Sim2Real Robot Learning

Pieris Panagi
University of Cyprus
ppanag02@ucy.ac.cy

Matteo Rossi Reich
University of Bologna
matteo.rossireich@studio.unibo.it

Giuseppe Tanzi
University of Bologna
giuseppe.tanzi@studio.unibo.it

Abstract

Sim2Real robot learning is an innovative field that aims to train robots in simulated environments and transfer their acquired skills to real-world applications. By combining simulation, machine learning, and robotics, Sim2Real robot learning tackles the challenges associated with training in real-world scenarios. This comprehensive report explores the core concepts and methodologies of Sim2Real robot learning, covering essential topics such as reinforcement learning, evolutionary robotics, simulators, domain adaptation, domain randomization, and meta learning. It delves into the advantages and limitations of simulation-based training, addressing the reality gap between simulations and real-world environments. The report highlights the importance of domain adaptation and domain randomization techniques in minimizing the discrepancies between simulated and real-world performance.

1 Introduction

In recent years, the field of robotics has witnessed remarkable advancements, fueling the aspiration to deploy autonomous systems in real-world scenarios across various domains. However, achieving reliable and robust robot behavior in complex and dynamic environments poses a significant challenge. While Reinforcement Learning (RL) has emerged as a powerful technique for training intelligent agents, applying RL directly to physical robots can be impractical due to safety concerns, high costs, and potential damage to the robots or their environment. To overcome these limitations, researchers have turned to Sim2Real Robot Learning, an interdisciplinary field that leverages simulated training environments and simulators to bridge the gap between simulation and reality.

Sim2Real Robot Learning involves training robots in simulated environments and effectively transferring their acquired knowledge and policies to real-world settings. By harnessing the power of data-driven techniques such as RL, Evolutionary Robotics (ER), and Meta Learning, researchers can develop intelligent control strategies for robotic systems. Simulated environments provide a safe and controlled testing ground, enabling researchers to iterate rapidly, explore diverse scenarios, and collect extensive data for training and evaluation purposes.

The use of simulations and simulators offers several advantages in the development of robotic systems. Firstly, simulations provide a controlled environment where robots can learn and explore without the risks associated with real-world deployment. Secondly, simulations allow for the generation of large amounts of training data, facilitating the training of complex behaviors and enhancing the learning capabilities of the robots. Furthermore, simulations enable researchers to iterate efficiently on algorithmic approaches, architectural designs, and hyperparameter settings, reducing the time and costs associated with experimentation on physical systems.

However, a challenge arises when transferring the learned policies from simulation to reality. Simulated environments, though valuable for training, often fail to perfectly replicate the complexities and nuances of the real world. This discrepancy between simulation and reality is known as the reality gap. The reality gap poses a significant hurdle as policies trained solely in simulation may struggle to generalize effectively to real-world scenarios.

To address the reality gap challenge, researchers have developed techniques such as domain adaptation and domain randomization. Domain adaptation aims to minimize the gap between simulation and reality by adapting the learned policies from simulation to the real-world domain. This technique leverages limited real-world data to fine-tune or retrain policies, improving their generalization capabilities and enhancing the transferability of learned knowledge. On the other hand, domain ran-

domization introduces variations and perturbations into the simulated environment, exposing robots to a wide range of conditions and preparing them for the uncertainties encountered in real-world settings.

In this report, we provide a comprehensive exploration of Sim2Real Robot Learning, encompassing its core concepts, methodologies, and challenges. The remaining sections of this paper are organized as follows. In Section 2, we provide an overview of Reinforcement Learning (RL) and its key concepts for training intelligent agents. Section 3 focuses on Sim2Real Robot Learning using evolutionary algorithms. We introduce the concept of Evolutionary Robotics (ER) and briefly discuss its relevance in developing adaptive robot controllers. Moving on to Section 4, we explore the role of simulators in training and evaluating robotic systems. We compare different simulators and highlight their benefits and limitations. In Section 5, we delve into Sim2Real transfer techniques, specifically domain adaptation and domain randomization. We discuss how these techniques can bridge the gap between simulation and real-world environments. Section 6 introduces Meta Learning and its potential applications in Sim2Real Robot Learning. Finally, in Section 7, we conclude the report by summarizing our findings and discussing potential challenges and future directions in Sim2Real Robot Learning.

2 Reinforcement Learning

Reinforcement learning (RL) is inspired by the idea of learning by interacting with the environment (Sutton and Barto, 2018). It can be motivated by looking at how we learn by drawing connections between the actions we do and the environment's responses. This mapping allows us to understand which actions should be performed to reach a certain goal. In a similar manner an agent trained with RL methods must learn how to map its actions to a future reward signal associated with successfully completing a goal. This learning paradigm is different from both Supervised and Unsupervised learning, both try to find connection structures in labeled or unlabeled data, while RL tries to maximize a reward signal through a combination of environment exploration and exploitation of known successful actions. This problem of learning is formalized using the framework of the Markov Decision Processes (MDPs) for sequential decision making where actions influence not only the immediate reward but also the long term one. In a MDP setting a reinforcement leaning problem is characterized by three signals between an agent and its environment. At each time step t the agent performs an action $A_t \in A$, the environment then returns a new state $S_{t+1} \in S$, which is a description of the environment, and a reward signal R_{t+1}. The *reward hypothesis* defines that the goal of an agent can be described as the maximization of the expected value of the reward signal. Crafting a reward signal which leads the agent to the desired goal is part of the challenge of creating an effective RL system. For later reference the concept of *transition dynamics* of a finite MDP is given as:

$$p(s', r|s, a) = Pr\{S_t = s', R_t|S_{t-1}, A_{t-1} = a\}$$

it states that the probability of each state and reward depends only on the previous state and action.

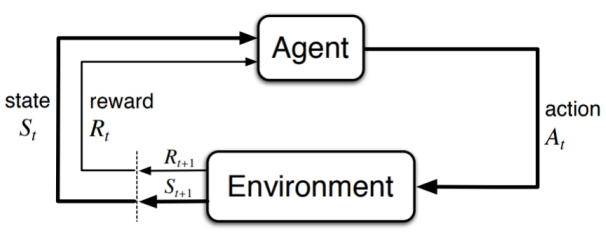

Figure 1: The agent-environment interaction. (Sutton and Barto, 2018)

3 Evolutionary Robotics

Evolutionary Robotics is a subfield of robotics that draws inspiration from the principles of biological evolution to design and optimize robot behaviors and morphologies (Nolfi and Floreano, 2000). It combines concepts from evolutionary computation and artificial intelligence to develop adaptive and autonomous robotic systems. By leveraging evolutionary algorithms, researchers in Evolutionary Robotics aim to create robots that can learn and adapt to their environment without explicit programming or human intervention. Evolutionary Robotics approaches the design of robot controllers and physical attributes as an optimization problem (Doncieux et al., 2015). It starts with a population of robots, each having different sets of parameters or neural architectures. These robots are then subjected to a simulated or real-world environment where they perform tasks or interact with their surroundings. Through a process of selection, reproduction, and mutation, the best-performing individuals are chosen to create the next generation

of robots, gradually improving their performance over successive generations. This iterative process mimics the mechanisms of natural selection, allowing robots to evolve and adapt to their specific tasks and environmental conditions.

3.1 The NEAT Algorithm

One influential approach in Evolutionary Robotics is NEAT (NeuroEvolution of Augmenting Topologies), proposed by (Stanley and Miikkulainen, 2002). NEAT revolutionized the field by introducing a method to evolve artificial neural networks with varying topologies, enabling the evolution of both the network structure and the connection weights. Traditionally, fixed-topology neural networks were used in evolutionary algorithms, limiting the complexity and adaptability of evolved behaviors. However, NEAT addressed this limitation by allowing the evolution of neural networks with different numbers of neurons and connections. This approach enables the evolution of more complex and diverse neural architectures, which can capture and exploit different strategies and behaviors. The key innovation of NEAT lies in its ability to perform both historical marking and structural innovation. Historical marking assigns each gene in a genome a historical marking that represents its age and origin. This marking ensures that crossover and mutation operations do not disrupt existing structures, allowing the preservation of promising innovations. Structural innovation, on the other hand, allows the addition of new nodes and connections during evolution, promoting the emergence of novel and more efficient neural network topologies. By evolving neural networks using NEAT, Evolutionary Robotics researchers can overcome the limitations of handcrafted control systems. Instead of relying on human expertise and manual design, the evolutionary process enables the automatic discovery and optimization of neural architectures that can efficiently solve complex robotic tasks.
By leveraging NEAT in simulation, researchers can evolve neural network controllers for robots in a virtual environment, allowing them to learn and adapt without the need for extensive human intervention or costly real-world experimentation.
The use of NEAT in Sim2Real Robot Learning offers several advantages. Firstly, it enables rapid exploration of the design space, facilitating the discovery of effective robot behaviors and neural architectures. The ability to evolve both the structure and weights of neural networks allows for a richer search for optimal solutions. Secondly, NEAT provides a flexible and adaptive approach that can handle complex, high-dimensional problems, enabling the optimization of robot control for a wide range of tasks and environments. Furthermore, NEAT overcomes the limitations of fixed-topology neural networks by promoting innovation and the emergence of novel solutions. By allowing the addition and removal of connections and nodes, NEAT can adapt the network architecture to match the requirements of the task and environment. This flexibility is particularly valuable in Sim2Real Robot Learning, where the challenge lies in transferring simulated policies to real-world robots that operate in dynamic and unpredictable environments.
NEAT, as a pioneering approach in Evolutionary Robotics, has significantly advanced the field by enabling the evolution of artificial neural networks with varying topologies. Its integration into Sim2Real Robot Learning provides a powerful framework for optimizing robot behaviors in simulation, which can then be transferred to physical robots with enhanced adaptability and autonomy.

4 Simulators

4.1 Gazebo

(Koenig and Howard, 2004) describe the Gazebo project, a high fidelity outdoor dynamics simulator designed to accurately reproduce the dynamic environments that robots may encounter. Gazebo is a completely open-source and freely available simulator that offers a rich environment to quickly develop and test multi-robot systems in new and interesting ways.
The development of Gazebo was driven by the increasing use of robotic vehicles for outdoor applications, which highlighted the need for a simulator capable of modeling outdoor environments and providing realistic sensor feedback. All simulated objects in Gazebo have mass, velocity, friction, and numerous other attributes that allow them to behave realistically when pushed, pulled, knocked over, or carried.
Almost every aspect of the simulation in Gazebo is controllable, from lighting conditions to friction coefficients. Gazebo is developed from the ground up to be fully compatible with the Player device server, and the hardware simulated in Gazebo is designed to accurately reflect the behavior of its physical counterpart.

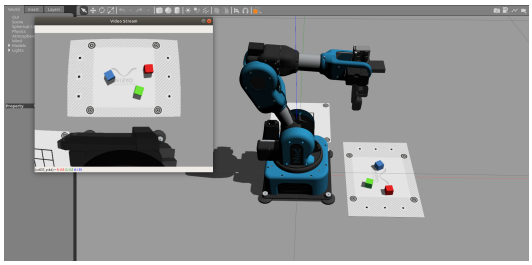

Figure 2: A robotic arm in Gazebo

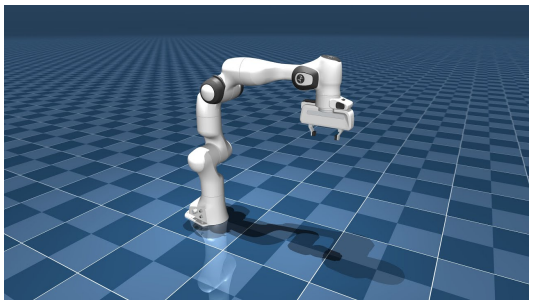

Figure 3: A robotic arm in MuJoCo

However, the complexity of simulating rigid body dynamics coupled with a 3D environment can severely tax even a high-performance computer, limiting Gazebo to the domain of a few robots, currently on the order of ten. Additionally, Gazebo has limitations in simulating pliable surfaces such as soil, sand, and grass that are normally found in nature.
One of the more useful features missing from Gazebo is distributed computation, which would greatly benefit the amount of computation involved while running such an intensive simulation. Despite these limitations, Gazebo remains a valuable tool for developers of multi-robot systems.

4.2 MuJoCo

According to (Todorov et al., 2012), the essence of control optimization involves the automatic construction of multiple candidate controllers and evaluating their performance in simulation. This is achieved through either sampling, as in evolutionary algorithms or reinforcement learning, or through gradient information obtained via finite differencing. Due to the complexity of a robot's dynamics, which cannot be differentiated analytically, optimization requires numerous dynamics evaluations for various states and controls.
Accuracy and stability are essential in control optimization. While approximations used in gaming engines may be justifiable, they cannot meet the demanding requirements for speed and accuracy in control optimization. Existing physics engines such as ODE, NVIDIA PhysX, and Bullet Physics represent the system state in overcomplete Cartesian coordinates and enforce joint constraints numerically, which can be inaccurate and inefficient when simulating elaborate multi-joint systems such as humanoids. Engines such as SD/FAST and OpenSim perform all computations in joint coordinates and do not need to enforce joint constraints numerically, but they either ignore contacts or use spring-dampers, resulting in large penetrations or stiff dynamics. This limits their applications to robotics where contact dynamics are key.
To address these limitations, the developers of MuJoCo (Multi-Joint dynamics with Contact) created a new engine that represents the state in joint coordinates and simulates contacts related to LCP (Linear Complementarity Problem) but better. They implemented several new formulations of the physics of contact in MuJoCo, providing multiple mechanisms for modeling contact dynamics, allowing the user to select the one most suitable for a given system.
MuJoCo has several unique features that greatly facilitate control applications, such as the ability to evaluate the same dynamical system in parallel for different states and controls, enabling numerical optimization through finite differencing. Additionally, inverse dynamics can always be computed even in the presence of contacts and equality constraints, which is useful for the analysis of recorded data and computed torque control applications. MuJoCo can also model actuator dynamics, including pressures inside pneumatic or hydraulic cylinders and the activations of biological muscles, and their ability to transmit forces via linkages or tendons, which can wrap around 3D shapes and have hard length limits. Users can access MuJoCo's functionality through various means, such as a standalone executable, interfaces with MATLAB and C library.

4.3 Isaac Gym

(Makoviychuk et al., 2021) discuss the importance of reinforcement learning (RL) in machine learning and its potential for solving complex decision-making problems. It highlights the use of popular physics engines such as MuJoCo, PyBullet, DART, Drake, and V-Rep for training robots to solve challenging RL tasks. However, due to the high computational requirements of these tasks, large CPU

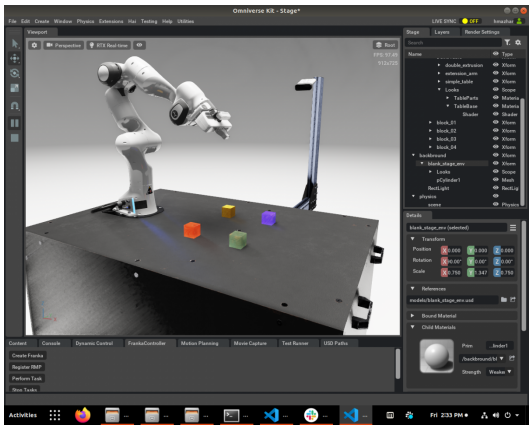

Figure 4: A robotic arm in Isaac Sim

clusters are needed. The paper also mentions the use of GPUs for highly parallel simulations and their success in computer graphics. However, there are still bottlenecks that need to be addressed, such as the simulation being on the GPU, but the physics state being copied back to the CPU.

To overcome these bottlenecks, they introduce Isaac Gym, an end-to-end high-performance robotics simulation platform that runs an end-to-end GPU-accelerated training pipeline. This allows researchers to achieve 2-3 orders of magnitude of training speed-up in continuous control tasks, and it leverages NVIDIA PhysX to provide a GPU-accelerated simulation back-end. It also provides a PyTorch tensor-based API to access the results of physics simulation natively on the GPU. Observation tensors can be used as inputs to a policy network, and the resulting action tensors can be directly fed back into the physics system. The authors note that others have recently begun attempting a similar approach to Isaac Gym for running end-to-end training on hardware accelerators. Lastly, Isaac Gym provides a straightforward API for creating and populating a scene with robots and objects, supporting loading data from the common URDF and MJCF file formats.

4.4 Comparison

(Collins et al., 2021b) discuss the landscape of simulation options for researchers working in robotics. The field of simulation is constantly changing as new simulators are developed to keep up with the latest research trends, while others become outdated. The authors of the review have identified seven sub-domains that cover the majority of robotics work involving simulators. For each sub-domain, they discuss current challenges in the field, necessary capabilities of robotics simulators used in the field, and existing approaches to robot simulation in the literature of the sub-domain. Finally, they provide a summarizing table in each section that details the capabilities of numerous robotics simulators in features relevant to robotics simulation in the sub-domain. Overall, the article provides an overview of the challenges and opportunities researchers face when selecting and using simulation tools in robotics.

4.4.1 Mobile Ground Robotics

According to (Rubio et al., 2019) autonomous ground vehicle research, which covers legged, wheeled and tracked robots, is one of the largest domains in robotics. The sub-domain incorporates various fields, such as navigation, locomotion, cognition, control, perception, and Simultaneous Localisation and Mapping (SLAM). Gazebo, one of the most popular simulators for mobile ground robot research, is used for both legged (Bellicoso et al., 2018) and wheeled (Takaya et al., 2016) robots. Gazebo offers a model library of commonly used sensors, including camera, GPS, and IMU, and allows the import of environments from digital elevation models, SDF meshes, and OpenStreetMap. It also provides the Robot Operating System (ROS) interface, making it easier to test control software in simulation and transfer it to the physical system.

As demonstrated by (Rath et al., 2018) CoppeliaSim is another popular simulator for simulating ground-based mobile robots, which offers a large model library of common mobile robot platforms and sensors, including 2D/3D laser, accelerometer, stereocamera, camera, event camera, GPS, and gyro. CoppeliaSim also provides path planning functionality through the commonly used OMPL library and supports height-fields for terrain specification. Webots, which has a large model database of mobile robots, environments, and sensors, is another popular choice, and is used in state-of-the-art research to investigate non-holonomic robot trajectory tracking and evolve bipedal robot gaits.

Raisim, developed by ETH Zurich, is a physics simulator used in research into learning dynamic policies for legged platforms, as indicated by (Hwa). PyBullet (Tan et al., 2018), which supports rigid-body simulation and faster-than-real-time simulation of multiple robots, is applicable to most mobile robotics applications and can import height-fields

Simulator	GPS	Tracks	Wheels	Legs	Mecanum / Omni Wheels	Heightmap Import	OpenDrive / OpenStreetMap	Pathplanning	ROS Support	RGBD	LiDAR	Realistic Rendering
Gazebo	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
CoppeliaSim	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
Raisim	✗	✗	✓	✓	✗	✓	✗	✗	✗	✓	✓	✓, Unity
Webots	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
PyBullet	✗	✗	✓	✓	✗	✓	✗	✓	✗	✓	✓	✗
CARLA	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓, Unreal
Project Chrono	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓, POV-Ray

Figure 5: Feature comparison between popular robotics simulators used for Mobile Ground Robotics (Collins et al., 2021b)

for terrain geometry specification.
CARLA (Codevilla et al., 2018), a simulator designed for self-driving car research, is used in state-of-the-art research for learning driving policies and transferring policies trained in CARLA to the real world.

4.4.2 Manipulation

The manipulation sub-field in robotics covers a wide range of topics, including physical design of arms and grippers, as well as algorithms for motion planning and control. Simulators play an important role in manipulation research, as they can be used for multi-task or multi-step scenarios that require fine movement and contact modeling of rigid bodies. However, due to the need for stable physics that can handle contacts, only a few simulators are viable for this field. To be useful for manipulation research, simulators must have actuator models for position control, velocity control, and torque control.
MuJoCo (Todorov et al., 2012) is a popular simulator in manipulation research, thanks to its notable contact stability. It has been used to solve complex in-hand manipulation tasks, such as solving a Rubik's cube with a 24DOF robotic hand actuated with tendons. MuJoCo is also used in state-of-the-art research to train policies for robotic manipulators in simulation, whether for proof of concept or later transfer to real-world systems.

4.4.3 Medical Robotics

According to (Troccaz et al., 2019) medical robotics is a field of robotics research that involves applying automation and robotics to various medical applications, such as surgery, therapy, rehabilitation, and hospital automation. Robotic surgical systems are often teleoperated by surgeons, and advanced teleoperation controls provide haptic feedback, which enables the user to receive force/torque feedback directly from the robot. Real-time simulators are essential for teleoperation systems, especially for medical robotics research. Additionally, deformable object simulation is critical for realistic simulations because humans consist primarily of non-rigid tissue. SOFA and CHAI3D are two simulation frameworks commonly used in medical robotics research. SOFA supports real-time deformable object simulation, teleoperation and haptic control plugin, and tendon driven joints. CHAI3D (Shin et al., 2019) supports real-time simulation, multi-DOF teleoperation controllers and haptic feedback systems, and deformable object simulation. AMBF is a simulator developed for medical applications that provides fast-running simulation using CHAI3D for teleoperation support and haptic feedback and bullet for soft body simulation and prismatic joints. UnityFlexML is another simulator developed for use in machine learning applications that supports teleoperation control with haptics and deformable object simulation using Nvidia flex. However, other simulators used in medical robotics research offer only a limited subset of the necessary features.

4.4.4 Marine Robotics

There are two distinct categories of simulators for marine robotics: those specifically designed for underwater vehicles like AUVs and ROVs, and those intended for surface vehicles such as USVs, ships, and boats.
One open-source simulator for underwater vehicles is the Unmanned Underwater Vehicle (UUV) Sim-

Simulator	Pathplanning	Inverse Dynamics	Inverse Kinematics	Suction	Deformable Objects	Force/Torque Sensor	Realistic Rendering
SimGrasp	✓	✓	✓	✗	✗	✓	✗
Gazebo	✓	✓	✓	✗	✗	✓	✗
CoppeliaSim	✓	✗	✓	✓	✗	✓	✗
Pybullet	✓	✓	✓	✗	✓	✓	✗
MuJoCo	✗	✓	✗	✗	✓	✓	✗
NVidia Isaac	✓	✗	✓	✓	✓	✓	✓

Figure 6: Feature comparison for popular robotics simulators used for Manipulation (Collins et al., 2021b)

Simulator	Teleop	Haptic Feedback	Tendon Actuator	Deformable Objects	VR support
SOFA	✓	✓	✓	✓	Unity
Chai3D	✓	✓	✗	✓	✓
AMBF	✓	✓	✗	✓	✓
FLEX	✓	✓	✓	✓	Unity + Unreal
CoppeliaSim	✓	Chai3D plugin	✗	✗	✓

Figure 7: Feature comparison of popular robotics simulators used for Medical Robotics (Collins et al., 2021b)

ulator, which is an extension for Gazebo (Manhães et al., 2016). It supports multiple underwater vehicles and robotic manipulators, and includes high fidelity representations of hydrostatic and hydrodynamic forces. The simulator includes commonly used sensors, such as underwater cameras, pressure sensors, IMUs, magnetometers, and Doppler Velocity Logs (DVLs), as well as models for fins and thrusters for actuation. UUV allows researchers to create complex underwater environments, including models of seabeds, lakes, and shipwrecks. UUV has been used for applications such as mapping and path following.

Another open-source simulator for underwater vehicles is UWSim (de la Cruz et al., 2020), developed at the Interactive and Robotic Systems Lab at Jaume-I University. UWSim uses Bullet and OpenSceneGraph (OSG) for contact physics and supports a wide range of simulated sensors, including pressure sensors, force sensors, GPS, range cameras/sensors, IMUs, and DVLs. However, it lags behind in terms of accuracy of simulation for vehicle dynamics and hydrodynamics, and it only supports simulation of manipulator kinematics, not dynamics.

StoneFish Library (Cieślak, 2019) is another simulator for underwater vehicles, which is a wrapper for Bullet that supports standard sensors such as cameras, pressure sensors, DVLs, and multibeam sensors. The simulator is based on the actual geometry of the vehicle, allowing for better approximation of hydrodynamic forces. It includes hydrodynamic effects such as added mass, buoyancy, and drag, as well as underwater thrusters and manipulator systems for modeling more complex setups. StoneFish also supports advanced rendering of underwater scenes, including scattering and light absorption. However, the advanced rendering is computationally expensive and requires a recent GPU.

Unity ROS Simulator (URSim) is a simulator that uses ROS and the Unity 3D game engine to simulate both surface and underwater vehicles. URSim includes sensor models for cameras, IMUs, and pressure sensors, as well as noise models for sensor input. The simulator is capable of modeling environments used in competitions such as SAUVC and RoboSub.

Simulating surface vehicles is more complex due to environmental factors such as waves, wind, and water currents. One dedicated simulator for this application is the Unmanned Surface Vehicle simulator (USVSim), which is an extension of Gazebo. The freefloating plugin supports USV simulations by improving the hydrodynamics and buoyancy effects. The lift-drag plugin is used for calculating foil dynamics. USVSim offers accurate modeling of wave and water visual effects. By reusing and improving elements from the above tools, the authors have created a robust simulator that has been used for path planning.

4.4.5 Aerial Robotics

This section focuses on unmanned aerial vehicles (UAVs), which is a popular area of research within aerial robotics. Modern UAV simulators enable researchers to create intricate real-world environments that account for various fluid mechanics constraints, including turbulence, air density, wind shear, clouds, and precipitation. In addition, these simulators support a range of sensors such as LIDARs, GPS, and cameras. Digital elevation models are also used to simulate the terrain below the UAV. Some popular aerial robotics simulators include

Simulator	Hydrodynamics	Hydrostatics	Thruster	Fins	Pressure Sensor	GPS	DVL	Realistic Rendering	Contact Dynamics	Wind	Waves	Water Currents
UWSim	✓	✗	✓	✗	✓	✓	✓	✓, osgOcean	✓	✗	✓	✗
UUV	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗
Stonefish	✓	✓	✓	✗	✓	✓	✓	✓, custom	✓	✓	✓	✓
URSim	✓	✓	✓	✗	✓	✗	✗	✓, Unity	✗	✗	✓	✗
USVSim	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓

Figure 8: Feature comparison of popular simulators used for Marine Robotics (Collins et al., 2021b)

Simulator	Realistic Rendering	GPS	Barometer	Sonar	Radar	PX4	ArduPilot	HITL	RGBD + LiDAR	ROS Support	VR Support
AirSim	✓, Unreal + Unity	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
Flightmare	✓, Unity	✗	✗	✗	✗	✗	✗	✗	RGBD only	✓	✓
Gazebo	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Webots	✗	✓	✗	✓	✓	✗	✓	✗	✓	✓	✓

Figure 9: Feature comparison of popular simulators used for Aerial Robotics (Collins et al., 2021b)

Gazebo, AirSim, Flightmare, jMAVSim, and Webots.

Microsoft's AirSim is based on the Unreal engine and supports IMU, magnetometer, GPS, barometer, and camera sensors. It provides a built-in controller called simple_flight and supports open-source controllers such as PX4 (Shah et al., 2017). AirSim is resource-intensive and requires large computing power to run when compared to other simulators. It has been used in drone racing, wildlife conservation, and depth perception from visual images.

Flightmare (Song et al., 2021) combines a flexible physics engine with the Unity rendering engine to create a powerful simulator. It can simulate high-fidelity environments, including warehouses and forests. Sensor models are available for IMU and RGB cameras with ground-truth depth and semantic segmentation. Flightmare is suitable for deep/reinforcement learning applications.

jMAVSim is another popular simulator, mainly due to its close coupling with the open-source PX4 controller, which was the initial goal of testing PX4 firmware and devices. jMavSim supports basic sensing and rendering.

Webots is an open-source simulator that supports an extensive set of sensors, including cameras, LIDARs, GPS, etc. Users can add custom physics to simulate things such as wind and integrate data from OpenStreetMap to create more realistic environments. Integration with the Ardupilot flight controller is supported. Webots has been used in multi-agent simulations, mitigation of bird strikes, and landing applications (Michel, 2004).

4.4.6 Soft Robotics

Soft robotics simulation presents unique challenges compared to traditional rigid robotics simulations. Soft robots require simulating deformable objects and unconventional modes of actuation such as tendon/cable, pneumatic, and heat transfer. Additionally, simulators must account for contact dynamics between the soft robot and various materials, including both soft and solid objects, as well as fluids. Multiphysics packages such as COMSOL, ANSYS, and Abaqus are commonly used for soft robotics simulation. These packages use Finite Element Method (FEM) and can simulate a range of physics, such as heat transfer, electric conduction, magnetism, and fluid flow. However, they generally lack sensing capabilities, and are more suited for fundamental studies rather than environmental modeling. Abaqus is considered the industry standard, modeling non-linear behavior well and supporting a range of material properties. ANSYS is popular for fluid simulation, while COMSOL is more user-friendly and offers greater flexibility in defining material properties.

4.4.7 Learning for Robotics

Learning for robotics has been a popular topic of research in the past decade. However, due to the sample inefficiency of current Reinforcement Learning (RL) algorithms and the risk of damage to the robot during training, most works on deep RL are first learned in simulation before being deployed on hardware. Unlike the other sections covered in this work, learning for robotics is concerned with implementation on a robot rather than the type of robot or the environment that a robot is deployed in.

OpenAI Gym is a well-known toolkit for training and evaluating RL algorithms and provides environments in MuJoCo that serve as baselines to evaluate new RL algorithms and methods in the literature. It is used in conjunction with simulators like PyBullet, Webots, Nvidia Flex, Nvidia Isaac, CARLA, Project Chrono, Raisim, and Gazebo.

Simulator	FEA	Spring-Mass	Heat Conduction	Electrical Conduction	Magnetism	Pneumatics	Tendon
Evosoro	✗	✓	✗	✗	✗	✓	✗
SOFA	✓	✗	✓	✓	✗	✓	✓
COMSOL	✓	✗	✓	✓	✓	✓	✓
ABAQUS	✓	✗	✓	✓	✓	✓	✓
ANSYS	✓	✗	✓	✓	✓	✓	✓

Figure 10: Feature comparison of popular simulators used for Soft Robotics (Collins et al., 2021b)

Simulator	Random External Forces	RGBD + LiDAR	Force Sensor	Multiple Physics Engines	Realistic Rendering
Raisim	✓	✓	✗	✗	✓, Unity
Gazebo	✓	✓	✓	✓	✗
Nvidia Isaac	✓	✓	✗	✗	✓, Unity + Unreal
MuJoCo	✓	✓	✓	✗	✗
PyBullet	✓	✓	✓	✗	✗
CARLA	✗	✓	✗	✗	✓, Unreal
Webots	✓	✓	✓	✗	✗
CoppeliaSim	✓	✓	✓	✓	✗

Figure 11: Feature comparison of popular simulators used in learning for robotics (Collins et al., 2021b)

Deep learning can be used in various robotics fields. In manipulation and grasping tasks, sensor support and fidelity of rigid or soft body contact dynamics are important. In path planning or locomotion tasks, researchers require complex terrains to be modelled. Many simulators like Gazebo, Raisim, MuJoco, and PyBullet can import non-flat rigid terrains from heightmap images or mesh files but do not model soft or granular materials at a large enough scale to mimic soils, gravels, or fluid terrains. Project Chrono, however, is an alternative with built-in support for deformable terrain, granular terrain, and fluid simulations.

Sensor support is a critical consideration for researchers to select a suitable simulator for their application. Force-torque sensors and vision sensors are common requirements and are supported in simulators like Gazebo, PyBullet, and CoppeliaSim. To overcome the reality gap, researchers must address the most important consideration in the learning community when selecting a simulator, which is the need for simulation environments to vary between episodes. Many simulators have built-in support for resetting a simulation environment without shutting down the entire simulator and randomizing the textures of rendered objects in simulation.

Simulator features like **supporting parallel simulation, running in headless mode, and rapid dynamics solvers** can facilitate deep learning in a timely fashion. Nvidia Flex supports distributed GPU simulations, allowing it to reduce training times by up to eight times on some tasks. Flightmare can maintain 200,000 steps per second while simulating 150 quadrotors in parallel, allowing it to train locomotion policies for the quadrotors much faster than in real-time. Researchers should consider the computational load when selecting a simulator for learning, as it is essential to run simulations as quickly as possible or perform as many simulations in parallel as possible to reduce training time.

5 Sim2Real Transfer Techniques

5.1 Domain Adaptation

In the setting of reinforcement learning domain adaptation (DA) can be formalized as follows (Zhao et al., 2020): the source domain is the simulator environment defined as $D_s \equiv (S_s, A_s, P_s, R_S)$ while real-world is the target domain as $D_t \equiv (S_t, A_t, P_t, R_t)$. In this context while source and target domain share action space A and the transition probabilities P, have similar reward structure R, they can however have very different states S due to the sim2real gap.

Domain adaptation aims to bridge this gap by learning a discriminator or a predictor which maps the real-world features to a unified feature space making use of minimal target data. This can help to overcome both the visual and the dynamic challenges of deploying a robot in the real world.

There are two types of DA methods: *Unsupervised domain adaptation* (UDA) and *Supervised domain adaptation* (SDA), depending on the availability of labeled target data. UDA methods do not use any target labels and rely on domain-invariant features or adversarial learning (Chen et al., 2022). SDA methods use some target labels and leverage them to improve the adaptation performance (Prabhu et al., 2023). SDA methods are more suitable for safety-critical applications such as autonomous driving.

5.1.1 Dynamics Adaptation

Simulation-based policies often capture the general idea of how to achieve a goal, but fail in the real world due to fine-level details such as friction, backlash, precise measurements, and deformation that are not properly modeled in simulation. To address this challenge, we review some methods that leverage deep learning to adapt simulation-based policies to the real world.

The first method is **Deep Inverse Dynamics Model** (Christiano et al., 2016), it leverages the action generated by a Forward Dynamics

model, trained in simulation, and uses an Inverse Dynamics model to adapt that action to the real world. The inverse dynamics model is learned incrementally by collecting data from the target domain before deployment. This model can transfer policies from simulation to real world with minimal changes and outperforms baselines such as output error control and Gaussian dynamics adaptation.

Real world deployment of a robot requires not only to close the sim2real gap, but also to have a robot able to adapt in real time to unseen scenario. The **Rapid Motor Adaptation (RMA)**(Kumar et al., 2021) algorithm does that using two subsystems: an adaptation module and a policy. The adaptation module encodes the environment configuration into a latent vector using a short history of state action pairs. The base policy takes the current state, previous action and the latent vector. The adaptation module is trained with supervised learning during the simulation by regressing the output of an Environmental Factor Encoder which, since it is a simulation, has access to the actual environment configuration. RMA allows for online adaptation with minimal target data, speed is critical since collecting too much data could mean for the robot to crash when encountering an unseen terrain.

5.1.2 Visual Adaptation

An example of Supervised Domain Adaptation for 2D object detection is given by **Domain Translation via Conditional Alignment and Reweighting (CARE)** (Prabhu et al., 2023), a method which combines both synthetic source data and human-labeled target data. In this scenario a substantial amount of ground truth labels for the target dataset are available during training. CARE addresses both the appearance gap (the visual disparities between the two domains) and the content gap (the shift in label distribution and bounding box size) by rassigning different weights to samples to minimize the difference in distributions and conditionally align intermediate instance representations.

Traversability prediction is a crucial skill for autonomous driving systems, as it allows to determine whether a certain path can be safely taken or not. This task requires to perform semantic segmentation of the environment in order to detect objects and potential obstacles. One of the proposed UDA models is **Coarse-to-Fine Alignments (CALI)**(Chen et al., 2022) which combines both Domain Alignment (DA) and Class Alignment (CA). DA treats the deep features as a whole, while CA considers the class distributions. CALI implements a game structure where a feature extractor G plays with both a discriminator D, to perform DA, and two classifiers C_i to perform class alignment. The game between G and D involves the optimization of a Generative Adversarial Networks (GAN) to perform Domain Alignment. The game between G and C_i performs class alignment indirectly by minimizing the disagreement between the two classifiers, which have been trained on the source domain. The disagreement is a proxy for the domain shift and is used as a measure of it.

One way to bridge the gap between simulated and real-world environments for robotic grasping is to employ **Randomized-to-Canonical Adaptation Networks (RCAN)**(James et al., 2018). This network learns to adapt a randomized simulation scene to its canonical simulated counterpart using an image-conditioned generative adversarial network (cGAN). The same transformation is then applied to real-word scenes, resulting in canonical images that are suitable for the control model. This approach makes uses both randomization, which is used to train RCAN and not the control model itself, and adaptation, which ensures that the control model operates on the canonical domain regardless of the input domain.

An interesting approach which deals with both the visual and the dynamics gap is **Bi-directional Domain Adaptation (BDA)**(Truong et al., 2021). DA consists of two modules: a real2sim Observation Adaptor that transforms real images to look like simulated ones, and a sim2real Dynamics Adaptor that adjusts the simulator parameters to match the real dynamics. Unlike Deep Inverse Dynamics Model that only use real data to adapt the policy, BDA uses real data to adapt the simulator. BDA has been shown to improve the performance of point-goal navigation tasks in different environments.

5.2 Domain Randomization

Domain randomization (Tobin et al., 2017) is a complementary class of techniques for domain adaptation. The goal is to close the reality gap by generating synthetic data with sufficient variation that the network views real-world data as just another variation. Domain randomization is a technique used in simulation to cover the real distribution of real-world data by highly randomizing

the simulation, rather than carefully modeling all the parameters of the real world. Instead of training a model on a single simulated environment, the simulator is randomized to expose the model to a wide range of environments at training time. In light of this, domain randomization leverages the fact that with significant variability in simulation, models can be trained to generalize effectively to real-world scenarios with no further requisites for training. This is particularly useful for robotic vision tasks such as object localization, object detection, pose estimation, and semantic segmentation, where training data from a simulator may have different textures, lighting, and camera positions from the realistic environments. Domain randomization can be divided into two kinds: visual randomization and dynamics randomization, with the former providing simulated variability of visual parameters at training time to help the model generalize to real-world data, and the latter randomizing physical parameters in the simulator to help acquire a robust policy where controlling policy is needed. Successful sim-to-real transfer experiments have shown the powerful effect of domain randomization on acquiring diverse experience for robotic systems.

5.2.1 Visual Randomization

Object Localization

In the field of robotics, object localization from pixels has always been a challenging problem, and its state-of-the-art methods employ complex, hand-engineered image processing pipelines. However, recent advances in deep learning hold promise for improving the accuracy of object detection pipelines. In this regard, the paper "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World" (Tobin et al., 2017) describes a method for training an object detector that can map a monocular camera frame to the Cartesian coordinates of objects of interest, while ignoring distractor objects. The approach utilizes domain randomization to create enough simulated variability during training that the model can generalize well to real-world data. The domain randomization involves randomizing various aspects of the simulated domain, including the number and shape of distractor objects, the textures of all objects on the table, the position and orientation of the camera, and the type and amount of random noise added to images. The training is performed using the MuJoCo Physics Engine's (Todorov et al., 2012) built-in renderer, which does not require physically plausible choices of textures and lighting. The textures of objects are chosen uniformly at random and between 0 and 10 distractor objects are added to the table in each scene.

Object detection

Another paper (Tremblay et al., 2018) presents an extension of domain randomization (DR) to the challenging task of real-world object detection using synthetic images. This paper makes important contributions such as extending DR to complex object detection tasks, proposing a new DR component, namely, flying distractors, to enhance object detection and estimation accuracy, and evaluating the significance of DR parameters for such tasks. The approach starts by using 3D models of objects of interest, such as cars, and placing a random number of these objects in a 3D scene at random positions and orientations. To better enable the network to learn to ignore extraneous objects in the scene, a random number of geometric shapes are added to the scene as flying distractors. Random textures are then applied to both the objects of interest and the flying distractors. A random number of lights of different types are inserted at random locations, and the scene is rendered from a random camera viewpoint, after which the result is composed over a random background image. The resulting images, with automatically generated ground truth labels, are then used for training the neural network. By varying different aspects of the scene, such as the number and types of objects, the textures, and the camera positioning, the network is trained to focus on important structures of the problem at hand.

Semantic segmentation

(Yue et al., 2019) place a significant emphasis on domain randomization techniques in their research, utilizing them to generate synthetic images that closely mimic the styles of real images. They go further to ensure consistency in the semantic segmentation network's prediction across these randomized domains. The approach they take to domain randomization and consistency-enforced training, as well as their use of image-to-image translation to randomize the source domain imagery, set this paper apart as truly innovative. Through domain randomization with stylization, they mapped the synthetic imagery to multiple auxiliary real domains during the training stage, which allowed the CNN model to easily recognize the tar-

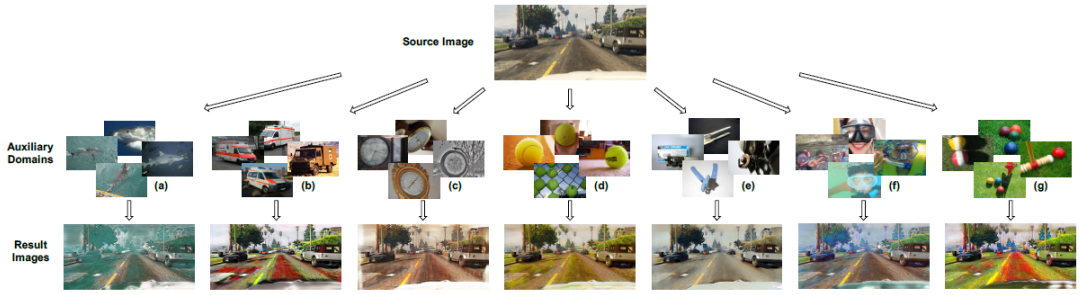

Figure 12: The domain randomization process in semantic segmentation. Top: an original synthetic image from the source domain; Mid: auxiliary image sets composed of ImageNet classes: (a) great white shark, (b) ambulance, (c) barometer, (d) tennis ball, (e) can opener, (f) snorkel, (g) tennis ball; Bottom: stylized images with same image content as the synthetic image and meanwhile corresponding styles of the ImageNet classes. (Yue et al., 2019)

get domain as just another real domain during the test stage. The technique used in this paper during the domain randomization phase, involves adding K ImageNet categories to the training set to stylize synthetic images. The synthetic images are then mapped to each of the K domains using image-to-image translation. This expands the training set to K+1 times the original size. A CNN segmentation model is trained with this augmented set, and it outperforms single-source domain methods.

5.2.2 Dynamics Randomization

Dynamics Randomization enhances learned policies by introducing controlled randomness into a robot's dynamics during training. By varying the physical parameters that govern interactions with the environment, such as mass and friction, the learning agent adapts to a diverse range of dynamic conditions. This mitigates overfitting and improves the transferability of behaviors to real-world scenarios. With increased resilience to perturbations, Dynamics Randomization has proven effective in domains like robotic manipulation, locomotion, and aerial robotics. Its potential to facilitate efficient sim2real transfer paves the way for advancements in robotics and autonomous systems.

The research article titled *Learning dexterous in-hand manipulation* (Andrychowicz et al., 2020) focuses on the challenges robots face when performing intricate tasks compared to humans. These challenges arise from the narrow design of robots built for specific purposes in limited environments. To address this issue, the study introduces domain randomization as a key concept. The authors employ several techniques, such as comprehensive randomizations, additional effects in simulated environments, memory augmented control policies, calibration, and distributed reinforcement learning, to achieve domain randomization. This approach enables the training of control policies for in-hand manipulation, their transfer to physical robots, and the facilitation of adaptive behavior and implicit system identification. The main objective of the paper is to reorient objects within the hand to a desired configuration. To avoid overfitting to specific simulated environments, most aspects of the simulated environment are randomized. Sensing modalities are carefully selected, and parameters with high uncertainty, like actuation parameters, undergo more randomization compared to parameters with low uncertainty. Physical parameters, such as object dimension, object and robot link masses, surface friction coefficient, robot joint damping coefficient and the actuator force gains are sampled at the beginning of each episode and remain fixed throughout the episode to enhance learning accuracy. Although the simulation used in the study is not an exact representation of the physical setup, modifications are made to bridge the *reality gap* between simulation and the real environment. These modifications involve creating a distribution of multiple simulations, selecting appropriate sensing modalities, and randomizing various aspects of the simulated environment. By doing so, the study aims to train policies that generalize effectively and can be successfully transferred to physical robots. Randomized parameters are centered around reasonable physical values obtained through calibration. Gaussian noise is added to policy observations to mimic real-world noise, and structured noise is introduced by displacing motion capture markers. Action noise, delay,

and a backlash model are included to account for imperfections in the actuation system and the discrepancy between the tendon-actuated physical hand and the MuJoCo (Todorov et al., 2012) model. Random forces are occasionally applied to the object to represent unmodeled dynamics. Visual appearance randomizations encompass various aspects of the rendered scene, including camera positions, lighting conditions, object poses, materials, and textures. The utilization of domain randomization, along with these various randomizations, enhances the generalization capabilities of control policies and vision models, enabling more successful transfer to physical robots.

The use of domain randomization, combined with the randomization of various parameters and effects, improves the ability of control policies and vision models to generalize and transfer effectively to physical robots.

Dynamics Randomization Revisited

The authors (Xie et al., 2020) provide insights into the role of dynamics randomization in learning robust locomotion policies for the Laikago quadruped robot (Figure 13). In contrast to previous work, the authors surprisingly find that direct sim-to-real transfer is achievable without the need for dynamics randomization or on-robot adaptation schemes. They demonstrate that dynamics randomization is not necessary for successful sim-to-real transfer in their specific settings, encompassing various gaits and speeds, while maintaining robustness against common perturbations.

Firstly, they highlight that randomization is often employed without a comprehensive understanding of other potential sources of failure or the need for randomization itself. Moreover, they emphasize that the impact and unintended consequences of randomization choices are often overlooked. In their study, the authors demonstrate that their learned controllers do not require dynamics randomization for successful performance. To evaluate the robustness of the policies, the authors subject them to various perturbations in simulation and on the physical robot. These perturbations include mass variations, changes in proportional gain, latency simulation, lateral pushes, and walking on slopes. Notably, the policies exhibit robustness beyond what is typically explored in related sim-to-real research, accommodating a wider range of perturbations and modeling errors. Importantly, when directly applied to the physical robot, the policies demonstrate sim-to-real success without the need for adaptation. While the performance of the policies may vary for more dynamic motions, such as gallops or jumps, the authors emphasize that their findings challenge the notion that dynamics randomization is universally required for sim-to-real transfer. They highlight the policies' robustness against a broad range of perturbations and the ease with which the reality gap is overcome.

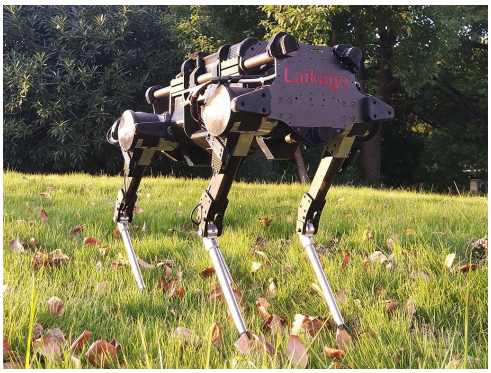

Figure 13: Laikago quadruped robot [1]

The authors also argue that domain randomization alone is not sufficient, based on their observations and analysis. They find that trotting policies trained with different proportional gains (kp) exhibit distinct motion characteristics and sensitivities. Policies trained with high kp values (e.g., kp = 160) display stiff and precise motion but become sensitive to contact timing, leading to occasional backward foot kicks when the policy expects the foot to be in the stance phase but it has not yet made contact with the ground. On the other hand, policies trained with low kp values (e.g., kp = 40) generate motions with larger PD-tracking errors, behaving more like torque controllers. The authors hypothesize that low proportional gains result in policies that behave like torque controllers, which are typically more robust to unanticipated impacts, while high proportional gains lead to behaviors resembling position controllers.

Based on these observations, the authors argue that domain randomization alone is insufficient to address the challenges posed by specific motion characteristics and control sensitivities. They find that dynamics randomization can lead to conservative policies that prioritize unnecessary robustness in randomized parameters. Policies trained with dy-

[1] http://www.unitree.cc/?utm_source=robots.ieee.org

namics randomization perform worse in terms of overall performance compared to policies trained without it, except in dealing with latency. However, the unnecessary robustness against latency compromises performance and robustness in other dimensions. The authors emphasize the importance of randomizing parameters that matter, as demonstrated by the latency test where policies trained without randomization fail when latency exceeds 17 ms, while policies trained with randomized latency can handle latency up to 32 ms. This highlights that dynamics randomization can be useful in scenarios with significant modeling errors, such as latency, by selectively randomizing the responsible parameters.

5.2.3 Active Domain Randomization

The basic idea behind DR is to randomly vary environmental parameters in simulation in order to train agents to be robust to domain shifts. However, recent research has shown that **the sample complexity of DR can grow exponentially with the number of randomized parameters**, even when only dealing with transfers between simulations. In (Mehta et al., 2019), the authors propose an alternative technique called Active Domain Randomization (ADR), which takes a targeted approach to sampling randomized parameters and aims to improve the efficiency and efficacy of DR. By targeting problematic parameter settings, ADR dedicates more time to training and optimizes performance. It leverages Reinforcement Learning (RL) to achieve its goals, using Stein Variational Policy Gradient (SVPG) (Liu et al., 2017) to parameterize its sampling policy. To hone in on the most challenging areas of randomized space, ADR creates a discriminative reward by comparing policy rollouts from randomized and reference environments. This allows ADR to focus on the regions that are causing the most difficulty and adapt as needed. ADR generates randomized environments and simulation instances from a simulator and rolls out agent policies in these scenarios. The discriminator learns to distinguish between reference and randomized environments, which is used to train SVPG particles. These particles generate a broad range of environments to pinpoint the parameters that are most problematic for the agent.

5.2.4 Auto-Tuned Sim-to-Real Transfer

(Du et al., 2021) presents a novel approach to reducing the amount of task-specific engineering required for domain randomization in robotics, by proposing an automatic system identification procedure that aims to tune a simulation to match the real world. The authors introduce a task-agnostic reinforcement learning method that uses a small amount of real-world video data and learns to shift the simulation system parameters to become more similar to the real world. This is achieved by predicting whether the current parameters are higher, lower, or close to the real world values in a given sequence of observations, instead of directly predicting the parameters from observations. The paper introduces a Search Param Model (SPM) that updates the system parameters using raw pixel observations of the real world. The proposed method aims to automatically find the true system parameters of a robotic task by using a function that maps a sequence of observations and actions to their corresponding system parameters. This is achieved by reformulating the auto-tuning problem as a search procedure, where a binary classifier is trained to predict the probability that the true set of system parameters is higher, lower, or equal to the given predicted parameters. The Search Param Model (SPM) is trained in two stages, first pretraining with simulated samples and then jointly with policy training. Using the SPM, the system parameters are iteratively updated to become closer to the true system parameters by occasionally collecting real-world trajectories and using the SPM to predict whether the current parameters are too high or too low. The proposed method outperforms domain randomization on a range of robotic control tasks in both sim-to-sim and sim-to-real transfer and the main limitation is the difficulty in learning the true simulation parameters when the initial randomization is too far from the true values.

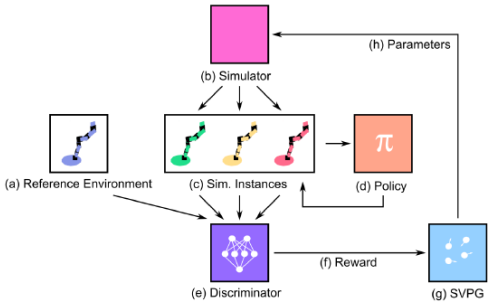

Figure 14: Active Domain Randomization (Mehta et al., 2019)

6 Meta Learning

Meta-Learning is learning to learn (Tian et al., 2022), to train a model which is able to learn a new task from just a few examples. The learner is trained on a set of different tasks so that later it will be able to quickly adapt to a new one.
As noted by Lake et al. (Lake et al., 2015) people can learn from just a handful of examples, while machine learning algorithms take thousands examples or more to reach human-level performances. This is however an unfair challenge, we do not learn to perform a new task *tabula rasa* like a model, we reuse and adapt a big set of skills and knowledge that we accumulate since our birth and even before in our DNA. The challenge of meta-learning is to train a model that is able to learn a new task using its previously learnt learning ability.
A popular meta-learning approach is Model Agnostic Meta Learning (MAML) (Finn et al., 2017), which learns a model initialization that can be fine-tuned for any new task with a few gradient steps. MAML is applicable to any model that can be trained with gradient descent, and has been shown to work well for various tasks such as classification, regression, reinforcement learning, and imitation learning. MAML has inspired many extensions and variations, such as Reptile (Nichol et al., 2018), CAVIA (Zintgraf et al., 2019), FOMAML (Biswas, 2018).

6.1 Sim2Real Applications

The idea of meta-learning lends itself to sim2real applications, learning how to perform a new task quickly and *learning* how to adapt the task to a new environment are similar problems. In (Arndt et al., 2019) a variation of MAML is used for an adaptive policy in combination with a generative model to provide an action space which is easier to explore. Another example of the use of meta learning is (Yu et al., 2018) where a robot is trained to perform one-shot imitation of human behaviour from a video. This method again builds upon MAML to handle the domain shift between the human demo provided in the video and the robot execution.

7 Conclusion

Sim-to-real transfer has been increasingly adopted in the robotics community for controlling complex robots and multi-robot systems or providing end-to-end policies from perception to control. However, the sim2real gap remains a significant challenge in achieving successful transfer, where the simulated environment may differ significantly from the real-world setting. To address this challenge, researchers have proposed several approaches such as domain adaptation and domain randomization.
Domain adaptation involves learning a discriminator or predictor that maps real-world features to a unified feature space, making use of minimal target data, thereby bridging the gap between the source and target domains. Both unsupervised and supervised domain adaptation methods have been proposed, with supervised methods being more suitable for safety-critical applications.
On the other hand, domain randomization aims to generate synthetic data with sufficient variation, which enables the network to view real-world data as just another variation. This technique provides simulated variability of visual and physical parameters during training time to help the model generalize to real-world data. Successful sim-to-real transfer experiments have shown the powerful effect of domain randomization on acquiring diverse experience for robotic systems.
Simulators such as Isaac Gym and Mujoco have been used in the literature, with Gazebo being suitable for complex scenarios while PyBullet and MuJoCo provide faster training. In addition, meta-learning and evolutionary robotics have been identified as two promising research directions for sim-to-real transfer, where multiple problems can be addressed.
Differentiable simulators (Collins et al., 2021a) are a promising direction for improving sim2real methods, because they allow the use of gradient-based methods to optimize actions, states and model variables. Differentiable physics engines are more interpretable and flexible than differentiable data-trained models. Therefore have many potential applications in robotics, and can deal with soft body physics (Hu et al., 2019).
GPU accelerated training environment like Isaac Gym have a lot to offer, enabling massive parallel training (Rudin et al., 2022) and better physics simulation.

References

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. 2020. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20.

Karol Arndt, Murtaza Hazara, Ali Ghadirzadeh, and Ville Kyrki. 2019. Meta reinforcement learning for sim-to-real domain adaptation.

C. Dario Bellicoso, Fabian Jenelten, Christian Gehring, and Marco Hutter. 2018. Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots. *IEEE Robotics and Automation Letters*, 3(3):2261–2268.

Abhijat Biswas. 2018. First-order meta-learned initialization for faster adaptation in deep reinforcement learning.

Zheng Chen, Durgakant Pushp, and Lantao Liu. 2022. CALI: Coarse-to-Fine ALIgnments Based Unsupervised Domain Adaptation of Traversability Prediction for Deployable Autonomous Navigation. In *Proceedings of Robotics: Science and Systems*, New York City, NY, USA.

Paul F. Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. 2016. Transfer from simulation to real world through learning deep inverse dynamics model. *CoRR*, abs/1610.03518.

Patryk Cieślak. 2019. Stonefish: An advanced open-source simulation tool designed for marine robotics, with a ros interface. *OCEANS 2019 - Marseille*, pages 1–6.

Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. 2018. End-to-end driving via conditional imitation learning.

Jack Collins, Ross Brown, Jürgen Leitner, and David Howard. 2021a. Follow the gradient: crossing the reality gap using differentiable physics (realitygrad). *arXiv preprint arXiv:2109.04674*.

Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. 2021b. A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431.

Marcos de la Cruz, Gustavo A. Casañ, Pedro J. Sanz, and Raúl Marín. 2020. A new virtual reality interface for underwater intervention missions. *IFAC-PapersOnLine*, 53(2):14600–14607. 21st IFAC World Congress.

Stephane Doncieux, Nicolas Bredeche, Jean-Baptiste Mouret, and Agoston E. (Gusz) Eiben. 2015. Evolutionary robotics: What, why, and where to. *Frontiers in Robotics and AI*, 2.

Yuqing Du, Olivia Watkins, Trevor Darrell, Pieter Abbeel, and Deepak Pathak. 2021. Auto-tuned sim-to-real transfer. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1290–1296.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks.

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. 2019. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, pages 6265–6271. IEEE.

Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. 2018. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *CoRR*, abs/1812.07252.

N. Koenig and A. Howard. 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. 2021. RMA: Rapid Motor Adaptation for Legged Robots. In *Proceedings of Robotics: Science and Systems*, Virtual.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng. 2017. Stein variational policy gradient. *CoRR*, abs/1704.02399.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. 2021. Isaac gym: High performance gpu-based physics simulation for robot learning. *CoRR*, abs/2108.10470.

M.M.M. Manhães, S.A. Scherer, M. Voss, L.R. Douat, and T. Rauschenbach. 2016. Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. 2019. Active domain randomization. *CoRR*, abs/1904.04762.

Olivier Michel. 2004. Cyberbotics ltd. webots™: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5.

Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms.

Stefano Nolfi and Dario Floreano. 2000. *Evolutionary Robotics: The Biology,Intelligence,and Technology*. MIT Press, Cambridge, MA, USA.

Viraj Prabhu, David Acuna, Andrew Liao, Rafid Mahmood, Marc T. Law, Judy Hoffman, Sanja Fidler, and James Lucas. 2023. Bridging the sim2real gap with care: Supervised detection adaptation with conditional alignment and reweighting.

Asita Kumar Rath, Dayal R. Parhi, Harish Chandra Das, Manoj Kumar Muni, and Priyadarshi Biplab Kumar. 2018. Analysis and use of fuzzy intelligent technique for navigation of humanoid robot in obstacle prone zone. *Defence Technology*, 14(6):677–682.

Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. 2019. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2):1729881419839596.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. 2022. Learning to walk in minutes using massively parallel deep reinforcement learning.

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2017. Airsim: High-fidelity visual and physical simulation for autonomous vehicles.

Changyeob Shin, Peter Walker Ferguson, Sahba Aghajani Pedram, Ji Ma, Erik P. Dutson, and Jacob Rosen. 2019. Autonomous tissue manipulation via surgical robot using learning based model predictive control. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE.

Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. 2021. Flightmare: A flexible quadrotor simulator.

Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*, second edition. The MIT Press.

Kenta Takaya, Toshinori Asai, Valeri Kroumov, and Florentin Smarandache. 2016. Simulation environment for mobile robots testing using ros and gazebo. pages 96–101.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. 2018. Sim-to-real: Learning agile locomotion for quadruped robots.

Yingjie Tian, Xiaoxi Zhao, and Wei Huang. 2022. Meta-learning approaches for learning-to-learn in deep learning: A survey. *Neurocomputing*, 494:203–223.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30.

Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. 2018. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. *CoRR*, abs/1804.06516.

Jocelyne Troccaz, Giulio Dagnino, and Guang-Zhong Yang. 2019. Frontiers of medical robotics: From concept to systems to clinical translation. *Annual Review of Biomedical Engineering*, 21(1):193–218. PMID: 30822100.

Joanne Truong, Sonia Chernova, and Dhruv Batra. 2021. Bi-directional domain adaptation for sim2real transfer of embodied navigation agents. *IEEE Robotics and Automation Letters*, 6(2):2634–2641.

Zhaoming Xie, Xingye Da, Michiel van de Panne, Buck Babich, and Animesh Garg. 2020. Dynamics randomization revisited: A case study for quadrupedal locomotion. *CoRR*, abs/2011.02404.

Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. 2018. One-shot imitation from observing humans via domain-adaptive meta-learning. *CoRR*, abs/1802.01557.

Xiangyu Yue, Yang Zhang, Sicheng Zhao, Alberto L. Sangiovanni-Vincentelli, Kurt Keutzer, and Boqing Gong. 2019. Domain randomization and pyramid consistency: Simulation-to-real generalization without accessing target domain data. *CoRR*, abs/1909.00889.

Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. 2020. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744.

Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. 2019. Fast context adaptation via meta-learning.