robot is the required angle changes for the two wheels during some constant time step. Furthermore, we assume that the surface in which the agent moves on is uniform such that its transition does not depend on the current state. Therefore, transition model (1) depends solely on the action $\Delta\mathbf{x}_t = f(\mathbf{a}_t)$ where $\Delta\mathbf{x}_t = (\Delta x_t, \Delta y_t, \Delta\theta_t)^T$

[1] https://github.com/eranbTAU/Closing-the-Reality

TABLE I: Accuracy comparison for different models trained with real or synthetic data

	Model	Micro-Ground Robot (MGR)				Micro-Underwater Robot (MUR)			
		Position error (mm)		Orientation error (°)		Position error (mm)		Orientation error (°)	
		Train vehicle	Test vehicle	Train vehicle	Test vehicle	Train vehicle	Test vehicle	Train vehicle	Test vehicle
Real	FC-NN	12.31±7.24	11.08±6.03	15.55±7.77	13.55±7.12	29.59±14.62	30.26±14.61	23.64±17.08	22.77±16.57
	CNN	6.91±4.41	6.12±3.84	6.75±6.17	5.95±6.18	22.74±14.27	22.81±14.93	15.78±14.75	14.72±14.90
	LSTM	6.88±3.85	6.88±3.20	10.55±6.90	10.60±6.17	26.02±15.09	25.24±14.54	16.71±15.01	17.42±15.58
	GRU	9.45±5.61	7.71±4.59	19.56±14.85	15.89±13.40	26.69±14.58	27.65±15.20	27.65±15.20	19.77±15.74
	CR-Net-R	4.66±4.12	4.06±3.60	5.46±4.06	4.93±5.08	17.16±13.48	14.56±11.96	8.08±9.125	6.54±7.70
Synthetic	FC-NN	11.15±5.54	9.33±4.73	8.19±6.63	6.74±5.23	30.41±15.03	31.22±15.11	23.23±18.07	22.70±17.66
	CNN	6.73±3.32	5.75±2.8	6.84±5.57	5.59±5.25	22.51±14.48	22.94±14.90	16.48±15.75	15.41±15.16
	LSTM	6.15±4.19	5.92±3.73	7.40±5.80	6.06±5.34	26.32±15.25	25.38±14.75	17.37±14.96	18.11±15.58
	GRU	9.31±6.44	8.01±6.27	11.47±10.13	9.72±10.71	27.87±15.55	27.03±14.89	27.03±14.89	21.65±17.32
	CR-Net-S	4.53±4.23	4.06±3.53	5.23±5.56	4.82±5.14	18.34±13.94	16.53±13.09	16.53±13.09	6.94±8.17

is the state change relative to the MGR coordinate frame at time t as seen in Figure 4b. We have built five MGR units while only one was used for data collection. Data was collected as described in Section III-B while having the agent move across a smooth whiteboard plane (Figure 1). A set of $N = 55,068$ labeled data points from various trajectories was collected. Test data was acquired from another agent. The other MGR's will further be used for RL demonstration.

2) Micro-Underwater Robot (MUR): MUR is an underwater mobile robot that swims in a constant depth of 200 mm. The constant depth is maintained by a float connected to the MUR through an aluminum pole as seen in Figures 5a-5b. The body of the MUR is fabricated using Stereolithography (SLA) 3D printing with a resin photo-polymer material. Its size is $160 \times 205 \times 40$ mm and it weighs 270 grams. The MUR is propelled with a tail-fin actuated by a micro waterproof servo motor. The float contains an Arduino MK1000 controller and a lithium battery. The fin of the MUR oscillates in a sinusoidal motion such that its angle relative to the body is given by $\alpha(t) = \phi + \varphi \sin(\omega t)$ where ϕ, φ and ω are the oscillation mean angle, amplitude and frequency, respectively (Figure 5c). Hence, an action $\mathbf{a}_t \in \mathbb{R}^3$ is defined by $\mathbf{a}_t = (\phi, \varphi, \omega)^T$ which affects propulsion direction and velocity. Such motion is a low-level mimic of underwater fish locomotion. Unlike the MGR, the transition model of the MUR considers its state (i.e., $\Delta\mathbf{x} = f(\mathbf{x}_t, \mathbf{a}_t)$) since moving on the sides of the pool may be different than on its interior. Data was collected in a $2m \times 1m$ pool reaching $N = 15,000$ labeled data points from various trajectories of the train agent. Trajectories from another test agent were used for validation.

For both MGR and MUR, commands are transmitted through Wi-Fi from a PC terminal in a 1.25-2 Hz frequency. Furthermore, each robot has an Aruco marker with a unique ID such that an upper camera can identify and acquire its state $\mathbf{x}_t \in SE(2)$ relative to some world coordinate frame in real-time.

B. Model training

The collected training set is acquired as described above. The acquired training dataset is used to train either a GAN model to generate more training samples or directly the transition model. The hyper-parameters of the tested networks were optimized to produce the best loss value. In particular, the optimal hyper-parameters of the CR-Net are $n_c = 2$, $n_b = 8$, $n_l = 2$, $n_h = 5$ and $n_f = 8$. Thus, the total number of trainable parameters is 37,211. We used the AdamW optimizer along with a general loss function that sums the Mean Squared Error (MSE) of the agent's position and Mean Absolute Error (MAE) of its orientation. Moreover, we used adaptive learning rate initialized at 0.005138, L2 regularization of 0.006 and batch size of $d_s = 64$ along 40 epochs.

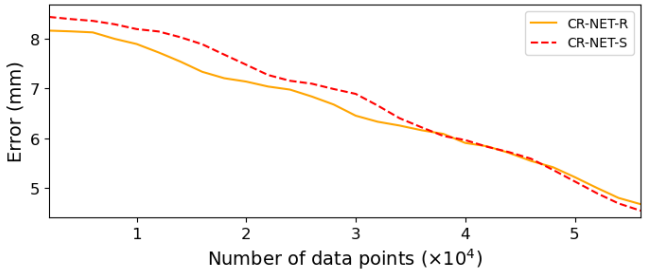


Fig. 6: MGR position accuracy for the CR-Net when trained using real and synthetic data with regards to data size.

C. Model Evaluation

We begin by analyzing the accuracy of the proposed approach. We present a comparison to other common models including FC-NN, CNN, LSTM and Gated Recurrent Unit (GRU) [33]. All models were optimized to yield the lowest loss value and evaluated on the test data. Table I shows results for the average one-step prediction error while training the models using the real train data without using GAN and synthetic data generated by the GAN. We include accuracy results evaluated on test data acquired from train and test vehicles. Results show that CR-Net outperforms all other methods.

Figure 6 compares accuracy of the CR-Net-R and CR-Net-S models, and with regards to the amount of data used. For CR-Net-S, a GAN was trained with the entire real dataset. Similarly, Figure 7 shows the accuracy of a mixed CR-Net model trained on both real and synthetic data. The figure shows the position accuracy with regards to the ratio between synthetic and real data. Results show that synthetic data is at least equivalent to real data and can even improve accuracy.

We next justify the use of GAN with regards to other generative models. We compare the standard GAN model

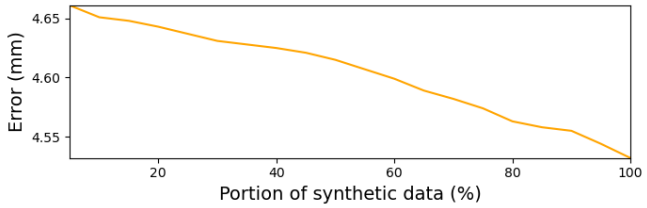


Fig. 7: MGR position accuracy for the CR-Net with regards to the ratio between real and synthetic training data.

TABLE II: Accuracy comparison for different generative models for the test MGR

	Position error (mm)	Orientation error (°)
DC-GAN	5.12	7.47
FCC-GAN	6.76	9.90
GMM-GAN	5.01	5.73
Cycle-GAN	4.95	5.42
GAN	4.06	4.82

with DC-GAN, FCC-GAN, GMM-GAN and Cycle-GAN. All models were trained with the same data. For each trained model, we have generated $55,000$ synthetic points and trained a CR-Net-S. Table II reports the mean position and orientation errors for the CR-Net-S with regards to different generative models. Synthetic data generated by the standard GAN provides better robustness to the CR-Net.

Figure 8 presents the position error of different regression models with regards to the amount of data used to train the GAN. Once trained a GAN, the amount of synthetic data generated for training the models was constant and equal to 55,000 points. CR-Net-S evidently maximizes the use of the synthetic data. To better emphasize the contribution of synthetic data, Figure 9 shows an heat-map of the MGR position accuracy with regards to the real data used to train the GAN and the synthetic data generated from the GAN used to train the CR-Net-S. A relatively good accuracy can be obtained with less than half of the collected real data and the generation of a large synthetic dataset. For example, with only $15,000$ real points and $54,000$ synthetic points (green marker in Figure 9), the mean position accuracy ($6.00\ mm$) is equal to $54,000$ real points (red marker). The results imply that it may be better to invest a limited amount of recorded data for training a GAN rather than directly training some NN model. With the GAN, one can generate enough synthetic data points to have an accurate model.

With the transition model, one can simulate the motion of

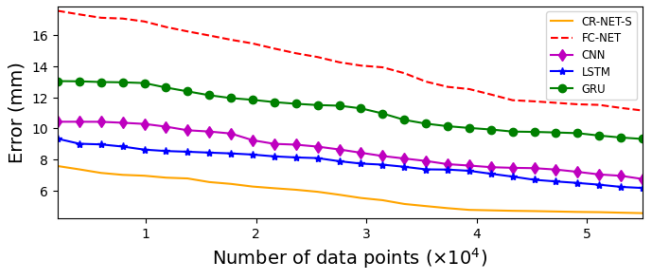


Fig. 8: Transition model accuracy trained on synthetic data with regards to the size of data used to train the GAN.

Fig. 9: A heat-map of the position accuracy (mm) with regards to the number real MGR data points used to train the GAN and the number of synthetic data points (from GAN) used to train the CR-Net-S. Green and red markers exemplify equal accuracy points.

an agent. Figure 10 shows trajectory predictions of an agent in an open-loop fashion with CR-Net-S along with FC-NN for comparison. A trajectory prediction is based solely on the initial state and a pre-determined sequence of actions. The predictions are shown along with the Root Mean Square Errors (RMSE) with regards to the recorded ground truth. Results show that, while FC-NN performs poorly, a simulator based on CR-Net-S sufficiently represents the real world. The simplified simulation environment is seen in Figure 1.

D. MARL Evaluation

We now evaluate the performance of the data-based MGR simulator for training single-agent RL and MARL policies. We first set a scenario in which one MGR agent must reach a target goal while avoiding collision with a circular obstacle (Figure 11). We have implemented the Deep Deterministic Policy Gradient (DDPG) [34] with a reward function that rewards for reaching the target and penalizes distance from the target or collision. Two policies were trained in simulation based on FC-NN and CR-Net-S. Table III compares between the two policies and presents the average rewards for ten roll-out trials in simulation and in real deployment. Each simulation is rolled-out from the same initial pose of the matching real roll-out. Furthermore, failure of a roll-out is the inability of the agent to reach the goal (either collision or getting stuck). The FC-NN based policy shows significant reward difference between simulation and real world along with very low success rate. Hence, the FC-NN does not encapsulates the real system well. On the other hand, CR-Net exhibits a good ability to simulate the real world with matching simulated and real rewards, and high success rate.

Next, a MARL scenario consists $K = 5$ MGR agents arranged in some arbitrary formation and must reach a target marker. A trained transition model is used to simulate each individual agent in the group. Here, DDPG is used in a multi-agent setting where all agents share the same policy. It is assumed that each agent has global knowledge about the state of all participating agents. Hence, the reward function

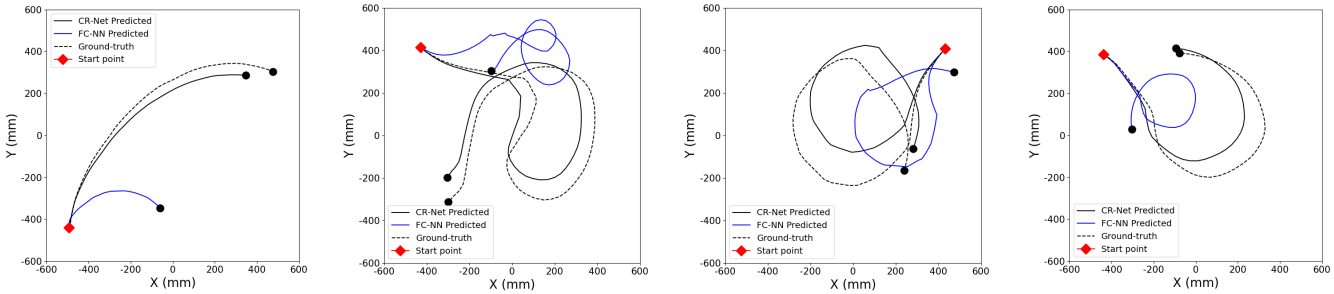


Fig. 10: Open-loop prediction of MGR trajectories based solely on the initial state and sequence of actions, while using CR-Net-S and FC-NN. The tracking RMSE of the CR-Net-S trajectories is, from left to right, 40.4 mm, 59.9 mm, 70.1 mm and 19.8 mm. In contrast, the RMSE of the FC-NN is 562.4 mm, 458.1 mm, 366.8 mm and 304.0 mm.

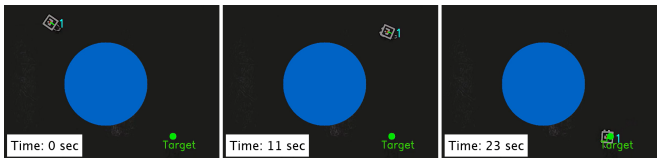


Fig. 11: An MGR agent avoiding an obstacle (blue circle) and reaching a target using an RL policy trained on a simulation based on CR-Net-S.

TABLE III: Average rewards for single-agent RL policy roll-outs

	Simulator reward	Real world reward	Real world success rate
FC-NN	-0.32±0.15	-0.58±0.20	20%
CR-Net-S	-0.26±0.04	-0.22±0.04	100%

$r : \mathcal{C} \times \mathcal{A} \to \mathbb{R}$ for each agent is defined to be

$$r(\mathbf{x}, \mathbf{a}) = r_t + \sum_{i=1}^{K-1} r_i, \text{ for } r_t = \begin{cases} -5, & \text{if } d_t < 150 \\ -20\frac{d_t}{d_{\max}}, & \text{otherwise,} \end{cases}$$

where $d_t = \|\mathbf{x} - \mathbf{x}_T\|$ is the agent's distance from target $\mathbf{x}_T$ and $d_{max} = 1,400$ is some user-defined constant. In addition, reward r_i is given by

$$r_i = \begin{cases} -5, & \text{if } d_i < 150 \\ 0, & \text{otherwise,} \end{cases}$$

where $d_i = \|\mathbf{x} - \mathbf{x}_i\|$ is the distance of the agent from agent i. The reward function, therefore, rewards agents moving towards the target $\mathbf{x}_T$ and punishes if the agent is too close (less then 150 mm) to nearby agents with the risk of collision. Then, two policies were trained, each using CR-Net-R or CR-Net-S, in simulation over 500 episodes while collecting experience data from all agents. We note that a policy trained based on the FC-NN model failed to achieve a feasible working one due to low accuracy.

The trained policies were then deployed on the real multi-agent system. To test the robustness of the CR-Net models, policy roll-outs were conducted on a black concrete surface rather then on the smooth surface used for data collection. We have tested ten roll-outs of 150 time-steps for each of the two policies. Each roll-out was deployed from different initial states and towards some arbitrary target. Furthermore, each roll-out was compared to a simulated roll-out from the same initial states and target. Table IV reports the average rewards for the simulated and real-world roll-outs with the two policies. The average real-world reward is almost similar to the simulated one. Hence, the simulator sufficiently represent the real world. Figure 12 shows an example of a policy roll-out based on CR-Net-S. The average reward acquired by the agents during the roll-out is -3.15. For comparison, the average reward in the simulation environment for the same initial states and target is -2.92. The results show that one trained CR-Net is applicable and sufficiently accurate for all agents in the group. We also demonstrate the roll-out of a policy trained in simulation for two MUR agents to reach target locations in the pool. Snapshots of the motion between two targets are seen in Figure 13.

TABLE IV: Average rewards for MARL policy roll-outs

Trained Model	Simulator	Real world
CR-Net-R	-3.29±0.33	-3.25±0.27
CR-Net-S	-3.13±0.26	-3.37±0.32

V. CONCLUSIONS

In this paper, we have proposed a real-to-sim-to-real framework to simulate a multi-agent system and control it in real-time. The framework is based on a novel ANN termed CR-Net to model the motion of the agents and a generative model for generating synthetic data. By collecting data solely from one agent and despite inaccuracies in the fabrication of other agents, a trained CR-Net can accurately predict motion for all agents in the group. Hence, the model is robust and can generalize to all agents. Furthermore, we examined the influence of training the CR-Net with synthetic data generated by a GAN. Results show that CR-Net along with synthetic data from GAN can achieve high accuracy that is at least equivalent to training with real data. The GAN is also a dissemination tool that can be accompanied to open-source hardware rather than real recorded data. In such case, the GAN can generate a massive amount of synthetic data points to be used in a custom model. The framework was analyzed and demonstrated on ground and underwater multi-agent systems while exhibiting accurate and robust performance.

REFERENCES

[1] J. N. Foerster, "Deep multi-agent reinforcement learning," Ph.D. dissertation, University of Oxford, 2018.

[2] K. Zhang, Z. Yang, and T. Başar, *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*. Springer International Publishing, 2021, pp. 321–384.

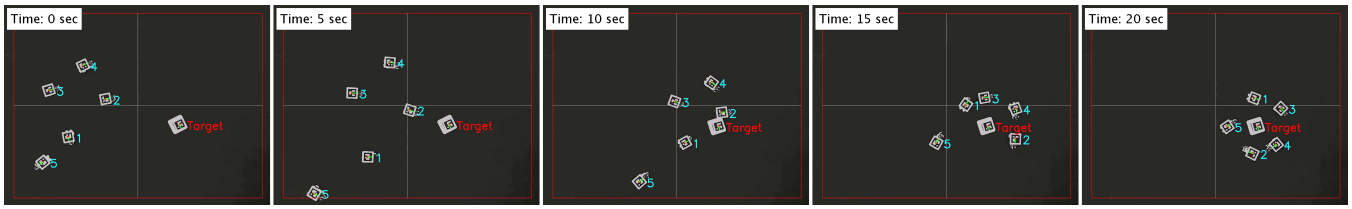


Fig. 12: A set of five MGR agents reaching a target using a trained policy while maintaining distance among them. The policy was trained on a simulation based on CR-Net-S.

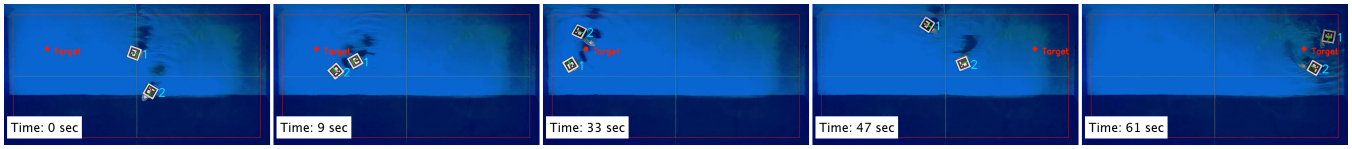


Fig. 13: Two MUR agents move in water between two targets using a trained policy while keeping minimal distance among them. The policy was trained on a simulation based on CR-Net-S.

[3] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Başar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *Inter. Conf. on Machine Learning*, vol. 80, 2018, pp. 5872–5881.

[4] B. Osinski, A. Jakubowski, P. Milos, P. Ziecina, C. Galias, S. Homoceanu, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 6411–6418.

[5] H. Zheng and D. Shi, "A multi-agent system for environmental monitoring using boolean networks and reinforcement learning," *Journal of Cyber Security*, vol. 2, pp. 85–96, 01 2020.

[6] M. Hüttenrauch, A. Šošić, and G. Neumann, "Deep reinforcement learning for swarm systems," *J. Mach. Learn. Res.*, vol. 20(1), p. 1966–1996, 2019.

[7] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Inter. Conf. on Autonomous Agents and Multiagent Systems*, 2017, pp. 66–83.

[8] T. Yasuda and K. Ohkura, "Sharing experience for behavior generation of real swarm robot systems using deep reinforcement learning," *Jour. of Robotics and Mechatronics*, vol. 31, no. 4, pp. 520–525, 2019.

[9] M. A. Billah and I. A. Faruque, "Bioinspired visuomotor feedback in a multiagent group/swarm context," *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 603–614, 2021.

[10] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020.

[11] O. Azulay and A. Shapiro, "Wheel loader scooping controller using deep reinforcement learning," *IEEE Access*, pp. 24 145–24 154, 2021.

[12] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE Inter. Conf. on Robotics and Automation (ICRA)*, 2018, pp. 3803–3810.

[13] R. R. Ma and A. M. Dollar, "Yale openhand project: Optimizing open-source hand designs for ease of fabrication and adoption," *IEEE Rob. & Aut. Mag.*, vol. 24, pp. 32–40, 2017.

[14] J. Yu, S. D. Han, W. N. Tang, and D. Rus, "A portable, 3d-printing enabled multi-vehicle platform for robotics research and education," in *IEEE Inter. Conf. on Robotics and Automation*, 2017, pp. 1475–1480.

[15] D. Hahn, P. Banzet, J. M. Bern, and S. Coros, "Real2sim: Visco-elastic parameter estimation from dynamic motion," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–13, 2019.

[16] A. Sintov, A. S. Morgan, A. Kimmel, A. M. Dollar, K. E. Bekris, and A. Boularias, "Learning a state transition model of an underactuated adaptive hand," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1287–1294, 2019.

[17] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the International Conference on Neural Information Processing Systems*, vol. 2. MIT Press, 2014, p. 2672–2680.

[18] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The

[20] F. Golemo, "How to train your robot-new environments for robotic use of simulation in evolutionary robotics," in *European Conference on Artificial Life*. Springer, 1995, pp. 704–720.

[19] M. Kaspar, J. D. M. Osorio, and J. Bock, "Sim2real transfer for reinforcement learning without dynamics randomization," *IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems*, pp. 4383–4388, 2020. training and new methods for transferring policies from the simulator to the real robot," Ph.D. dissertation, Université de Bordeaux, 2018.

[21] L. Ruthotto and E. Haber, "An introduction to deep generative modeling," *GAMM-Mitteilungen*, vol. 44, no. 2, p. e202100008, 2021.

[22] H. GM, M. K. Gourisaria, M. Pandey, and S. S. Rautaray, "A comprehensive survey and analysis of generative models in machine learning," *Computer Science Review*, vol. 38, p. 100285, 2020.

[23] N.-T. Tran, V.-H. Tran, N.-B. Nguyen, T.-K. Nguyen, and N.-M. Cheung, "On data augmentation for GAN training," *IEEE Transactions on Image Processing*, vol. 30, pp. 1882–1897, 2021.

[24] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *IEEE Inter. Conf. on Robotics and Automation (ICRA)*, 2016, pp. 512–519.

[25] T. Golany, D. Freedman, and K. Radinsky, "Ecg ode-gan: Learning ordinary differential equations of ecg dynamics via generative adversarial learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 1, 2021, pp. 134–141.

[26] T. S. Lembono, E. Pignat, J. Jankowski, and S. Calinon, "Learning constrained distributions of robot configurations with generative adversarial network," *IEEE Rob. & Aut. Let.*, vol. 6(2), 2021.

[27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[28] Y. Yu, X. Si, C. Hu, and J. Zhang, "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, 07 2019.

[29] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] A. Dhillon and G. Verma, "Convolutional neural network: a review of models, methodologies and applications to object detection," *Progress in Artificial Intelligence*, vol. 9, 12 2019.

[32] Y. Chen, J. Yang, and J. Qian, "Recurrent neural network for facial landmark detection," *Neurocomputing*, vol. 219, pp. 26–38, 2017.

[33] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (GRU) neural networks," in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 1597–1600.

[34] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2016.