Learning a Data-Efficient Model for a Single Agent in Homogeneous Multi-Agent Systems

Anton Gurevich[1†], Eran Bamani[1†] and Avishai Sintov[1*]

[1]School of Mechanical Engineering, Tel-Aviv University, Haim Levanon, Tel-Aviv, 6997801, Israel.

*Corresponding author(s). E-mail(s): sintov1@tauex.tau.ac.il;
Contributing authors: antong@mail.tau.ac.il;
eranbamani@mail.tau.ac.il;
†These authors contributed equally to this work.

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. This is more challenging in a multi-agent system where individual data may be 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 paper, we present a novel real-to-sim-to-real framework to bridge the reality gap for an agent in collective motion of a homogeneous multi-agent system. First, we propose a novel deep neural-network architecture termed *Convolutional-Recurrent Network (CR-Net)* to capture the complex state transition of an agent and simulate its motion. Once trained with data from one agent, we show that the CR-Net can accurately predict motion of all agents in the group. Second, we propose to invest a limited amount of real data from the 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. Hence, the proposed approach provides a sufficiently accurate model with significantly less real data. The generative model can also be disseminated along with open-source hardware for easier usage. We show experiments on ground and underwater vehicles in which multi-agent RL policies are trained in the simulation for collective motion and successfully transferred to the real-world.

Keywords: Robot modeling, forward dynamics model, Multi-Agent, Collective motion

1 Introduction

Multi-Agent Reinforcement Learning (MARL) addresses the decision-making for multiple agents in a common environment moving collectively [1, 2]. Many MARL algorithms were developed in recent years [3, 4] while only few were tested on real robots [5, 6]. In general, training Reinforcement Learning (RL) policies on real robots is a tedious and time consuming task [7]. In addition, the robot must work for a very long time which may cause damage and wear, and may pose danger. 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 [8]. Simulation-based learning provides a cost-effective way to collect data through interactions with the environment. Such an approach, for instance, was used for obtaining control for an autonomous vehicle using a simulation with synthetic images [9]. A similar approach was used for autonomous soil excavation [10]. However, simulations rarely capture reality and the trained policies are usually poorly transferred [11]. This problem is even worse for open-source hardware such as for underactuated robotic hands [12] and mobile robots [13]. 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, acquiring analytical models for these systems is a challenging problem, even for experienced practitioners, leading to the lack of a good simulator [14].

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 robot 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 *forward dynamics model* which maps a current state of the robot and a desired action to the next state [16]. This is commonly done using a supervised learning method such as an Artificial Neural Network (ANN) or Gaussian Processes [17]. With such model, one can train RL policies using a nearly real simulator and transfer them to the real system. However, this approach yet requires a large amount of training data, may be time consuming, can damage the robot and pose danger to its surrounding.

In this work, we propose a *real-to-sim-to-real* framework to learn a general and task-agnostic data-based simulation for one mobile robot. The simulation is later used to simulate all robots in a homogeneous multi-agent system to be deployed in the real world for collective motion control. In other words, we exploit the nature of the system such that having a model for one agent

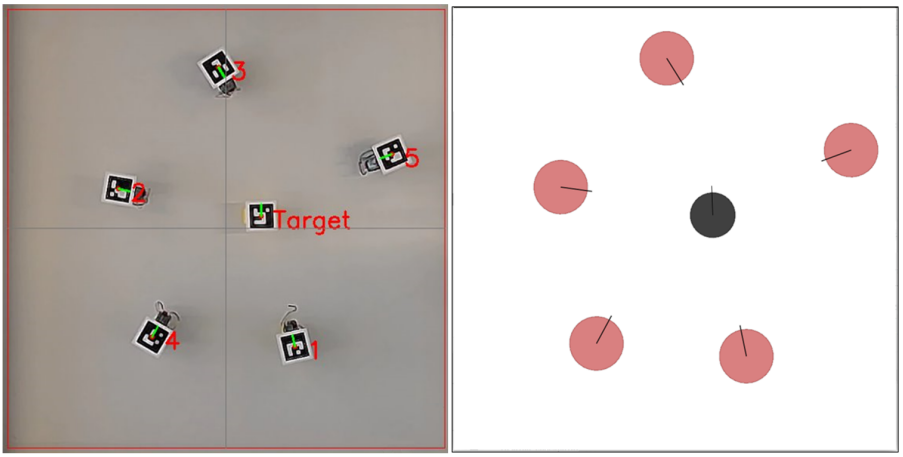


Fig. 1: (left) A group of micro-ground robots moving collectively towards a target with a trained reinforcement learning policy and (right) the data-based simulator used for training the policy.

and treat the multi-agent system as a set of single-agent forward models. This allows easy model learning that is suitable for various mobile robot group tasks. The learned model can be used to simulate a large number of agents in RL policy learning for collective motion while each agent can sense neighboring agents. In this work, we address small mobile robots with low-dimensional action and configuration spaces. Once a model of such robot is acquired, one can train a MARL policy in the data-based simulator and deploy many agents with low-cost. While we focus and demonstrate MARL, motion planning [18] and Model Predictive Control [19] are additional applications where synchronization between agents can be done using the proposed modeling framework. An example of a multi-agent system and the corresponding data-based simulator is seen in Figure 1. Furthermore, the proposed framework does not require any mechanical or dynamic modeling expertise for the particular mobile robot. A practitioner is only required to record a limited amount of real data during random motion of a single robot.

The proposed framework illustrated in Figure 2 consists of two main components: a novel ANN architecture for learning a forward dynamics model and the generation of synthetic data to train it. We first propose the *Convolutional-Recurrent Network* (CR-Net) designed to learn a forward dynamic 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 for all agents with data collected from a single agent. Furthermore, acquiring an accurate forward dynamics model for the robots may require extensive recording of a large number of real state-action transition samples. Therefore, rather than investing a large amount of data to train the forward dynamics model, 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 with low computational resources and without moving any robot. The limited amount of real data required to train the generative model is significantly lower than the amount of real data required to directly train the forward dynamics model. Hence, the required effort to collect data is reduced. We utilize the *Generative Adversarial Network* (GAN) [20]. 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. In fact, we show that our approach can significantly reduce the size of real data required in order to acquire an accurate forward dynamics model.

While the use of generative models to create synthetic training data is not novel, the common applications are the generation of synthetic image data [21]. For instance, recent work used GAN to generate synthetic simulation images for domain adaptation in order to reduce the number of real-world images required to achieve high performance [22]. Contrary to previous GAN work, we propose the use of GAN to generate synthetic one-dimensional data of state-action transitions. To the best of the author's knowledge, this is the first use of synthetic data that does not consider images and in the context of robot forward dynamics model. The proposed 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. It is important to note that, in this work, the goal is not to propose any RL policy or controller for a multi-agent system, but rather to improve sim-to-real transfer and to provide the foundation for a data-efficient and accurate simulator for training collective motion policies for such system.

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 forward dynamics 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 system is then straight-forward without additional effort.

2 Related Work

In MARL, 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 [23]. Learning collective motion

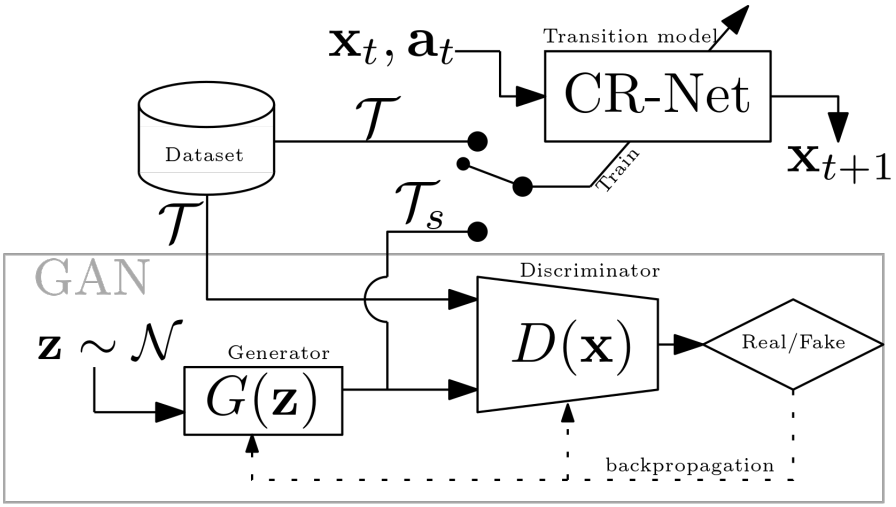


Fig. 2: The proposed CR-Net is a forward dynamics model trained to map current state and future action to the next state. The model can be trained using data $\mathcal{T}$ from the real system or synthetic data $\mathcal{T}_s$ from the generator of the GAN.

for multi-agent systems has potential applications in coordinated search and rescue, autonomous vehicle routing [9] and wireless sensor networks [24]. In addition, MARL also applies to policy training for aerial, ground and underwater swarms [25]. In this work, we focus on collective motion of a group of robots. Collective motion can be, for instance, joint reach to a goal, joint motion along a guiding path, reach to some organized form or distributed coverage across a pre-defined region [26, 27].

One may consider group collective motion with centralized or decentralized policies [28]. In a centralized policy, a global decision maker has access to the full state of the agents and commands joint actions to them all [29]. One approach is to learn a policy where the decision maker has a joint state representation of all agents. However, the state-space is exponentially grown with the increase of agents [30]. Hence, a different policy approach trains a set of independent sub-policies where the sum of local rewards is the joint reward [31]. However, some agents may become ineffective or passive so to not interfere other agents. In a decentralized policy, on the other hand, there is no global decision maker [32–34]. Therefore, each agent must take individual actions based on partial or local observation of the global system state [35]. In other words, an agent cannot see the local states of all other agents nor their next actions. Hence, it must decide the next action on its own, based on local observations of neighboring agents while also considering the target. From the view-point of the agent, the environment is dynamic since global knowledge is significantly limited [30]. When comparing centralized and decentralized policies, the latter has inferior performance partly due to poor ability to scale with

the number of agents [36]. The method proposed in this work can be applied to either centralized and decentralized policies.

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 [37]. More recently, domain randomization or domain adaptation was proposed where various properties in an existing simulation are constantly varied [8]. Similarly, Peng et al. [11] 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. [38] 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 [39]. All of these approaches require the formation of a physics-engine based simulation that is sufficiently close to the real system and environment. It is also worth mentioning some prior work on transition or forward models used for robot control [14, 16, 40].

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 [41]. 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 [42]. We focus in our work on GAN which is a powerful method to learn complex data distributions. GAN is mostly used for image processing [43] and visual perception [44] while also having other applications such as in health care [45] and motion planning [46]. Additional work worth mentioning are the COT-GAN [47] and SPATE-GAN [48] aimed to learn complex spatio-temporal patterns. Nevertheless, the work is not suitable for short term decision making as required in this work.

3 Methods

An homogeneous group of agents is required to perform collective motion. The group can either work under a centralized or decentralized policy. In both cases, one can find a model for a single agent and utilize it for a simulation of multiple ones. Therefore, this section presents a framework for accurate and data-efficient modeling of an agent. The model will later be used to evaluate collective motion of additional agents.

3.1 Problem Formulation

Let $\mathbf{x} \in \mathcal{C}$ be the state of agent A where $\mathcal{C} \subset \mathbb{R}^n$ and n is the dimensionality of state. Further, $\mathbf{a} \in \mathcal{U}$ is an m-dimensional 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 $f : \mathcal{C} \times \mathcal{U} \to \mathcal{C}$ 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 forward dynamics model f is usually unavailable or inadequate due to fabrication inaccuracies and inherent uncertainties in the environment. Consequently, an analytical model f particularly tuned for agent A will, most likely, yield erroneous predictions for agent B. Therefore, we aim to learn a forward dynamics model $\tilde{f}$ trained over data from one agent that can be independently applied to each individual agent in a multi-agent system.

3.2 Data collection

Training data is collected by driving an agent in the state space with uniform 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 forward dynamics 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 some time step. 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

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

It is assumed that uniform random actions provide sufficient coverage of the state space. Nevertheless and as will be discussed later on, GAN has an ability to interpolate over holes in the training dataset and, therefore, reduces uncertainties in the corresponding regions [49]. While low-dimensional state-spaces are considered, sufficient data (either real or synthetic) with enough coverage is required. This will be shown and analyzed with experiments. Hence, we record data for a long period of time using random actions. Yet, such data is collected once and it is shown that a trained model can be used on other agents in the group. Analysis for the required size of data and the improvement with synthetic data will be provided. If coverage is significantly insufficient in some state regions, a stochastic model as in [50, 51] may be required. While not in the scope of this paper, such a stochastic model would provide a measure of uncertainty about state predictions.

3.3 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 [52]. As reviewed in Section 1, 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., $\log(1 - D(G(z)))$.

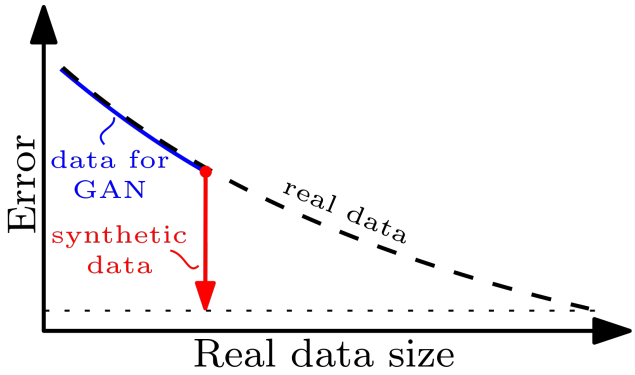


Fig. 3: Illustration of model error with regards to data size. Training an accurate model requires a large amount of real data (black dashed curve). We propose to collect a limited amount of real data (blue curve) to train a GAN, and then generate synthetic data (red arrow) from the GAN to train the model to reach the same error (dotted line).

3.4 Training with synthetic data

Acquiring an accurate model f may require a large number N of real samples in $\mathcal{T}$. Collecting real data from a robot can be expensive, slow and cause damage. Nevertheless, we hypothesize that one could acquire the same model accuracy with a smaller amount of real data for training a GAN and then generate synthetic data. In more details, a GAN is trained while requiring a limited amount of real data in $\mathcal{T}$. Then, generate synthetic data comprised of states and actions of an agent that resemble real recorded data. That is, generator G would generate joint transition data $\{(\mathbf{x}_k, \mathbf{a}_k, \mathbf{x}_{k+1})\}_{k=1}^{M}$ from the distribution of $\mathcal{Q}$. The generated data is processed to a synthetic training dataset $\mathcal{T}_s = \{(\mathbf{x}_i, \mathbf{a}_i), (\Delta\mathbf{x}_{i+1})\}_{i=1}^{M-1}$. Once the GAN is trained, increasing the size M of $\mathcal{T}_s$ is computationally cheap and does not require actual data collection from real robots. Consequently, forward dynamics model $\tilde{f}$ can be trained with a large amount of synthetic data to reach the same accuracy as with solely real data. In summary, instead of collecting a large amount of training data from the real system, we propose to collect a limited amount of real data to train a GAN which, in turn, can generate a large amount of synthetic data for training the forward dynamics model. Illustration for this can be seen in Figure 3.

3.5 Long Short Term Memory

Long Short Term Memory (LSTM) is a class of Recurrent Neural-Network (RNN) aimed to learn sequential data [53]. 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 [54]. LSTM, on the other hand, is capable of learning long-term

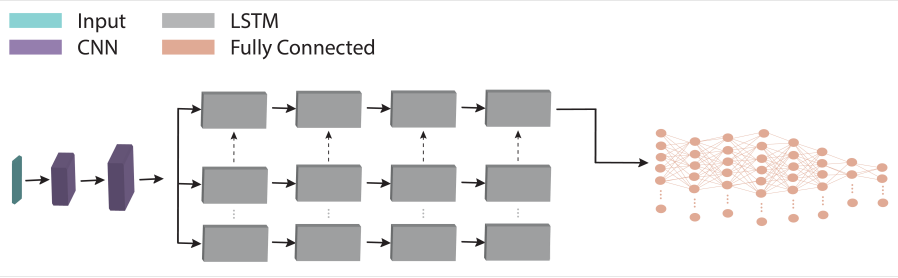


Fig. 4: The proposed CR-Net architecture to predict the next step of the agent.

dependencies by utilizing memory about previous inputs for an extended time duration [55]. 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.

3.6 CR-Net

We propose the CR-Net model aimed to estimate the forward dynamics function f of an agent. As discussed in Section 3.1, we assume an MDP system where the state change $\Delta\mathbf{x}_{i+1}$ depends on the current state $\mathbf{x}_i$ and future action $\mathbf{a}_i$ as stated in (1). In addition, the real function mapping $\mathbf{x}_i$ and $\mathbf{a}_i$ to $\Delta\mathbf{x}_{i+1}$ may be stochastic such that $\tilde{f} \sim P(\Delta\mathbf{x}_{i+1} \mid \mathbf{x}_i, \mathbf{a}_i)$. Nevertheless, we consider a deterministic model in which the most probable state change is predicted.

The CR-Net 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 4. 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 classification and segmentation [56]. A salient advantage of CNN is in its ability to automatically capture important features without human supervision. In a convolution operation, a filter is applied such that a window of constant width slides along the tabular data and study patterns. The amount of filters varies from layer to layer. Consequently, 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, the proposed CR-Net 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. The LSTM does not consider temporal data (e.g., a set of past consecutive states) but sequential feed of the features of the input vector. While the features are non-sequential [57], 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.

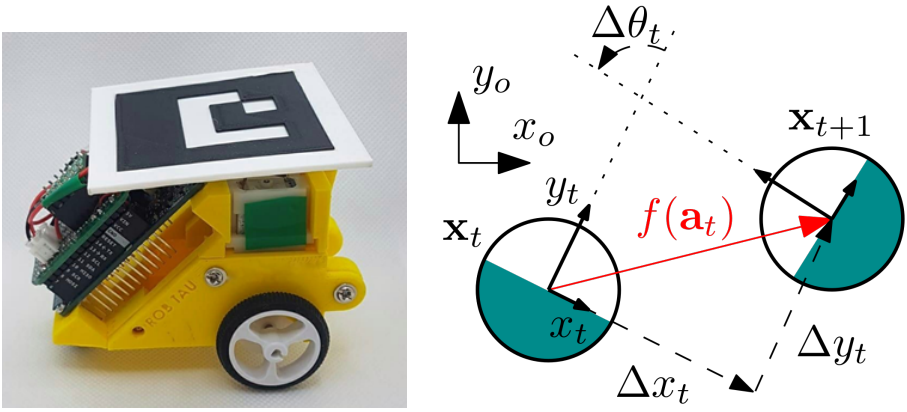
A CR-Net trained with real data $\mathcal{T}$ directly collected from the robot is denoted as *CR-Net-R*. Similarly, a CR-Net trained solely with synthetic data $\mathcal{T}_s$ sampled by the generator of the GAN is denoted as *CR-Net-S*. In addition, the CR-Net can be trained with a mix of real and synthetic data as will be demonstrated in the experiments. As discussed in Section 3.4, CR-Net-R is trained with real data from $\mathcal{T}$. On the other hand, much less data in $\mathcal{T}$ can be used to train a GAN which, in turn, would generate synthetic data $\mathcal{T}_s$ to train CR-Net-S. It is claimed that CR-Net-S would be at least as accurate as CR-Net-R while requiring much less real data in total. Experiments in the next section test this claim along with extensive analysis of the proposed approach.

4 Experiments

In this section, we test and analyze the proposed methodology and framework. We experiment on collective motion of 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 in collective motion scenarios. Videos of the experiments can be seen in the supplementary material.

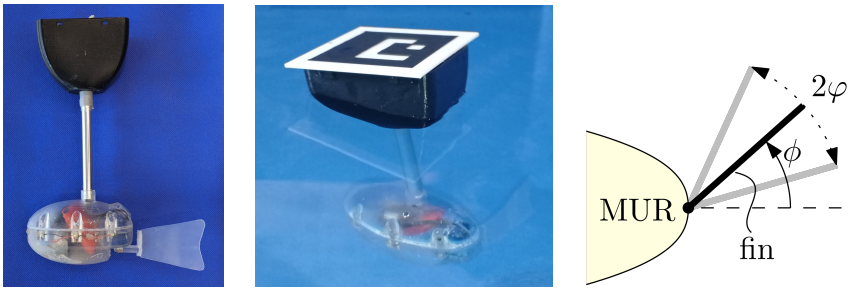
4.1 Systems

We test our approach on two micro-robot groups including the *Micro-Ground Robot (MGR)* and the *Micro-Underwater Robot (MUR)*. System CAD models and already trained GAN of the MGR are open-sourced as described in the data availability statement. Therefore, any user can build multiple robots and easily generate synthetic data for modeling them. For both MGR and MUR,



(a) (b)

Fig. 5: (a) Micro-Ground Robot (MGR) and (b) its state transition from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$.



(a) (b) (c)

Fig. 6: The (a) Micro-Underwater Robot (b) in the water and (c) a diagram of its tail-fin propulsion.

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.

4.1.1 Micro-Ground Robot (MGR)

MGR is a two wheel drive mobile robot seen in Figure 5a. 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, forward dynamics model (1) depends solely on the action $\Delta\mathbf{x}_t = f(\mathbf{a}_t)$ where $\Delta\mathbf{x}_t = (\Delta x_t, \Delta y_t, \Delta\theta_t)^T$ is the state change relative to the MGR coordinate frame at time t as seen in Figure 5b. We have built five MGR units while only one was used for data collection. Data was collected as described in Section 3.2 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 of 5,874 points was acquired from another agent. The other MGR's will further be used for RL experiments.

4.1.2 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 6a-6b. 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 6c). 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 forward dynamics 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 with 1,372 transition points were used for testing. While an analytical model for the MGR can be acquired with manual tuning by an experienced practitioner [58], an accurate analytical model for the MUR depends on a multitude of parameters [59] and, therefore, is significantly challenging to acquire. In both MGR and MUR cases, the proposed approach does not require any experience nor understanding the mechanics of the platforms.

4.2 Model training

The training set is collected as described above. The acquired training dataset is used to train either a GAN model to generate more training samples or directly the forward dynamics model. The hyper-parameters of the tested networks were optimized using the *Ray-Tune* package [60] 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. Additional optimized

Table 1: Accuracy comparison for different models trained with real or synthetic data with regards to the Micro-Ground Robot (MGR)

	Model	Position error (mm)		Orientation error (°)	
		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
	CNN	6.91±4.41	6.12±3.84	6.75±6.17	5.95±6.18
	LSTM	6.88±3.85	6.88±3.20	10.55±6.90	10.60±6.17
	GRU	9.45±5.61	7.71±4.59	19.56±14.85	15.89±13.40
	CR-Net-R	4.66±4.12	4.06±3.60	5.46±4.06	4.93±5.08
Synthetic	FC-NN	11.15±5.54	9.33±4.73	8.19±6.63	6.74±5.23
	CNN	6.73±3.32	5.75±2.8	6.84±5.57	5.59±5.25
	LSTM	6.15±4.19	5.92±3.73	7.40±5.80	6.06±5.34
	GRU	9.31±6.44	8.01±6.27	11.47±10.13	9.72±10.71
	CR-Net-S	4.53±4.23	4.06±3.53	5.23±5.56	4.82±5.14

Table 2: Accuracy comparison for different models trained with real or synthetic data with regards to the Micro-Underwater Robot (MUR)

	Model	Position error (mm)		Orientation error (°)	
		Train vehicle	Test vehicle	Train vehicle	Test vehicle
Real	FC-NN	29.59±14.62	30.26±14.61	23.64±17.08	22.77±16.57
	CNN	22.74±14.27	22.81±14.93	15.78±14.75	14.72±14.90
	LSTM	26.02±15.09	25.24±14.54	16.71±15.01	17.42±15.58
	GRU	26.69±14.58	27.65±15.20	27.65±15.20	19.77±15.74
	CR-Net-R	17.16±13.48	14.56±11.96	8.08±9.125	6.54±7.70
Synthetic	FC-NN	30.41±15.03	31.22±15.11	23.23±18.07	22.70±17.66
	CNN	22.51±14.48	22.94±14.90	16.48±15.75	15.41±15.16
	LSTM	26.32±15.25	25.38±14.75	17.37±14.96	18.11±15.58
	GRU	27.87±15.55	27.03±14.89	27.03±14.89	21.65±17.32
	CR-Net-S	18.34±13.94	16.53±13.09	9.25±10.116	6.94±8.17

hyper-parameters include an adaptive learning rate initialized at 0.005138, L2 regularization of 0.006 and batch size of $d_s = 64$ along 40 epochs.

4.3 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) [61]. All models were optimized to yield the lowest loss value and evaluated on the test data. Resulting from the optimization of the hyper-parameters, the FC-NN, CNN, LSTM, GRU and CR-Net have 126,019, 89,275, 73,879, 73,626 and 85,511 trainable parameters, respectively. In this part and when using synthetic data, all N real points (55,068 and 15,000 for MGR and MUR, respectively) in the dataset are used to train the GAN. Tables 1 and 2 show average one-step prediction errors for the MGR and MUR, respectively, over the test data. The tables show the training of the models using real data without using GAN and synthetic data generated by

Table 3: Accuracy comparison for different models trained with real data for the same learning rate (0.0001) with regards to the MGR

Model	Position error (mm)	Orientation error (°)
FC-NN	12.89±7.66	11.71±5.98
CNN	7.24±4.84	6.51±4.02
LSTM	7.43±4.04	7.03±3.92
GRU	9.82±5.72	8.03±4.76
CR-Net-R	5.09±4.88	4.91±4.15

the GAN. We include accuracy results evaluated on test data acquired from train and test vehicles. For robustness analysis, Table 3 presents comparison between the MGR models while trained with a common and arbitrary learning rate of 0.0001. Results show that CR-Net, with either real or synthetic data, outperforms all other methods. While other methods use either fully-connected, convolutional or recurrent layers, CR-Net is able to combine the benefits of them all. Hence, CR-Net is able to better embed the transition data and provide most accurate predictions.

Figure 7 compares accuracy of the CR-Net-R and CR-Net-S models, and with regards to the amount of real or synthetic training data used, respectively. For CR-Net-S, a GAN was trained with the entire real dataset. Points in the figure are averaged over 10 runs with cross-validation of different batches along the training data. Similarly, Figure 8 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. Synthetic data exhibits slightly better accuracy since the GAN inherently provides data augmentation. In practice, GAN interpolates over holes in the training dataset and, therefore, reduces uncertainties in the corresponding regions during testing [49]. In other words, GAN provides access to additional information in the real dataset which cannot be accessed directly with only a simple model [62]. Consequently, the GAN augmentation enables better coping with uncertainties of a new test vehicle.

We next justify the use of GAN with regards to other generative models. We compare the standard GAN model 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 4 reports the mean and standard deviation for position and orientation errors over the test data, 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. Further, we are interested in observing the accuracy of the CR-Net-S model with regards to the training quality of the GAN. Hence, an analysis is performed where the accuracy of the CR-Net-S is computed with regards to the number of epochs for training the GAN. For each number of epochs, the GAN was trained with the real data and then used to generate 55,000 synthetic points for training the CR-Net-S. Figure 9 presents

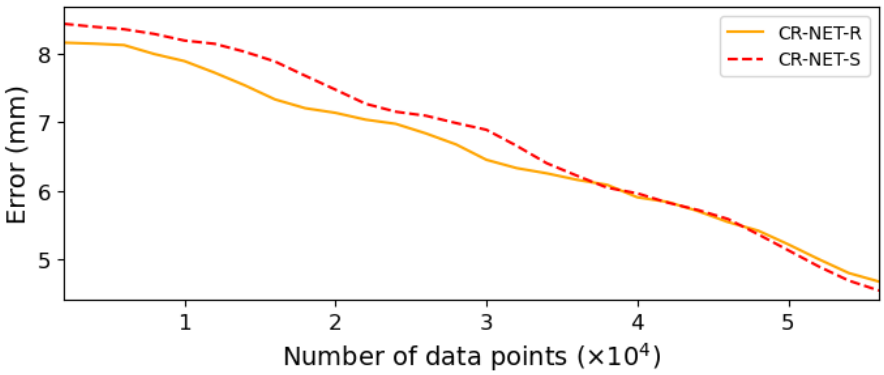


Fig. 7: MGR position accuracy for CR-Net-R and CR-Net-S when trained using real and synthetic data, respectively, with regards to data size. For CR-Net-S, synthetic data was generated from a GAN trained with the entire real dataset.

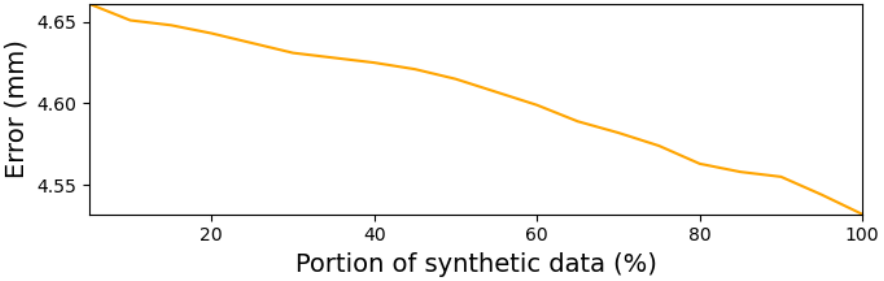


Fig. 8: MGR position accuracy for the CR-Net with regards to the ratio between real and synthetic training data.

the accuracy with regards to the number of epochs. The results show the mean and standard deviation of the positional error over five repetitions for each number of epochs. Results show the ability of the GAN and CR-Net-S to provide relatively good accuracy with an ill-trained GAN. Nevertheless, the GAN requires sufficient training (at least 17 epochs) in order to be at least equivalent to the CR-Net-R model trained with real data.

Figure 10 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 11 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

Table 4: Accuracy comparison for different generative models for the test MGR

	Position error (mm)	Orientation error (°)
DC-GAN	5.12±3.59	7.47±6.35
FCC-GAN	6.76±4.79	9.90±10.01
GMM-GAN	5.01±3.62	5.73±4.97
Cycle-GAN	4.95±3.89	5.42±5.18
GAN	4.06±3.53	4.82±5.14

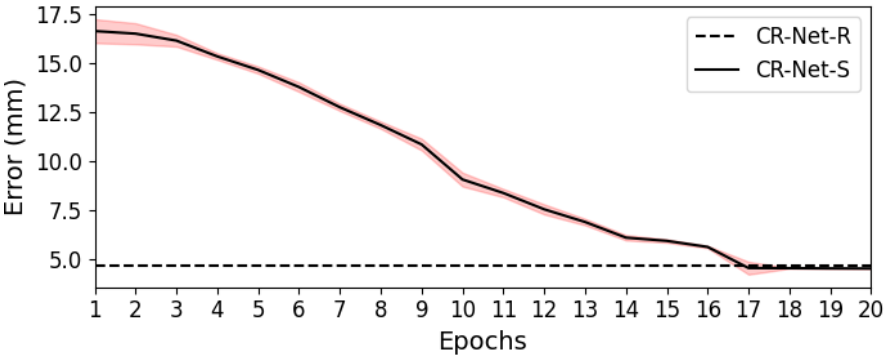


Fig. 9: MGR positional error mean (solid curve) and standard deviation (pink) of the CR-Net-S with regards to the number of training epochs for the GAN. The accuracy of the CR-Net-R is included (dashed line) for reference.

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 11), the mean position accuracy (6.00 mm) is equal to 54,000 real points (red marker). That is, the same accuracy was achieved with only 27% of the real data, or 73% less real data.For reference, we have included an additional analysis where only 27% of the real data is used for training the CR-Net-R. However, the data is augmented by adding a zero-mean normal noise $\mathcal{N}(0, \sigma^2)$ where σ is the standard deviation of the noise. Figure 12 presents the accuracy with regards to $\sigma \in [0, 5.4]$. The results show the mean and standard deviation of the positional error for five cross validations over different batches of the training data for each σ. Results show that adding noise to the training data does not improve robustness of the model nor compensate for the lack of sufficient data. The above results imply that it is 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 forward dynamics model, one can simulate the motion of an agent. Figure 13 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

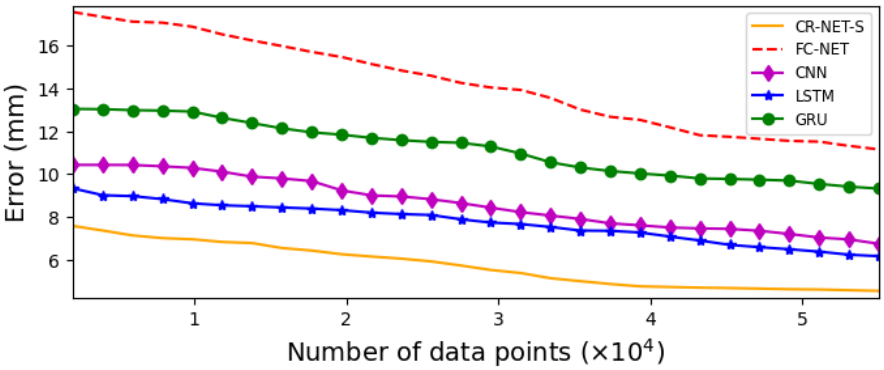


Fig. 10: Accuracy of forward dynamics models trained on synthetic data with regards to the size of data used to train the GAN.

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 for all agents while using the trained CR-Net-S for each is seen in Figure 1.

4.4 RL Evaluation for a Single Agent

Prior to testing collective motion, we evaluate the performance of the data-based MGR simulator for training single-agent RL policies. We set a scenario in which one MGR agent must reach a target goal while avoiding collision with a circular obstacle (Figure 14). We have implemented the Deep Deterministic Policy Gradient (DDPG) [63]. DDPG is a model-free RL algorithm which will use the data-based simulator to train a policy for the agent. It is implemented with a reward function that rewards for reaching the target and penalizes distance from the target or collision. Two policies were trained in simulation using trained models presented in previous section, one based on FC-NN and the other on CR-Net-S. Table 5 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 encapsulate the real system well. On the other hand, CR-Net exhibits a good ability to simulate the real world with fairly matching simulated and real rewards, and high success rate.

Fig. 11: 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.

4.5 MARL Evaluation

We next evaluate the performance of the data-based simulators for training MARL policies in collective motion. A decentralized MARL scenario is considered with $K = 5$ MGR agents. The agents are arranged in some arbitrary initial formation and must reach a target marker while avoiding collisions with other neighboring agents. A trained forward dynamics 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. While training a decentralized policy, it is assumed that each agent has global knowledge about the state of all participating agents. Note that global knowledge is a choice for the demonstration while local sensing can also be integrated. Hence, the reward function $r : \mathcal{C} \times \mathcal{A} \rightarrow \mathbb{R}$ for each agent is defined to be

$$r(\mathbf{x}, \mathbf{a}) = r_t + \sum_{i=1}^{K-1} r_i, \quad \text{for } r_t = \begin{cases} -5, & \text{if } d_t < 150 \\ -20\frac{d_t}{d_{\max}}, & \text{otherwise,} \end{cases} \tag{3}$$

where $d_t = \|\mathbf{x} - \mathbf{x}_T\|$ is the agent's distance from target $\mathbf{x}_T$ and $d_{max} = 1,400\ mm$ is some user-defined constant. The r_t reward component drives the agent towards the target while avoiding collision with it. Hence, the agent

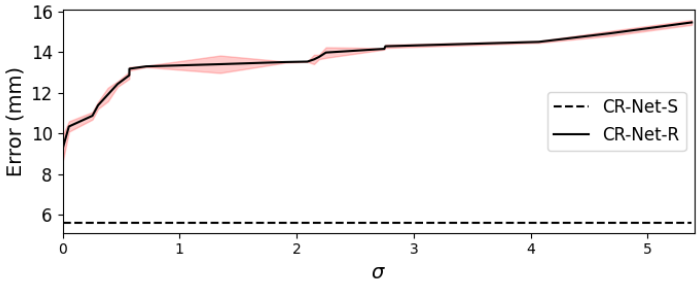


Fig. 12: MGR positional error mean (solid curve) and standard deviation (pink) of the CR-Net-R with regards to the standard deviation σ of noise added to the training data. In this case, only 27% of the training data is used. The accuracy of the CR-Net-S with the same amount of data (as seen in Figure 11) is included (dashed line) for reference.

Table 5: 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%

should reach up to 150mm from it. 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. Hence, mismatch errors between the simulated and real motion lead to complete failure.

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 (75 seconds) 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 6 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.

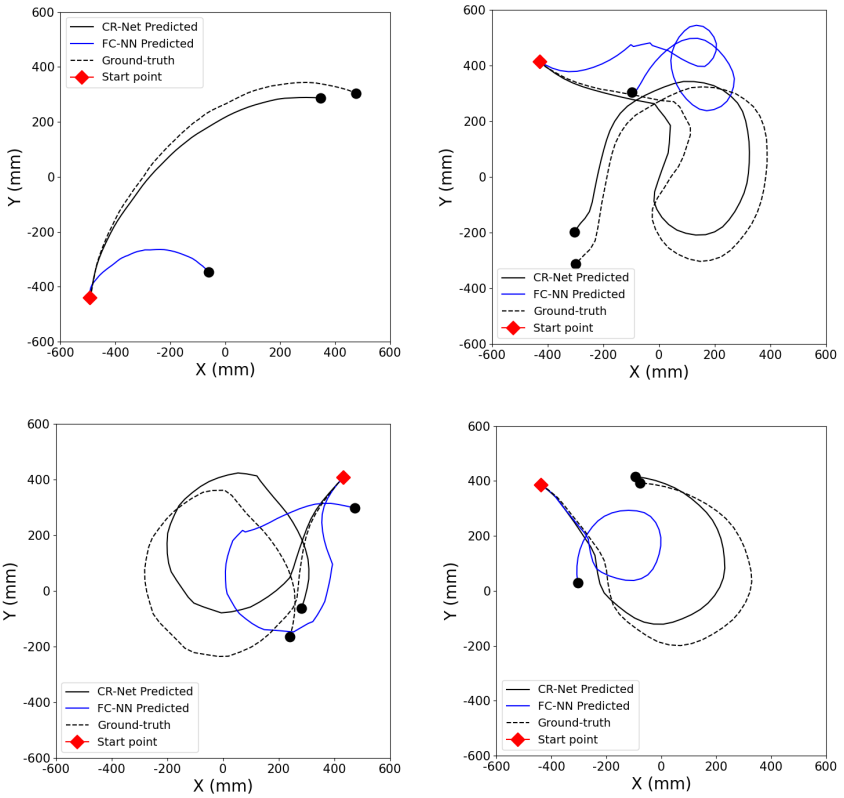


Fig. 13: Open-loop prediction of MGR trajectories based solely on the initial state and sequence of actions, while using CR-Net-S and FC-NN. The tracking RMSE of the CR-Net-S trajectories is, from left to right, 40.4 *mm*, 59.9 *mm*, 70.1 *mm* and 19.8 *mm*. In contrast, the RMSE of the FC-NN is 562.4 *mm*, 458.1 *mm*, 366.8 *mm* and 304.0 *mm*.

Hence, the simulator sufficiently represents the real world. Figures 15 and 16 show two examples of policy roll-outs 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-S is applicable and sufficiently accurate for all agents in the group.

Roll-out demonstration was also performed for the MUR agents. Here also, DDPG is used with a shared policy for all agents. A policy was trained in simulation to reach target locations in the pool. The rewards function was similar to (3) while with $d_{max} = 2,000$ *mm* to reflect the size of the pool. Snapshots of the motion between two targets are seen in Figure 17.

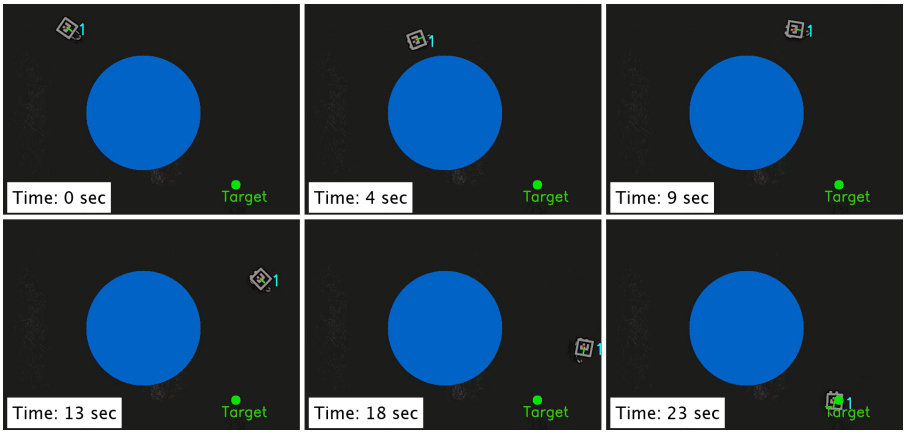


Fig. 14: An MGR agent avoiding an obstacle (blue circle) and reaching a target using an RL policy trained on a simulation based on CR-Net-S.

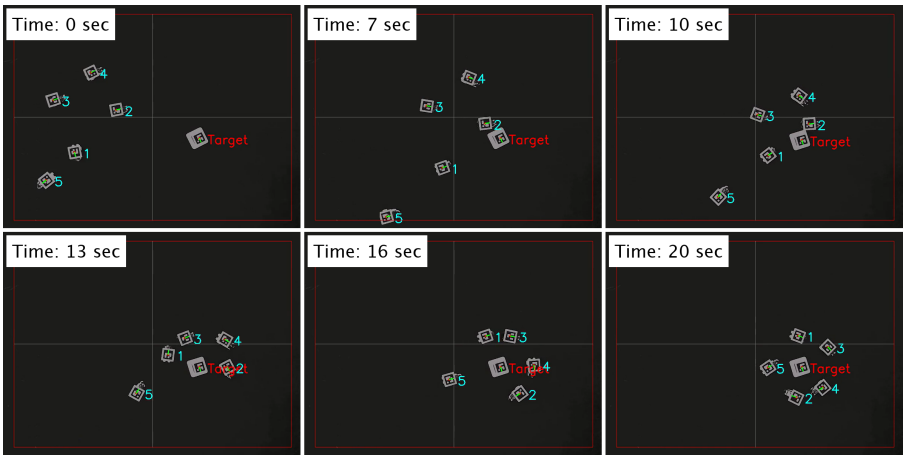


Fig. 15: A set of five MGR agents reaching a target using a trained policy while maintaining distance among them. The policy was trained on a simulation based on CR-Net-S.

5 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, a trained CR-Net can accurately predict motion for all agents in the group. Furthermore, we examined the influence of training the CR-Net with synthetic data generated by a GAN. Results

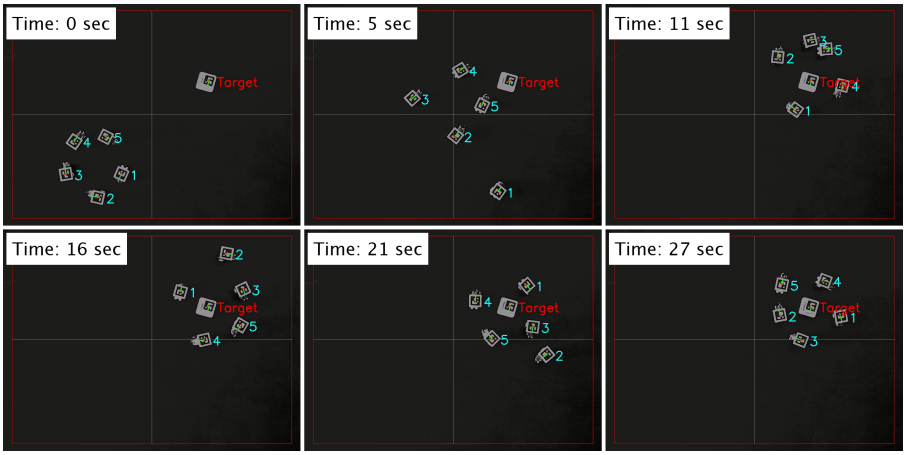


Fig. 16: A set of five MGR agents reaching a target using a trained policy while maintaining distance among them. The policy was trained on a simulation based on CR-Net-S.

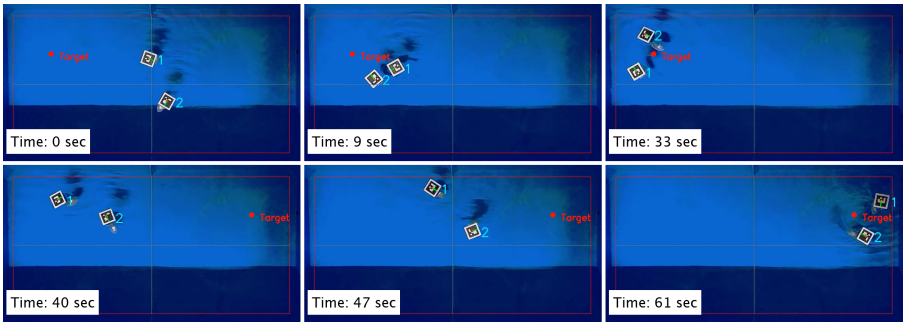


Fig. 17: Two MUR agents move in water between two targets using a trained policy while keeping minimal distance among them. The policy was trained on a simulation based on CR-Net-S.

Table 6: 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

show that CR-Net along with synthetic data from GAN can achieve high accuracy with significantly less 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. Future work may observe the performance of the proposed approach to more complex systems of higher-dimension states. In addition, additional work may consider building a stochastic model that encodes the uncertainty and differences between the vehicles.

Data availability. The datasets and models generated during the current study are available in the Git repository, https://github.com/eranbTAU/Closing-the-Reality.

Supplementary information. Supplementary material is attached to the submission of this manuscript.

Acknowledgments. This research was supported by the Zimin Institute for Engineering Solutions Advancing Better Lives.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

[1] Foerster, J.N.: Deep multi-agent reinforcement learning. PhD thesis, University of Oxford (2018)

[2] Zhang, K., Yang, Z., Başar, T.: In: Vamvoudakis, K.G., Wan, Y., Lewis, F.L., Cansever, D. (eds.) Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms, pp. 321–384. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-60990-0_12

[3] Gupta, J.K., Egorov, M., Kochenderfer, M.: Cooperative multi-agent control using deep reinforcement learning. In: Inter. Conf. on Autonomous Agents and Multiagent Systems, pp. 66–83 (2017)

[4] Hüttenrauch, M., Sosic, A., Neumann, G.: Guided deep reinforcement learning for swarm systems. CoRR **abs/1709.06011** (2017)

[5] Yasuda, T., Ohkura, K.: Sharing experience for behavior generation of real swarm robot systems using deep reinforcement learning. Jour. of Robotics and Mechatronics **31**(4), 520–525 (2019)

[6] Billah, M.A., Faruque, I.A.: Bioinspired visuomotor feedback in a multiagent group/swarm context. IEEE Transactions on Robotics **37**(2), 603–614 (2021)

[7] Lim, V., Huang, H., Chen, L.Y., Wang, J., Ichnowski, J., Seita, D., Laskey, M., Goldberg, K.: Planar robot casting with real2sim2real self-supervised learning. CoRR (2021)

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

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

[10] Azulay, O., Shapiro, A.: Wheel loader scooping controller using deep reinforcement learning. IEEE Access, 24145–24154 (2021)

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

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

[13] Yu, J., Han, S.D., Tang, W.N., Rus, D.: A portable, 3d-printing enabled multi-vehicle platform for robotics research and education. In: IEEE Inter. Conf. on Robotics and Automation, pp. 1475–1480 (2017). https://doi.org/10.1109/ICRA.2017.7989176

[14] Nguyen-Tuong, D., Peters, J.: Model learning for robot control: a survey. Cognitive processing **12**(4), 319–340 (2011)

[15] Hahn, D., Banzet, P., Bern, J.M., Coros, S.: Real2sim: Visco-elastic parameter estimation from dynamic motion. ACM Transactions on Graphics (TOG) **38**(6), 1–13 (2019)

[16] Jordan, M.I., Rumelhart, D.E.: Forward models: Supervised learning with a distal teacher. Cognitive science **16**(3), 307–354 (1992)

[17] Sintov, A., Morgan, A.S., Kimmel, A., Dollar, A.M., Bekris, K.E., Boularias, A.: Learning a state transition model of an underactuated adaptive hand. IEEE Robotics and Automation Letters **4**(2), 1287–1294 (2019)

[18] Sun, D., Chen, J., Mitra, S., Fan, C.: Multi-agent motion planning from signal temporal logic specifications. IEEE Robotics and Automation Letters **7**(2), 3451–3458 (2022). https://doi.org/10.1109/LRA.2022.3146951

[19] Dai, L., Cao, Q., Xia, Y., Gao, Y.: Distributed mpc for formation of multi-agent systems with collision avoidance and obstacle avoidance. Journal of the Franklin Institute **354**(4), 2068–2085 (2017). https://doi.org/10.

1016/j.jfranklin.2016.12.021

[20] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Proceedings of the International Conference on Neural Information Processing Systems, vol. 2, pp. 2672–2680. MIT Press, ??? (2014)

[21] Frid-Adar, M., Diamant, I., Klang, E., Amitai, M., Goldberger, J., Greenspan, H.: Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. Neurocomputing **321**, 321–331 (2018). https://doi.org/10.1016/j.neucom.2018.09.013

[22] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., Vanhoucke, V.: Using simulation and domain adaptation to improve efficiency of deep robotic grasping. IEEE International Conference on Robotics and Automation (ICRA), 4243–4250 (2018)

[23] Zhang, K., Yang, Z., Liu, H., Zhang, T., Başar, T.: Fully decentralized multi-agent reinforcement learning with networked agents. In: Inter. Conf. on Machine Learning, vol. 80, pp. 5872–5881 (2018)

[24] Zheng, H., Shi, D.: A multi-agent system for environmental monitoring using boolean networks and reinforcement learning. Journal of Cyber Security **2**, 85–96 (2020)

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

[26] Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence **7**, 1–41 (2012)

[27] Rossi, F., Bandyopadhyay, S., Wolf, M., Pavone, M.: Review of multi-agent algorithms for collective behavior: a structural taxonomy. IFAC-PapersOnLine **51**(12), 112–117 (2018). https://doi.org/10.1016/j.ifacol.2018.07.097. IFAC Workshop on Networked & Autonomous Air & Space Systems NAASS 2018

[28] Xuan, P., Lesser, V.: Multi-agent policies: From centralized ones to decentralized ones. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 3. AAMAS '02, pp. 1098–1105. Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/545056.545078

[29] Zhang, Q., Lu, C., Garg, A., Foerster, J.: Centralized model and exploration policy for multi-agent rl. In: Proceedings of the 21st International

Conference on Autonomous Agents and Multiagent Systems, pp. 1500–1508. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2022)

[30] Gronauer, S., Diepold, K.: Multi-agent deep reinforcement learning: A survey. Artif. Intell. Rev. **55**(2), 895–943 (2022). https://doi.org/10.1007/s10462-021-09996-w

[31] Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J.Z., Tuyls, K., Graepel, T.: Value-decomposition networks for cooperative multi-agent learning based on team reward. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '18, pp. 2085–2087 (2018)

[32] Chamanbaz, M., Mateo, D., Zoss, B.M., Tokić, G., Wilhelm, E., Bouffanais, R., Yue, D.K.P.: Swarm-enabling technology for multi-robot systems. Frontiers in Robotics and AI **4** (2017)

[33] Ribeiro, R., Silvestre, D., Silvestre, C.: Decentralized control for multi-agent missions based on flocking rules. In: CONTROLO 2020, pp. 445–454 (2021)

[34] Mishra, R.K., Vasal, D., Vishwanath, S.: Decentralized multi-agent reinforcement learning with shared actions. In: Annual Conference on Information Sciences and Systems (CISS), pp. 1–6 (2021)

[35] Dobbe, R., Fridovich-Keil, D., Tomlin, C.: Fully decentralized policies for multi-agent systems: An information theoretic approach. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. NIPS'17, pp. 2945–2954. Curran Associates Inc., Red Hook, NY, USA (2017)

[36] Gupta, J.K., Egorov, M., Kochenderfer, M.: Cooperative multi-agent control using deep reinforcement learning. In: International Conference on Autonomous Agents and Multiagent Systems, pp. 66–83 (2017). Springer

[37] Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: The use of simulation in evolutionary robotics. In: European Conference on Artificial Life, pp. 704–720 (1995). Springer

[38] Kaspar, M., Osorio, J.D.M., Bock, J.: Sim2real transfer for reinforcement learning without dynamics randomization. IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems, 4383–4388 (2020)

[39] Golemo, F.: How to train your robot-new environments for robotic training and new methods for transferring policies from the simulator to the

real robot. PhD thesis, Université de Bordeaux (2018)

[40] Dearden, A., Demiris, Y.: Learning forward models for robots. In: IJCAI, vol. 5, p. 1440 (2005)

[41] Ruthotto, L., Haber, E.: An introduction to deep generative modeling. GAMM-Mitteilungen **44**(2), 202100008 (2021)

[42] GM, H., Gourisaria, M.K., Pandey, M., Rautaray, S.S.: A comprehensive survey and analysis of generative models in machine learning. Computer Science Review **38**, 100285 (2020)

[43] Tran, N.-T., Tran, V.-H., Nguyen, N.-B., Nguyen, T.-K., Cheung, N.-M.: On data augmentation for GAN training. IEEE Transactions on Image Processing **30**, 1882–1897 (2021)

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

[45] Golany, T., Freedman, D., Radinsky, K.: 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, pp. 134–141 (2021)

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

[47] Xu, T., Wenliang, L.K., Munn, M., Acciaio, B.: Cot-gan: Generating sequential data via causal optimal transport. In: Advances in Neural Information Processing Systems, vol. abs/2006.08571 (2020)

[48] Klemmer, K., Xu, T., Acciaio, B., Neill, D.B.: Spate-gan: Improved generative modeling of dynamic spatio-temporal patterns with an autoregressive embedding loss. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, pp. 4523–4531 (2022)

[49] Sampath, V., Maurtua, I., Aguilar, J., Gutierrez, A.: A survey on generative adversarial networks for imbalance problems in computer vision tasks. Journal of Big Data **8**(27) (2021)

[50] Sintov, A., Morgan, A.S., Kimmel, A., Dollar, A.M., Bekris, K.E., Boularias, A.: Learning a state transition model of an underactuated adaptive hand. IEEE Robotics and Automation Letters **4**(2), 1287–1294 (2019)

[51] Kimmel*, A., Sintov*, A., Wen, B., Boularias, A., Bekris, K.: Belief-space

planning using learned models with application to underactuated hands. In: Proc. of the 2019 International Symposium on Robotics Research, Hanoi, Vietnam (2019)

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

[53] Yu, Y., Si, X., Hu, C., Zhang, J.: A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. Neural Computation **31**(7), 1235–1270 (2019)

[54] Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks **5**(2), 157–166 (1994)

[55] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)

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

[57] Chen, Y., Yang, J., Qian, J.: Recurrent neural network for facial landmark detection. Neurocomputing **219**, 26–38 (2017)

[58] Malu, K., Majumdar, J., Sandeep: Kinematics, localization and control of differential drive mobile robot. Global Journal of Research In Engineering **14** (2014)

[59] Wang, W., Dai, X., Li, L., Gheneti, B.H., Ding, Y., Yu, J., Xie, G.: Three-dimensional modeling of a fin-actuated robotic fish with multimodal swimming. IEEE/ASME Transactions on Mechatronics **23**(4), 1641–1652 (2018). https://doi.org/10.1109/TMECH.2018.2848220

[60] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J.E., Stoica, I.: Tune: A research platform for distributed model selection and training. arXiv preprint arXiv:1807.05118 (2018)

[61] Dey, R., Salem, F.M.: Gate-variants of gated recurrent unit (GRU) neural networks. In: IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 1597–1600 (2017). https://doi.org/10.1109/MWSCAS.2017.8053243

[62] Bowles, C., Chen, L., Guerrero, R., Bentley, P., Gunn, R.N., Hammers, A., Dickie, D.A., del C. Valdés Hernández, M., Wardlaw, J.M., Rueckert, D.: GAN augmentation: Augmenting training data using generative

adversarial networks. In: CoRR, vol. abs/1810.10863 (2018)

[63] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N.M.O., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: CoRR, vol. abs/1509.02971 (2016)