COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

Institute for Research in Technology (IIT)
ICAI School of Engineering – Comillas Pontifical University

EFFICIENTLY TRANSFERRING DEEP REINFORCEMENT LEARNING EXPERIENCE TO INDUSTRIAL ASSETS

Author: Lucía Güitta López
Supervisors:
Dr. Álvaro Jesús López López
Dr. Jaime Boal Martín-Larrauri

Madrid
November 2024

Copyright © 2024 Lucía Güitta López

This dissertation was typeset with LaTeX and compiled in Overleaf using the TeX Live 2024 distribution. The font families used are Bitstream Charter, Open Sans, Inconsolata, and American Mathematical Society (AMS) fonts. Unless otherwise noted, all figures were created by the author using Microsoft Visio®, Microsoft Power Point®, and Python using the Plotly library (v5.10.0).

To me, for putting up with myself all these years
A mí, por soportarme durante tantos años

Acknowledgments

One of the things I told myself when I started this document was that the last thing I would write would be the acknowledgments. This way, I could look back, remember how it all began, relive the good and not-so-good moments, and thus close one of the most enriching chapters of my life.

"Standing on the shoulders of giants," one of Isaac Newton's most famous phrases, points out that his discoveries would not have been possible without the work of other scientists. In my case, my giants have been Álvaro and Jaime, my supervisors. Thanking them with just a couple of lines falls short of what they truly deserve, but since I don't want to reach 200 pages for the thesis, I'll try to summarize. Álvaro, Jaime, thank you so much for everything. This thesis would not have been possible without you. In the beginning, you knew how to lay out a pipeline that, although ambitious, has perfectly set the roadmap, and hey, we've met all the objectives. Thank you for always listening to the contributions I made as I gained more experience and for giving me full autonomy to investigate, something that may seem taken for granted but is not always the case. Thank you also for all the "healthy" disputes that have come up, which have improved both my ability to argue my case (even if that meant sleepless nights trying to find the best arguments) and my low tolerance for losing and not having things go my way. Ultimately, thank you for your help and guidance throughout these years. Learning from you has been the best experience I take away from this thesis, not just professionally. You are the best bosses I could have had.

Thank you to my colleagues at IIT, both veterans and newcomers, to the professors at the University who always had kind words for me and were willing to help, and to the student collaborators who, in one way or another, contributed to this thesis. A special mention goes to David and Pablo, who started as work colleagues and ended up being great friends. Thank you so much for always being so attentive, especially during this tough final stretch. You have been amazing older brothers. You are excellent researchers, but without a doubt, even better people.

Thank you to my parents Concha and Lionel and to my brother Lío for their unconditional support, no matter the adventure, which in this case has been longer than a weekend away for a triathlon competition. You have always been the best example of hard work, sacrifice, and discipline, and if I've gotten this far, it's because of you. Thank you to my family, especially my grandparents Lucía, Josiane, José, and León. Abueli, Memé, Figura, I hope you are as proud of me as I am of being your granddaughter. Thank you for always making your best effort to understand what I was working on, even if after each conversation it all boiled down to "well, yes, a robot that learns to move with that artificial intelligence stuff," which isn't far from what I actually did.

Thank you to my friends, the ones who have always been there, who, even though they can't tell a robot from an excavator (note the similarity in the orange color), have always been happy for what I've achieved and celebrated them as if they were their own. Irene, Marina, Ana, Popi, the next round of Tequifresa is on me. Thank you, Patri, for all those hours of listening to my complaints and frustrations, and thank you for sticking around even when I was in Rome or San Diego. I'll be careful with the lettuce, and I'll incorporate posterior chain stretches into my routine.

This last year, I was fortunate enough to do two research stays, one in Rome with Professor Nardi's research group and another in San Diego with Professor Atanasov's group. Grazie sia al professor Nardi che ai miei colleghi della Sapienza, i ragazzi della B120 e B121, per avermi accolto così velocemente. Mi avete fatto trascorrere dei mesi indimenticabili. Vi aspetto a Madrid, o di nuovo a Roma, per prendere altre tumba pecore. Questa esperienza non sarebbe stata la stessa senza Elena. Grazie raga per aver fatto sentire Roma come a casa mia. Magari mi ripropongo di rinnovare l'abbonamento, come se mi fosse stato regalato. Thank you also to Professor Atanasov and my colleagues at UCSD. Life and research in San Diego have been one of the most valuable experiences of this journey. I really appreciate how you welcomed me, your willingness to collaborate, and how we got the most out of our work. I am looking forward to seeing how our investigation continues. I hope we can meet in Madrid or back in San Diego soon.

Thank you to everyone who is reading this. I'm sure you've contributed something positive to my life at some point. And if you're considering reading the thesis, go ahead and enjoy it, because in the end, as my grandparents say, it's simply the story of how a robot learns to do things.

Agradecimientos

Una de las cosas que me dije al empezar este documento fue que lo último que escribiría serían los agradecimientos porque así podría mirar hacia atrás, recordar cómo empezó todo, revivir los buenos y no tan buenos momentos, y cerrar con ello una de las etapas más enriquecedoras que he vivido.

"A hombros de gigantes", una de las frases más conocidas de Isaac Newton con la que señala que sus descubrimientos no hubiesen sido posibles sin el trabajo de otros científcos. En mi caso mis gigantes han sido Álvaro y Jaime, mis directores. Agradecerles sólo con un par de líneas se queda corto para todo lo que se merecen, pero como no quiero llegar a las 200 páginas de tesis, intentaré resumirlo. Álvaro, Jaime, mil gracias por todo. Esta tesis no hubiese sido posible sin vosotros. En el inicio supisteis trazar un *pipeline* que aunque era ambicioso, ha ido marcando la hoja de ruta perfectamente, y oye, que hemos cumplido todos los objetivos. Gracias por haber escuchado siempre las aportaciones que iba proponiendo según cogía experiencia, y por haberme dado total autonomía para investigar, algo que puede darse por hecho pero que no siempre es así. Gracias también por todas esas disputas sanas que han ido surgiendo y han hecho que mejore, por un lado, mi capacidad de argumentación para llevarme el gato al agua, aunque eso supusiese noches de desvelo intentando encontrar los mejores argumentos, y por otro, mi poca tolerancia a perder y que no se haga lo que yo digo. En definitiva gracias por vuestra ayuda y seguimiento durante estos años, aprender de vosotros es la mejor experiencia que me llevo de esta tesis, y no sólo para el ámbito profesional. Sois los mejores jefes que he podido tener.

Gracias a mis compañeros del IIT, a veteranos y noveles, a los profesores de ICAI - Universidad Pontificia Comillas que siempre han tenido buenas palabras hacia mí y han estado dispuestos a ayudarme, y a los alumnos colaboradores que de algún modo u otro han aportado su grano de arena a esta tesis. Mención especial para David y Pablo, que empezaron siendo compañeros de trabajo y han acabado siendo grandes amigos. Mil gracias por haber estado siempre tan pendientes de mí, especialmente en esta recta final que ha sido tan dura. Habéis sido unos hermanos mayores geniales. Sois muy buenos investigadores, pero sin duda, mejores personas.

Gracias a mis padres Concha y Lionel y a mi hermano Lío por su apoyo incondicional siempre, toque la aventura que toque, que en este caso no ha sido tan corta como irnos un finde a una competición de triatlón. Habéis sido siempre el mejor ejemplo de trabajo, sacrificio y disciplina, y si he llegado hasta aquí es por vosotros. Gracias a mi familia, en especial a mis abuelos Lucía, Josiane, José y León. Abueli, Memé, Figura, espero que estéis tan orgullosos de mi como yo lo estoy de ser vuestra nieta. Gracias por haber hecho siempre vuestro mejor esfuerzo por entender en qué estaba trabajando aunque después de cada conversación, todo se resumiese en "bueno sí, un robot que aprende a moverse con cosas de la inteligencia artificial esa", lo cual no está para nada lejos de lo que he hecho.

Gracias a mis amigas, las de siempre, que aunque no distingan el robot de una excavadora (véase su parecido en el color naranja), siempre se han alegrado de las cosas que he ido consiguiendo y las han celebrado como si fuesen suyas. Irene, Marina, Ana, Popi, a los próximos Tequifresa invito yo. Gracias Patri por todas esas horas escuchando mis quejas y lamentos, gracias por haber seguido estando aunque yo estuviese en Roma o San Diego. Tendré cuidado con la lechuga e incorporaré estiramientos de cadena posterior en mi rutina.

Este último año tuve la suerte de poder realizar dos estancias de investigación, una en Roma con el grupo de investigación del profesor Nardi, y otra en San Diego con el grupo del profesor Atanasov. Grazie sia al professor Nardi che ai miei colleghi della Sapienza, i ragazzi della B120 e B121, per avermi accolto così velocemente. Mi avete fatto trascorrere dei mesi indimenticabili. Vi aspetto a Madrid, o di nuovo a Roma, per prendere altre tumba pecore. Questa esperienza non sarebbe stata la stessa senza Elena. Grazie raga per aver fatto sentire Roma come a casa mia. Magari mi ripropongo di rinnovare l'abbonamento, come se mi fosse stato regalato. Thank you also to Professor Atanasov and my colleagues at UCSD. Life and research in San Diego have been one of the most valuable experiences of this journey. I really appreciate how you welcomed me, your willingness to collaborate, and how we got the most out of our work. I am looking forward to seeing how our research continues. I hope we can meet in Madrid or back in San Diego soon.

En definitiva, gracias a todo aquel que esté leyendo esto. Seguro que me has aportado algo positivo en algún momento. Y si te estás planteando seguir leyendo la tesis, ánimo y disfrútalo, que al final, como dicen mis abuelos, es simplemente la historia de cómo un robot aprende a hacer cosas.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor-Critic
AI	Artificial Intelligence
AR	Augmented Reality
AR/VR	Augmented and Virtual Reality
BM	Baseline Model
CNN	Convolutional Neural Network
CycleGAN	Cycle-Consistent Generative Adversarial Network
DA	Domain Adaptation
DDQN	Double Deep Q-Network
DNN	Deep Neural Network
DoF	Degrees of Freedom
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DR	Domain Randomization
DRL	Deep Reinforcement Learning
DRM	Domain Randomization Model
Dueling DQN	Dueling Deep Q-Network
FC	Fully Connected
GAE	General Advantage Estimator
GAN	Generative Adversarial Network
GPI	Generalized Policy Iteration
IIoT	Industrial Internet of Things
IoT	Internet of Things
KGE	Knowledge Graph Embedding
LLM	Large Language Model
LSTM	Long-Short Term Memory
MAE	Mean Absolute Error

MDP	Markov Decision Process
MPI	Maximum Position Increment
MSE	Mean Squared Error
MuJoCo	Multi-Joint dynamics with Contact
NN	Neural Network
PNN	Progressive Neural Network
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
ROS	Robotic Operating System
SICGAN	StyleID-CycleGAN
std. dev.	standard deviation
TD3	Twin Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization

Abstract

The Fourth Industrial Revolution has driven the integration of advanced technologies into the industry, highlighting the use of Artificial Intelligence to improve process efficiency. In this context, Deep Reinforcement Learning (DRL) is a promising solution for complex sequential decision-making problems. One of the main challenges in this type of learning is sample efficiency, i.e., the number of agent-environment interactions needed to learn. A possible solution is using virtual environments where the agent collects the necessary experience to learn without incurring the costs of using the real environment. However, with this strategy, the problem now lies in efficiently transferring the experience gained from the virtual environment to the real one. This problem is known as *sim-to-real*.

The main objective of this thesis is to design and implement a methodology applicable to industrial problems that allows to efficiently transfer the experience gained by DRL agents in virtual environments to real setups. The same task has been solved using two different industrial assets to validate the generalization capacity of the proposed sim-to-real methodology. Finally, the general methodology to address sim-to-real transfer in industrial operations other than the one studied in this thesis has also been defined. Currently, this is an open issue in the literature as most works restrict themselves to research setups or applications that are not directly related to the industrial sector. The case study chosen addresses the approaching phase of pick-and-place using a robotic manipulator. The only observations fed to the agent are 64x64 pixel captures taken by a fixed, externally mounted RGB monocular camera. This simplifies the integration with different manufacturers by eliminating the need for proprioceptive information, albeit at the cost of increased computational complexity.

In this research field, which is still in the research phase, several techniques aim to minimize real-world training when transferring virtually acquired experience. Among the most promising approaches, which also align well with industrial applications, are **Domain Randomization** (DR), where the goal is for the agent to encounter such diversity in virtual training that real-world experience is perceived as a sample from a known distribution; **Progressive Neural Networks** (PNNs), whose architecture is based on knowledge sharing between a teacher agent and a student learning the task in a different domain; **Domain Adaptation** (DA), which seeks to merge the characteristics of two domains into a third one; and the integration of **Semantic Knowledge** about the environment into agent training, providing contextual information that allows it to better understand the scene.

1. **Domain Randomization (DR)**

   DR consists in exposing the agent to a wide distribution of virtual scenarios during training so that when transitioning to reality, the real-world observations are perceived as samples from a previously known distribution. First, high-level randomization is applied to certain characteristics of the elements that are part of the scene in a sim-to-sim approach. One of the

most notable conclusions is that for all cases where only one feature is randomized, agents perform better than the baseline model (BM) without DR within the same training time and with greater generalization capacity. Another interesting aspect is that this generalization remains effective when variations of more than one element are applied simultaneously during training. However, the success rate decreases significantly when the complexity is excessively increased (i.e., around five randomized features). This suggests there is a limit to the amount of variability the agent can handle efficiently.

Experiments were also conducted to explore the use of low-level DR by introducing Gaussian noise into the virtual image to better simulate the pixel distribution of real observations, thereby minimizing the discrepancies between the two images. As a result, the zero-shot transfer success rate of the agent trained in the virtual environment improved from the 15.8 % achieved by the BM to 34.1 %. All these results imply that DR should be combined with other techniques to enhance its strengths and achieve a more efficient transfer to real environments.

2. Progressive Neural Networks (PNNs)

PNNs are based on a continuous learning approach, where the knowledge of a "teacher" agent trained in one environment is transferred to a "student" agent trained in another environment or task. This is achieved through lateral connections that allow reusing previously acquired knowledge.

To analyze this architecture, three student agents were put to learn in three simulation environments with increasing difficulty levels. The three agents successfully learned the task, with success rates of 99 %, 94.7 %, and 85.6 %, in the easy, intermediate, and difficult scenes. It should be noted that the performance of the student agents trained in the easy and intermediate environments degraded with respect to that of the teacher in the original task. Apparently, this suggests that there is a partial knowledge loss in the process. However, the performance of the student agent trained in the complex environment is able to maintain a high accuracy level in the teacher's task. This suggests that in cases where the student agent does not learn the new task perfectly, the lateral connections influence dominates over the newly learned network.

Regarding sim-to-real, Progressive Neural Networks (PNNs) managed to successfully transfer knowledge between a teacher trained in the virtual environment and a student trained in the real environment. This is done with a few-shot approach, where the agent, by reusing the knowledge of the teacher's model, needs very few interactions to learn. In this case, 60,000 samples were enough to achieve success rates between 80 % and 100 % in most of the workspace.

3. Domain Adaptation (DA)

Domain Adaptation aims to unify the characteristics of two environments by, for example, altering the observations. This thesis proposes an original architecture based on the Cycle-Consistent Generative Adversarial Network (CycleGAN), called StyleID-CycleGAN (SICGAN), to achieve this correspondence between environments. This approach allows for the translation of virtual observations into observations that maintain a style similar to the real observation so that agents trained in virtual environments with this real-synthetic image can better generalize their behavior when directly transferred to the real scenario.

Experimental results show that the SICGAN model efficiently translates virtual images into real-synthetic images. The subsequent training of the agent in the virtual environment with real-synthetic observations achieves 100 % accuracy in its post-training evaluation. Finally, zero-shot transfer in the real environment shows behavior that exceeds 85 % in almost all the workspace and reaches 100 % in large regions.

4. Semantic Knowledge

As a complement to the previous techniques, the integration of semantic knowledge about the environment has been studied to improve the efficiency of agent learning. Contextual information enhances visual observations with structured information about the relationships between objects. A vector extracted from a knowledge graph embedding (KGE) is used to achieve this. The KGE captures the semantic relationships between entities and their characteristics in a continuous vector space. The proposed architecture concatenates these representations with the activations prior to the policy approximation stage.

Experimental results in different sim-to-sim configurations demonstrate that the incorporation of this information significantly improves both learning time, reduced by up to 60 % in some cases, and the agent's final performance, which increases by up to 15 %.

Proposed Methodology

Comparing the results obtained from PNNs in few-shot transfer with those of DA using the SICGAN model with the zero-shot transfer of the agent trained in the virtual setup, it stands out that the success rates of the agent with DA are better than those of the PNN. Besides, the distribution of failures in the workspace is more balanced, and the lowest success rate is higher than that of the PNN. Additionally, the transfer itself is more efficient since, unlike in a few-shot approach, with zero-shot there is no need to fine-tune the model with real experience. Even considering that the DA solution requires training an additional model, the SICGAN, the accumulated training time still remains lower. This is because learning in the real environment is more time-consuming. Despite the few interactions needed with the environment, the process cannot be speeded up by parallelizing learning among multiple agents as in simulation.

Figure 1 presents our proposed methodology to efficiently transfer experience between domains in industrial applications. It has the following steps:

1. In **Set-up stage 1** simplistically virtualize the real environment. A photorealistic reproduction is not necessary and may even be counterproductive. Define an MDP to solve the given problem bearing in mind that the only observation available is a low-resolution RGB image, from which all the proprioceptive information will have to be inferred. Validate that the agent is able to learn in the simulated environment or iterate otherwise.

2. In **Set-up stage 2** build a dataset of virtual and real labeled images to train a SICGAN that converts low-detail simulation captures into real-synthetic samples.

3. **Train and validate** the performance of the agent in the virtual environment feeding it with the transformed observations (i.e., real-synthetic observations). If validation fails, iterate the process.

4. **Zero-shot transfer** of the agent to the real industrial setup initiating on-process operations.

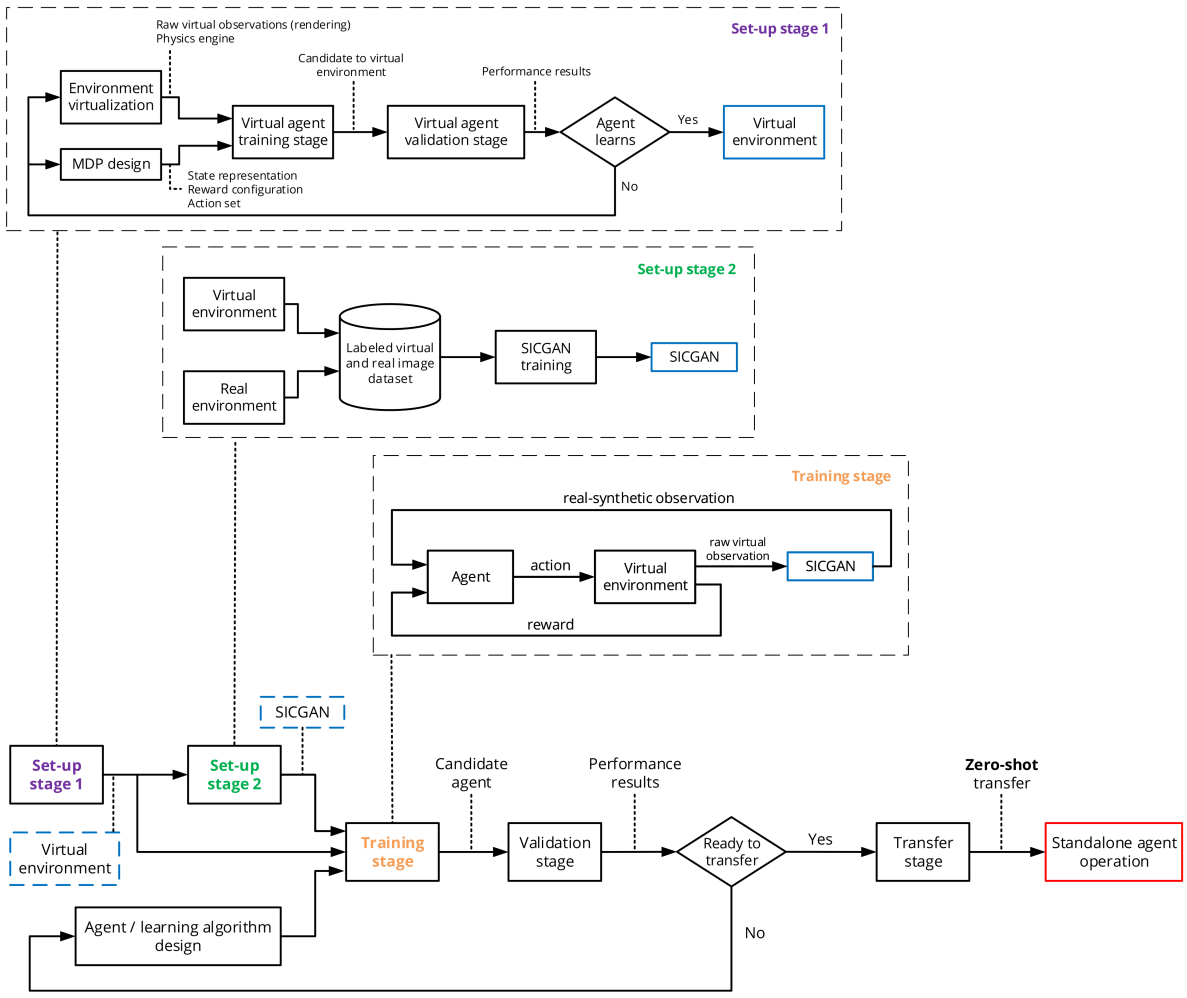


Figure 1. General sim-to-real zero-shot transfer methodology for industrial applications. The final pipeline begins with the set-up stage 1, in which the environment is virtualized and the MDP defined. After validating the virtual environment, set-up stage 2 defines the steps needed to train the SICGAN. Once this is accomplished, the DRL agent can be trained using real-synthetic observations generated by the SICGAN from the raw visual observations. After it is trained and evaluated in the virtual environment, the DRL agent can be deployed in the real environment using a zero-shot transfer.

Resumen

La Cuarta Revolución Industrial ha impulsado la integración de tecnologías avanzadas en la industria, destacando el uso de la Inteligencia Artificial para mejorar la eficiencia de los procesos. En este contexto, el aprendizaje por refuerzo profundo (DRL por sus siglas en inglés) es una solución prometedora para problemas complejos de toma de decisiones secuenciales. Uno de los principales desafíos en este tipo de aprendizaje es la eficiencia muestral, es decir, el número de interacciones agente-entorno necesarias para aprender. Una posible solución es utilizar entornos virtuales donde el agente recopila la experiencia necesaria para aprender sin incurrir en los costes asociados a usar el entorno real. Sin embargo, con esta estrategia, el problema radica ahora en transferir de manera eficiente la experiencia obtenida del entorno virtual al real. Este problema es conocido como *sim-to-real*.

El principal objetivo de esta tesis es diseñar e implementar una metodología aplicable a problemas industriales que permita transferir eficientemente la experiencia adquirida por los agentes en entornos virtuales a escenarios reales. La misma tarea se ha resuelto utilizando dos activos industriales diferentes para validar la capacidad de generalización del procedimiento *sim-to-real* propuesto. Finalmente, también se ha definido la metodología general para abordar la transferencia *sim-to-real* en operaciones industriales distintas a la estudiada en esta tesis. Actualmente, este es un tema abierto en la literatura, ya que la mayoría de los trabajos se limitan a entornos de investigación o aplicaciones que no están directamente relacionadas con el sector industrial. El caso de estudio elegido aborda la fase de acercamiento en una tarea de *pick-and-place* utilizando un manipulador robótico. Las únicas observaciones proporcionadas al agente son capturas de 64x64 píxeles tomadas por una cámara RGB monocular fija y montada externamente. Esto simplifica la integración con diferentes fabricantes al eliminar la necesidad de información propioceptiva, aunque a costa de una mayor complejidad computacional.

Así pues, en este campo que aún se encuentra en fase de investigación existen varias técnicas que tratan de reducir al máximo la cantidad de entrenamiento en el mundo real al transferir la experiencia adquirida virtualmente. Entre las más prometedoras, y que mejor encajan en aplicaciones industriales, se encuentran la **aleatorización del dominio** (DR por sus siglas en inglés), en la que el objetivo es que el agente observe tanta diversidad en el entrenamiento virtual, que la experiencia real sea percibida como una muestra de una distribución ya conocida; las **redes neuronales progresivas** (PNNs por sus siglas en inglés), cuya arquitectura se apoya en el intercambio de conocimiento entre un agente maestro y un agente estudiante que debe aprender la tarea en otro dominio; la **adaptación del dominio** (DA por sus siglas en inglés), la cual busca fusionar las características de dos dominios en un tercero; y la integración de **conocimiento semántico** sobre el entorno en el entrenamiento de los agentes, lo cual les proporciona información contextual que les permite comprender mejor la escena.

1. Aleatorización del dominio (DR por sus siglas en inglés)

DR consiste en exponer al agente a una amplia distribución de escenarios durante el entrenamiento virtual, de modo que, cuando se transfiera a la realidad, las observaciones reales sean percibidas como muestras de una distribución ya conocida. En primer lugar, se aplica una aleatorización de alto nivel a ciertas características de los elementos que forman parte de la escena en un enfoque *sim-to-sim*. Una de las conclusiones más notables es que, en todos los casos en los que sólo se aleatoriza una característica, los agentes superan el modelo base (BM) sin DR en el mismo tiempo de entrenamiento y con una mayor capacidad de generalización. Otro aspecto interesante es que esta generalización sigue siendo efectiva cuando se aplican variaciones de más de un elemento simultáneamente durante el entrenamiento. Sin embargo, la tasa de éxito disminuye significativamente cuando se incrementa excesivamente la complejidad (i.e., alrededor de cinco características aleatorizadas). Esto sugiere que hay un límite en la cantidad de variabilidad que el agente puede manejar de manera eficiente.

También se realizaron experimentos para explorar el uso de DR de bajo nivel introduciendo ruido gaussiano en la imagen virtual para simular mejor la distribución de píxeles de las observaciones reales, minimizando así las discrepancias entre ambas imágenes. Como resultado, la tasa de éxito de la transferencia *zero-shot* del agente entrenado en el entorno virtual mejoró del 15.8 % obtenido por el BM al 34.1 %. Todos estos resultados implican que DR debe combinarse con otras técnicas para potenciar sus fortalezas y lograr una transferencia más eficiente a los entornos reales.

2. Redes neuronales progresivas (PNNs por sus siglas en inglés)

Las PNNs se basan en un enfoque de aprendizaje continuo, donde el conocimiento de un agente “maestro” entrenado en un entorno se transfiere a un agente “estudiante” entrenado en otro entorno o tarea. Esto se logra mediante conexiones laterales que permiten reutilizar el conocimiento previamente adquirido.

Para analizar esta arquitectura, se entrenaron tres agentes estudiantes en tres entornos de simulación con niveles crecientes de dificultad. Los tres agentes aprendieron la tarea con éxito, con tasas de éxito del 99 %, 94.7 %, y 85.6 %, en los escenarios fácil, intermedio y difícil, respectivamente. Cabe destacar que el rendimiento de los agentes estudiantes entrenados en los entornos fácil e intermedio se degradó respecto al del maestro en la tarea original. Esto sugiere que puede haber una pérdida parcial de conocimiento en el proceso. Sin embargo, el rendimiento del agente estudiante entrenado en el entorno complejo mantiene un alto nivel de precisión en la tarea del maestro. Esto sugiere que en casos donde el agente estudiante no aprende perfectamente la nueva tarea, la influencia de las conexiones laterales domina sobre la nueva red aprendida.

En cuanto al *sim-to-real*, las PNNs lograron transferir con éxito el conocimiento entre un maestro entrenado en el entorno virtual y un estudiante entrenado en el entorno real. Esto se consiguió con una aproximación *few-shot*, donde el agente, al reutilizar el conocimiento del modelo maestro, necesita muy pocas interacciones para aprender. En este caso, 60.000 muestras fueron suficientes para lograr tasas de éxito entre 80 % y 100 % en la mayor parte del espacio de trabajo.

3. Adaptación del dominio (DA por sus siglas en inglés)

DA busca unificar las características de dos entornos, por ejemplo, alterando las observaciones. Esta tesis propone una arquitectura original basada en la *Cycle-Consistent Generative Adversarial Network (CycleGAN)*, llamada *StyleID-CycleGAN (SICGAN)*, para lograr esta correspondencia entre entornos. Este enfoque permite la traducción de observaciones virtuales en observaciones que mantienen un estilo similar a la observación real, de modo que los agentes entrenados en entornos virtuales con esta imagen real-sintética puedan generalizar mejor su comportamiento cuando se transfieren directamente al escenario real.

Los resultados experimentales muestran que el modelo SICGAN traduce eficientemente imágenes virtuales en imágenes real-sintéticas. El posterior entrenamiento del agente en el entorno virtual con observaciones real-sintéticas logra una precisión del 100 % en su evaluación post-entrenamiento. Finalmente, la transferencia *zero-shot* en el entorno real muestra un comportamiento que supera el 85 % en casi todo el espacio de trabajo y alcanza el 100 % en grandes regiones.

4. Conocimiento Semántico

Como complemento, se ha estudiado la integración de conocimiento semántico sobre el entorno para mejorar la eficiencia del aprendizaje del agente. La información contextual mejora las observaciones visuales con información estructurada sobre las relaciones entre los objetos. Para lograrlo, se utiliza un vector extraído de un grafo de conocimiento (KGE por sus siglas en inglés). El KGE captura las relaciones semánticas entre las entidades y sus características en un espacio vectorial continuo. La arquitectura propuesta concatena estas representaciones con las activaciones previas a la etapa de aproximación de la política.

Los resultados en diferentes configuraciones *sim-to-sim* demuestran que la incorporación de esta información mejora significativamente tanto el tiempo de aprendizaje, que se reduce hasta un 60 % en algunos casos, como el rendimiento final, que aumenta hasta un 15 %.

Metodología Propuesta

Comparando los resultados obtenidos con las PNNs en la transferencia *few-shot* con los de DA utilizando el modelo SICGAN en la transferencia *zero-shot* del agente entrenado en el entorno virtual, destaca que las tasas de éxito del agente con DA son mejores que las de las PNN. Además, la distribución de fallos en el espacio de trabajo está más equilibrada, y la tasa de éxito más baja es superior a la de las PNN. Además, la propia transferencia es más eficiente ya que, a diferencia de un enfoque *few-shot*, con *zero-shot* no es necesario ajustar el modelo con experiencia real. Incluso considerando que la solución de DA requiere entrenar un modelo adicional, el SICGAN, el tiempo de entrenamiento acumulado sigue siendo menor. Esto se debe a que el aprendizaje en el entorno real consume más tiempo. A pesar de las pocas interacciones necesarias con el entorno, el proceso no puede acelerarse paralelizando el aprendizaje entre múltiples agentes como ocurre en la simulación.

Figura 2 presenta la metodología propuesta para transferir eficientemente experiencia de aprendizaje por refuerzo en aplicaciones industriales. Consta de los siguientes pasos:

1. **Etapa 1**: Virtualizar de manera simplificada el entorno real. No es necesaria una reproducción fotorrealista, e incluso puede ser contraproducente. Definir un MDP para resolver el problema dado, teniendo en cuenta que la única observación disponible es una imagen RGB de baja resolución, a partir de la cual se deberá inferir toda la información

propioceptiva. Validar que el agente es capaz de aprender en el entorno de simulación anterior y, en caso contrario, iterar.

2. **Etapa 2:** Construir un conjunto de datos de imágenes virtuales y reales etiquetadas para entrenar la SICGAN que convierte las observaciones virtuales en observaciones reales sintéticas.

3. **Entrenar y validar** el rendimiento del agente en el entorno virtual durante el aprendizaje con las observaciones transformadas, es decir, las observaciones reales sintéticas. Si la validación falla, iterar el proceso.

4. **Transferencia *zero-shot*** del agente al entorno industrial real, iniciando las operaciones del proceso.

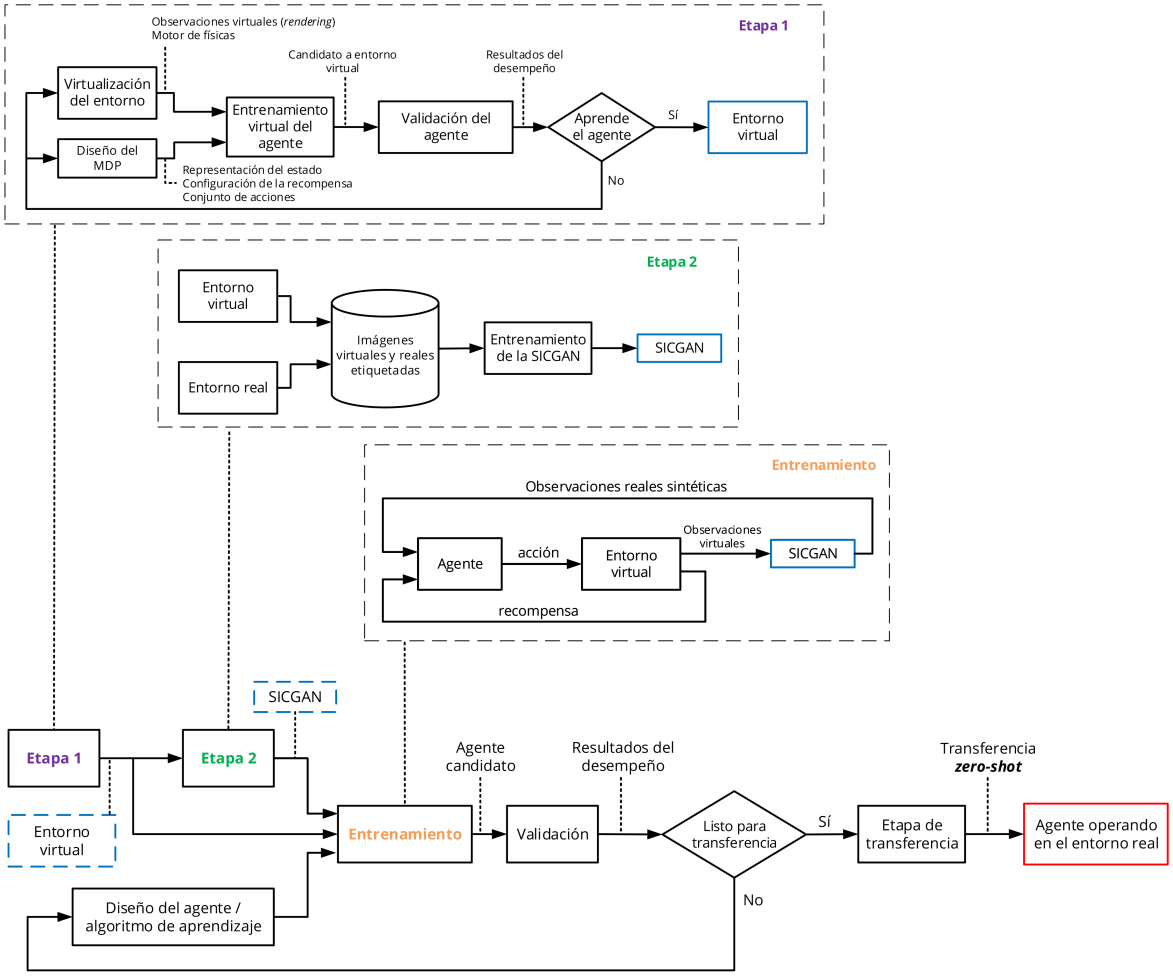


Figura 2. Metodología *sim-to-real* para una transferencia *zero-shot* de experiencia en aplicaciones industriales. El proceso comienza con la etapa 1, en que se virtualiza el entorno y se define el MDP. Tras validarlos, la etapa 2 establece los pasos necesarios para entrenar la SICGAN. Una vez completado, el agente se entrena utilizando observaciones reales sintéticas generadas por la SICGAN. Después de su entrenamiento y evaluación, el agente puede desplegarse en el entorno real mediante una transferencia de *zero-shot*.

1

Introduction

Defend your right to think, because
even thinking wrongly is better than
not thinking at all.
Hypatia of Alexandria (360–415)

This chapter introduces the context of Industry 4.0 and highlights the potential of Deep Reinforcement Learning in industrial applications. It addresses the challenges of sample efficiency and the sim-to-real transfer problem. Once the thesis is framed, the motivation and the objectives are defined, concluding with an overview of the dissertation's structure.

1.1. Industry 4.0

Over the last decade, the industry has embraced a new paradigm known as the Fourth Industrial Revolution or Industry 4.0. This term was introduced in 2011 by Professor Wolfgang Wahlster, Director and CEO of the German Research Center for Artificial Intelligence, to describe the emergence of new technologies and business models within the industry. Klaus Schwab, the founder and executive chairman of the World Economic Forum, argues in his book "*The Fourth Industrial Revolution*" [Sch16] that this era characterized by the convergence of digital, physical, and biological systems, differs from previous industrial revolutions in terms of its speed, scope, and impact. While the Third Industrial Revolution led to significant automation, Industry 4.0 aims to advance further through the cyber-physical transformation of processes, systems, and methods, placing people at the center of the value chain [MRP20].

According to [PR17], the main pillars of Industry 4.0 are *Smart Factories*, *Smart Products*, a new set of *Business Models*, and *Customers*. The concept of Smart Factories will be discussed in more detail later. The defining characteristic of Smart Products is that they must possess a degree of *self-awareness*, i.e., all the information related to their production, history, and current status must be recorded throughout their life cycle. This requirement imposes a series of technological demands on the capabilities and tools that both the factory and the company must have. Business Models have evolved to meet these requirements. New business opportunities have enabled more traditional enterprises to adapt and renew their supply and

value chains, which often have inherent inertia in their operations. Finally, Customers are among the greatest beneficiaries of the Fourth Industrial Revolution's changes. Enhanced user experiences, improved traceability of ordered products, and the production flexibility that allows consumers to purchase partially or fully customizable products with short delivery times are just a few of the advantages that characterize the modern industry-customer relationship.

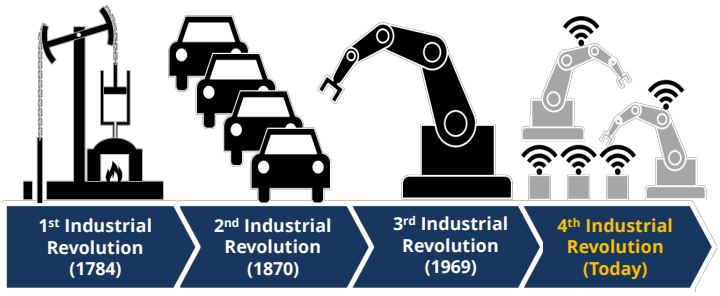


Figure 1.1. Timeline of the Industrial Revolutions. The First Industrial Revolution marks the transition to mechanized production through steam power. The Second Industrial Revolution is characterized by mass production and the widespread use of electricity. The Third Industrial Revolution introduces automation, computers, and electronics into production processes. Finally, the Fourth Industrial Revolution integrates digital, physical, and biological systems, driven by advancements in AI, IoT, cyber-physical systems, etc. Source: Wikimedia Commons. Christoph Roser at AllAboutLean.com licensed under the Creative Commons Attribution-Share Alike 4.0 International license.

As aforementioned, this technology-driven revolution is fundamentally anchored in the concept of Smart Factories. Although the term has generated some controversy due to its various definitions, at its core, it refers to the integration of multiple technologies that culminate in an interoperable manufacturing plant where nearly all supply chain assets are connected, coordinated, and share data. To achieve this level of integration and efficient cooperation, it is essential to have cyber-physical production systems with a degree of *intelligence* or decision-making capability [Xu+21]. Figure 1.2 depicts the key technologies driving these changes: Artificial Intelligence (AI), Big Data, Cybersecurity, Additive Manufacturing, Augmented and Virtual Reality (AR/VR), the Internet of Things (IoT) and the Industrial Internet of Things (IIoT), Cloud Services, and Collaborative Robots [Ort20]. The following section will focus on the role of AI learning algorithms in Smart factories, discussing their advantages and disadvantages.

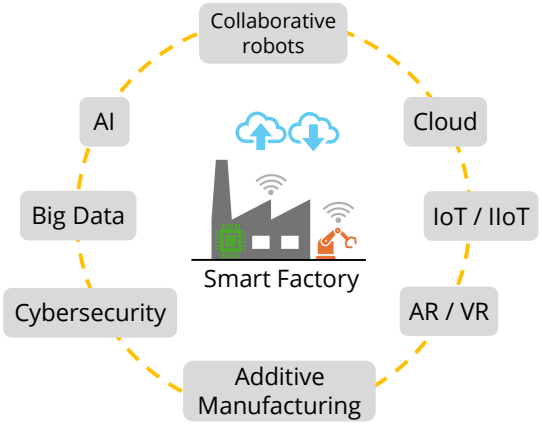


Figure 1.2. Key technologies that drive the definition of a smart factory. These technologies work together to create an interoperable, adaptive, and efficient manufacturing environment where all assets are connected, data-driven decisions are made in real-time, and processes are optimized for greater flexibility and productivity.

1.2. Reinforcement Learning in Industry

AI techniques have moved beyond the research field and into practical applications, driven by advances in software and hardware components and their increasing affordability. AI learning methods can be categorized into *Supervised Learning* [Bis06; PCD08; MY15], *Unsupervised Learning* [Gha04; HTF09; Dik+18] and *Reinforcement Learning (RL)* [SB18; Fra+18]. In supervised learning, models are trained on labeled datasets, meaning that the model also receives the corresponding expected output for each input. In unsupervised learning, algorithms aim to identify patterns or structures within the dataset that can be used to describe the data or predict its future behavior. Conversely, RL is used in sequential decision-making problems, relying on the interaction between an agent and its environment, where experience, rather than data, is accumulated during the learning process. While RL, and its deep version, Deep Reinforcement Learning (DRL) problems may be defined as goal-directed in which an agent can sense and act on the environment according to a reward. They can cope with environments that present variability where the action selection must consider future consequences. Conversely, in supervised learning, the output is based on a dataset built in a certain context, and it is used under the assumption that the layout will remain unchanged. Therefore, it is not designed to address problems where choices depend on the outcomes of previous actions and involve a dynamic and time-dependent process, which is the nature of many industrial operations. Figure 1.3 presents the main machine learning methods and some of their common uses.

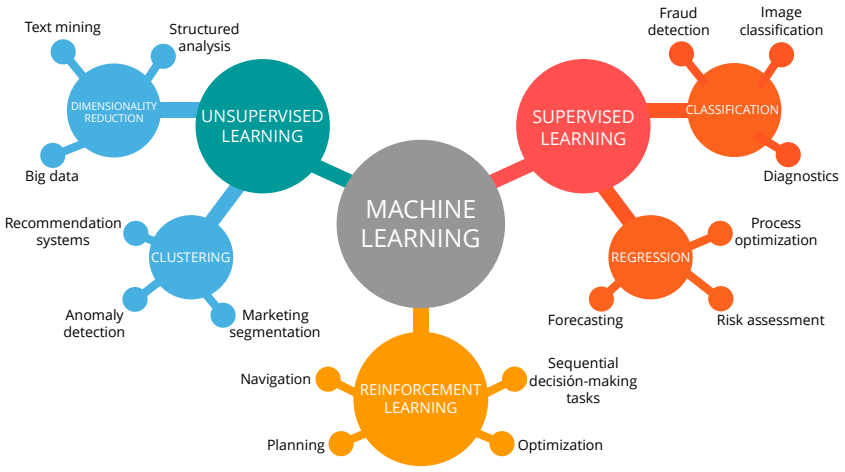


Figure 1.3. Machine learning approaches: supervised, unsupervised, and reinforcement learning, along with their typical applications. Source: Machine Learning Algorithms In Layman's Terms Part 1 by Audrey Lorberfeld.

One barrier slowing down the use of AI in industry is that both supervised and unsupervised learning algorithms are highly *data-intensive*, i.e., training AI models requires vast amounts of data to meet the industry's robustness standards. Furthermore, since industry players often do not share their data with their technology partners, data is not distributed. It is not only a matter of quantity but a trade-off between data quantity and data quality what makes the difference. Approaches like Small Data [FA18] and Data-Centric AI [MSK21; PZ21] encourage a shift from the traditional Model-Centric AI problems by emphasizing the importance of data quality, data understanding, and dataset generation.

In these situations, Reinforcement Learning (RL) algorithms are preferable. RL integrates principles from optimal control theory and dynamic programming. Similar to classical control, RL operates within a closed loop where a system evolves in response to the actions. Nonetheless,

RL has a broader scope and is more ambitious regarding the complexity of the problems it can address. A detailed explanation of the RL fundamentals can be found in Section 2.1.1.

As the industry continues to experience the boost of AI-driven advanced automation, increasingly complex tasks are expected to be automated. In this context, RL can handle high levels of abstraction that are challenging to model with traditional techniques. Therefore, we argue that RL algorithms should be considered over classical approaches such as Adaptive Control, Model Predictive Control, and PID control when the system's dynamics are unknown or difficult to model regarding the system's complexity and the time and economic effort required.

Thanks to the versatility and functionalities of RL, its applicability is broad, both in terms of the tasks it can solve and the sectors in which it can be applied. Examples of applications include forecasting, planning, navigation, recognition, prevention, resource management, scheduling, optimization, and control. The sectors benefiting from RL range from energy, telecommunication, and manufacturing plants to cyber-physical systems, robotics, and smart cities [KSA21; Li+23; Aka+21].

1.3. Motivation: the sample efficiency problem

RL has several advantages over other learning models. However, one of its major drawbacks is that agents need extensive experience, i.e., numerous interactions between the agent and the environment, to learn effectively. This is especially critical in DRL models, which involve much larger dimensional state spaces. Thus, an analogy can be drawn between the *data-intensive* problem faced by supervised and unsupervised learning algorithms and the *sample efficiency* problem in reinforcement learning approaches. The key difference between these issues is that, while supervised learning models rely on ground-truth datasets, the experience required by RL agents can be obtained in other environments, like a virtual simulation, allowing the agent to generalize its learned knowledge. Besides, virtualizing assets or processes, even with their corresponding inaccuracies, and simulating their behavior under different conditions enables efficiently gathering experience without compromising the integrity of the real-world systems.

The accuracy of the virtualization directly impacts the similarity between the agent-environment interaction in the simulation and the real world. Ideally, a simulator would exactly replicate the physical setup. Nevertheless, this is not always feasible, and in some cases, it may not be cost-effective. When there are discrepancies between the simulated and real environments, the agent's performance may not meet expectations when deployed in the real world. To deal with this issue, various transfer learning techniques have been developed to bridge the gap between the simulation and the reality, known as *sim-to-real*. As a result, while gathering experience in a virtual environment helps mitigate the sample efficiency problem, it introduces a new challenge: the need to efficiently transfer synthetic experience to the real world. Figure 1.4 summarizes the shift from the sample efficiency issue to the sim-to-real challenge.

Hence, this Ph.D. thesis focuses on devising an optimized pipeline to efficiently solve the sim-to-real problem for industrial applications. While most industrial processes rely on classical control algorithms that exploit the known operational dynamics, we argue that RL and DRL approaches can not only address the same problems but also tackle more complex challenges that remain unsolved, all while introducing generalization, scalability, and flexibility into the existing

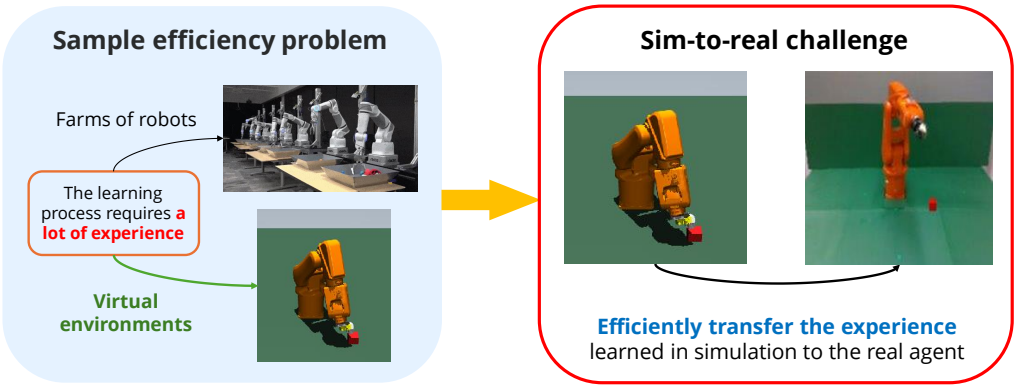


Figure 1.4. Shift from the sample efficiency problem to the sim-to-real challenge. Initially, the focus is on mitigating the high data requirements by utilizing virtual environments for extensive, low-cost agent training. Once the sample efficiency problem is mitigated using virtual environments, the focus shifts to the new challenge of effectively transferring the learned experience from the virtual environment to the real-world setting, addressing the sim-to-real gap.

industrial environment constraints. Particularly, Domain Randomization (DR) [Tob+17], PNNs [Rus+16a], Domain Adaptation (DA) [WD18], and Semantic Knowledge concepts [Dev+18], are the transfer learning techniques and methods studied as the most suitable approaches to the problem solved in this thesis. The goal is to develop a procedure that equips the resulting models with optimized generalization properties, thereby avoiding, on the one hand, the dependency on photorealistic simulators, which could increase the computational resources requirements, and on the other hand, the use of complex algorithms that might lead to task or environment overfitting.

1.4. Thesis objectives

This Ph.D. thesis aims to design, implement, and evaluate an efficient training pipeline that contributes to mitigating the sample efficiency problem of RL and DRL agents, with the final purpose of making these learning approaches more accessible in industrial applications. One possible solution to handle the sample efficiency problem that has been validated in the literature, and that serves as a starting point for this thesis, involves bridging the reality gap, i.e., the discrepancies between virtual and real environments. The general objective can be further particularized in three subgoals:

- **Design an optimized methodology suitable for industrial processes to efficiently transfer experience between DRL agents.** Considering the state-of-the-art sim-to-real techniques presented in Section 2.3, we argue that few of them are appropriate for industrial applications due to several limitations: some may require highly realistic virtual environments, others involve computationally complex training processes, or need a fine-tuning procedure that incurs in unaffordable time and financial costs. Therefore, this thesis explores a hybrid solution that combines the strengths of these techniques while minimizing their drawbacks and respecting industrial constraints.
- **Demonstrate and analyze the generalization capabilities of the developed methodology and techniques.** The pipeline must be validated in a different environment that performs the same industrial operation but with a different asset. The purpose of evaluating an agent trained in one environment and tested in another performing the same task is to demonstrate that the designed solution is not *ad hoc* for a specific layout and will not be limited by the specific assets available in the company.

- **Define the general form of the methodology to address the sim-to-real problem across different industrial operations beyond the specific case addressed in this thesis**. This methodology should outline how the process and assets should be virtualized, the appropriate model inputs, the training and validation pipeline for the agent's learning, how to transfer the learned model to the real environment, and potential recommendations to consider. To the best of our knowledge, such a methodology is a critical gap in the literature, as most research remains focused on specific applications without offering a general approach to the problem, particularly one adapted to the constraints of industrial operations.

Figure 1.5 presents the general pipeline that will be followed to accomplish these objectives. Set-up stage 1 begins with the environment virtualization and the MDP definition that models the task solved. After validating the designs and ensuring that the agent can efficiently learn, we will go through another virtual training stage in which transfer learning techniques that need a previous step before the domain shift can be implemented. Once the DRL agent is trained and its performance results are validated, we proceed with the transfer stage in which, depending on the technique chosen, we can achieve a zero-shot transfer where the DRL agent is directly deployed without requiring real experience, or a few-shot transfer where the DRL agent needs little real samples to fine-tune is knowledge and start operating on-process.

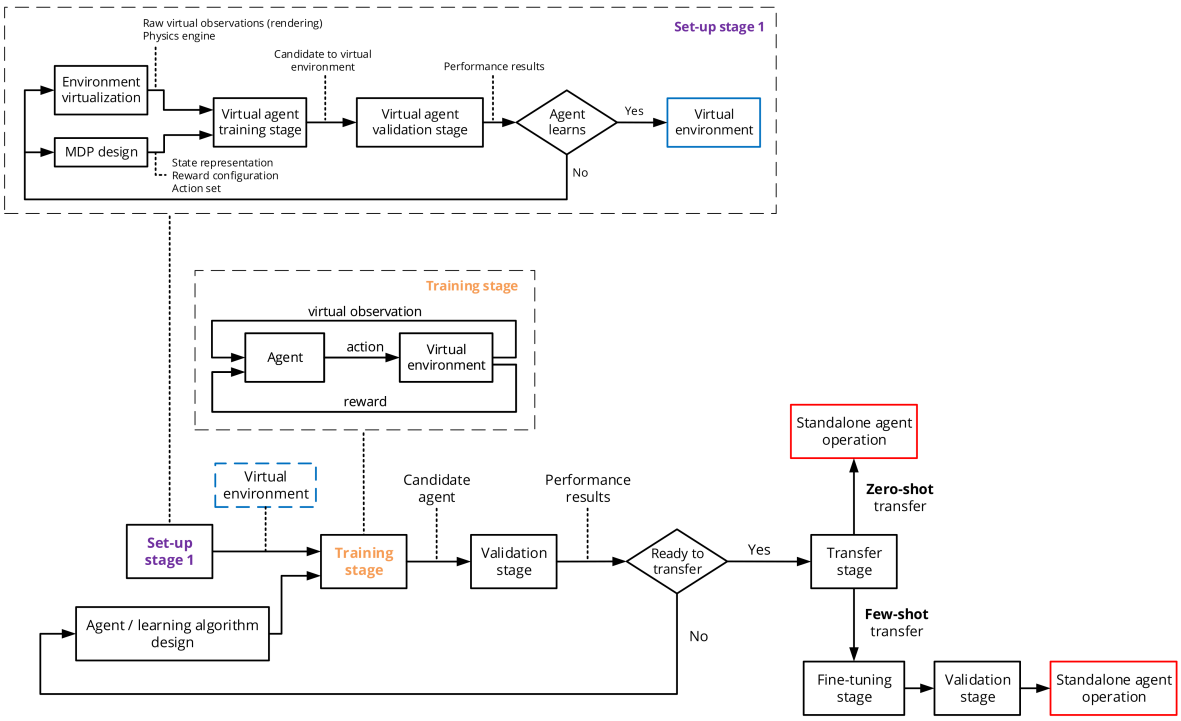


Figure 1.5. General pipeline that will be followed to accomplish the thesis objectives. First, we virtualize the environment and design a proper MDP according to the task solved. After validating them, we train the DRL agent in a second stage, preparing the transfer to the real setup. Depending on the transfer learning technique, we will achieve a zero-shot transfer where the agent can directly start operating, or a few-shot transfer that requires little fine-tuning before beginning the on-process operations.

1.4.1. Thesis limitations and scope

The scope of this thesis is designed to address industrial problems where the process state can be inferred from visual observations. Consequently, the ultimate objective is to establish a general methodology for solving the sim-to-real problem under these conditions. However,

due to limitations in accessing diverse industrial resources, the methodology is developed and implemented by first addressing a problem with a robotic industrial arm, and subsequently validating and assessing its generalization capabilities using a completely different asset. The choice of robotic arms is especially appropriate, as they are among the most common and reconfigurable assets within industrial settings, making them ideal for developing adaptable and transferable solutions across various production environments.

Regarding the learning algorithms employed, since the primary purpose of this thesis is the efficient transfer of experience between deep reinforcement learning agents, a comparison of potential DRL algorithms is not performed. Instead, a single algorithm is selected, grounded in its robust validation within the literature and the empirically satisfactory results it has demonstrated on tasks of a similar nature.

1.5. Dissertation outline

This dissertation is composed of nine chapters, including this introductory one. Chapter 2 offers an extensive literature review that serves as the foundation for this thesis. It begins with a brief explanation of the theoretical fundamentals of RL and DRL, essential for understanding the algorithms that will be employed. Readers who are familiar with the topic can safely skip this section. It then proceeds with a description of the most commonly used virtual environments and physics engines. Following this, the chapter delves into the transfer learning approaches explored in this thesis, providing an overview of their fundamentals and the relevant research conducted in these areas.

Chapter 3 provides a detailed description of the problem to be solved, as well as the setups. First, the task that will be addressed is defined. Next, the virtual and real environments of the asset used during the methodology development are explained. The chapter concludes with the same description of the experimental setups but for the asset used in the methodology validation.

Chapter 4 presents the formalization of the DRL algorithm used in the experiments, the Asynchronous Advantage Actor-Critic (A3C) algorithm, along with its architecture. Afterward, it describes the Markov Decision Process (MDP) design that led to the development of the baseline model used as the reference throughout the thesis, as well as the general training and post-training evaluation procedures.

Chapter 5, Chapter 6, and Chapter 7 focus on one of the transfer learning approaches explored in this thesis. These chapters follow a consistent format modeled after the typical structure of a scientific paper. This design allows the chapters to be read sequentially or independently, as they are structured to be standalone. Each chapter is organized as follows: first, a summary of the technique's fundamentals builds upon the introduction provided in Chapter 2. Next, the chapter's objectives and contributions are outlined. Following this, the method and experiments conducted are described to continue with the presentation of the results and their discussion. Finally, each chapter concludes with a summary of the main findings and future work.

Chapter 5 is centered in DR. The experiments are divided into those using a high-level approach, where DR is applied to visual features of the environment's elements, and low-level DR, where DR is used as Gaussian noise in the virtual observation. Next, Chapter 6 covers PNNs. This chapter is divided into experiments conducted using a sim-to-sim strategy, which analyzes

the PNN learning mechanism, and a sim-to-real approach, which transfers the virtually acquired knowledge to the real environment. Chapter 7 explores DA, specifically the implementation of an original version of the Cycle-Consistent Generative Adversarial Network (CycleGAN), named StyleID-CycleGAN (SICGAN), to address the sim-to-real mismatches in the environment observations. It presents the DRL agent's training in the virtual environment using the simulated translated image and its deployment in the real scenario.

Lastly, Chapter 8 validates the methodology developed but with a different industrial asset. Additionally, it presents the work resulting from the collaboration with the RoCoCo group[1] at the Sapienza University of Rome, where contextual environment information is provided to the DRL agent during training.

Note that the presentation of results in these chapters is not completely linear with respect to the evolution of the research. Initially, we conducted a series of quick experiments in the virtual environment to enable testing in the real scenario as soon as possible. Based on the conclusions drawn from these initial real assessments, we modified some minor initial considerations to improve the transfer process.

The final chapter, Chapter 9, summarizes the conclusions, contributions –including the published and working papers–, and future work. The future work is outlined based on the new research questions that emerged during the course of the thesis and the gaps that might remain open in the search for an efficient sim-to-real transfer process applicable to industrial problems.

As a small tribute to the many remarkable women who, throughout history, have made invaluable contributions to fields such as science, mathematics, engineering, and philosophy– often in times when societal norms made it much harder for them to pursue their professions– each chapter of this dissertation begins with a quote from one of these inspiring figures. These quotes, which are partially related to each chapter's content, serve as a reminder of their resilience, brilliance, and the essential role they have played in advancing knowledge. This is simply a way to acknowledge their achievements and recognize the impact they have had on shaping our lives.

[1] https://labrococo.diag.uniroma1.it/

2

Literature Review

Do research. Ask questions.
Find someone doing what
you are interested in.
Be curious.
Katherin G. Johnson (1918–2020)

This chapter begins by covering the fundamentals of RL and DRL, focusing on some of the key concepts needed to understand the methods implemented in this thesis, with special attention to model-free methods. It can be safely skipped if the reader is familiar with the DRL principles. Then, it provides a comprehensive literature review that lays the foundation for this thesis, going from the general concepts to the more specific matters. After discussing the main simulators and physics engines, the chapter's core focuses on the different transfer learning techniques that bridge the reality gap. The chapter concludes by summarizing the key insights and research opportunities identified in the existing literature.

2.1. Reinforcement and Deep Reinforcement Learning

2.1.1. Fundamentals

This section summarizes the most relevant Reinforcement Learning (RL) concepts, drawing heavily on the content and structure proposed by Sutton and Barto in [SB18] and the Lecture Series by Emma Brunskill from CS234 Stanford's Reinforcement Learning course. RL is a machine learning approach in which an agent learns through the feedback received from its environment as a result of their interactions. As stated in [SB18] "exercising this connection (between the agent and its environment) produces a wealth of information about cause and effect, about the consequences of actions, and about what to do to achieve goals." Thus, RL seeks to replicate the learning mechanism used by humans from birth, *learning by trial and error*, by adapting the actions taken in a given context based on the perceived information about the "value" of those actions.

2.1.1.1. Markov Decision Processes and expected return

A sequential decision-making problem can be formally represented using Markov Decision Process (MDP) [Bel57]. These processes must satisfy the Markov property, which states that the future evolution of the process depends only on its current state and is independent of past history. Consequently, these systems are considered memoryless. Equation (2.1) presents the mathematical expression of the Markov property.

$$P(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_0) = P(s_{t+1} \mid s_t) \tag{2.1}$$

where $P(\cdot)$ is the probability of future states, and s_t is the state s at time step t.

The subsequent presentation of the MDP fundamentals is based on finite MDPs to simplify the demonstrations and mathematical expressions. However, these principles are largely applicable to infinite MDPs with some particular differences in the implementation.

Figure 2.1 illustrates the interaction flow between the agent and its environment and introduces the notation of the main components of an RL setting. The agent observes the environment's state $s_t \in S$, where S is the finite set of all possible states. Based on this observation, the agent selects an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of all possible actions available in state s_t. As a result, the environment provides the agent a new state observation s_{t+1} and a numerical reward $r_{t+1} \in R \subset \mathbb{R}$ that should be maximized over the long-run, and weights how good is the action taken based on the agent's goal. The agent's decision on what action to take next follows a policy $\pi(a|s)$, which maps states to actions and represents the probability of taking action $a \in A(s)$ in state $s \in S$.

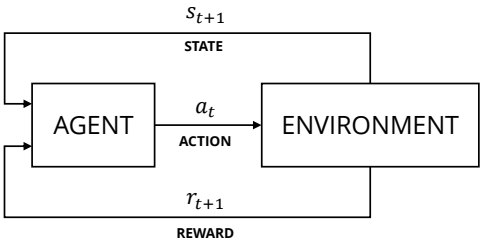


Figure 2.1. Agent-Environment interaction in RL. The agent performs an action a_t in the environment, which gives the new state observation s_{t+1} and a numerical reward r_{t+1} as feedback.

Equation (2.2) presents the discrete probability distribution p that captures the environment's dynamics. The probability of reaching a new state s' and receiving a reward r at time t depends only on the previous state s and the action a taken for all $s', s \in S$, $r \in R$, and $a \in A(s)$. This dependency holds true only if each state contains all the relevant information about past interactions between the agent and the environment, thereby satisfying the Markov property.

$$p(s' \mid s, a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathbb{R}} p(s', r \mid s, a) \tag{2.2}$$

where $p(s' \mid s, a)$ is the probability distribution of the environment's dynamics, $\Pr\{\cdot\}$ is the probability function which determines the probability of S_t being the next state s' given the previous state S_{t-1} and action A_{t-1} is the action taken at time step $t-1$, and r is the reward received.

As mentioned, the agent's goal is to maximize the cumulative reward received at each step. Besides, the agent must account for the delayed effects of rewards. To model this, the expected discounted return is defined as:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

where G_t is the expected discounted return and time step t, $\sum_{k=0}^{\infty}$ is the sum over all future time steps starting from $k = 0$ to ∞, R_{t+k+1} is the reward received at time step $t + k + 1$, and γ is the discount factor, whose value ranges from 0 to 1 ($0 \leq \gamma \leq 1$). γ manages the delayed effect of the action in the reward. If $\gamma = 0$, the agent will focus only on maximizing immediate rewards, whereas $\gamma = 1$ indicates that future rewards are heavily considered. From this point onwards, all mathematical reasoning will assume $\gamma < 1$, which is the most common case in practice.

It is important to differentiate between episodic and continuing tasks. In an episodic sequence, the interaction between the agent and the environment concludes at a terminal state that meets a certain condition after T time steps. After that, a new episode begins from the initial state. Therefore, in episodic problems, $s_t \in S^+$, where S^+ is the set of all possible states S plus the terminal state. In contrast, a continuing task has no terminal state, and the interaction persists indefinitely, with $T = \infty$ time steps. The ongoing formulation and descriptions in this dissertation will focus exclusively on episodic tasks due to the nature of the problems addressed.

2.1.1.2. State-value and action-value functions

In addition to the reward signal received, the agent should also be able to "sense" the value of a state or an action taken. The state-value function is used to measure the value of a state. Equation (2.4) presents the state-value function $v_\pi(s)$ for a state s under policy π. It is defined as the expectation over the discounted return that would be obtained if the agent starts at s and continues acting according to π.

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{2.4}$$

where $v_\pi(s)$ is the value of state s under policy π, $\mathbb{E}_\pi[\cdot]$ is the expectation under policy π, G_t is the discounted return from time step t onward, S_t is state s at time step t, γ is the discount factor, and R_{t+k+1} is the reward at time step $t + k + 1$.

Analogously, the value of choosing action a in state s following policy π afterwards is defined with the action-value function, whose expression $q_\pi(s, a)$ is:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{2.5}$$

where A_t is the action taken at time step t, and the rest of the terms are the same as in (2.4).

Recalling (2.3) and (2.2), which defines the dynamics of the environment, the expression of $v_\pi(s)$ can be presented as:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right], \quad \forall\, s \in S. \end{aligned} \tag{2.6}$$

Where $s' \in S^+$ for episodic tasks, $a \in A(s)$, and $r \in R$.

Equation (2.6) is the Bellman equation for v_π, which, in the end, represents an expectation where the probability of each triple (s', a, r) is calculated as $\pi(a|s)p(s', r|s, a)$. The product $\pi(a|s)p(s', r|s, a)$ captures the combined likelihood of selecting action a in state s according to policy π and then transitioning to state s' while receiving reward r. It is used to weigh the successor states' value and sum over all possible actions, future states, and rewards. For the sake of clarity, from this point onward, it will be assumed that the policies are deterministic, i.e., the policy assigns a probability of 1 to one action among all possible actions in a given state. Figure 2.2 is the wide backup diagram for v_π, which illustrates the relationship between a state and its successors according to the system's dynamics. Ultimately, the backup operation propagates the value of a state backward from its successors.

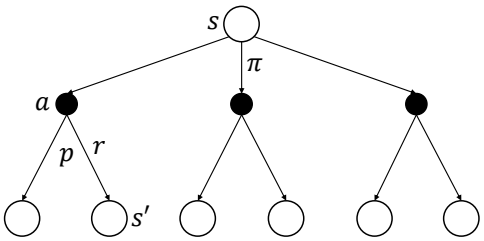


Figure 2.2. Wide backup diagram for v_π. It visually presents the relationship between a state and its successors.

Based on the previous definitions, to maximize the expected discounted return, the agent should seek a policy that enables the selection of the optimal action in each state. This optimal policy π_* is defined as the policy that is better than or equal to the rest of the policies. There may be more than one optimal policy. In this way, the optimal state-value function and the optimal action-value function can be expressed as:

$$v_*(s) \doteq \max_\pi v_\pi(s), \quad \forall\, s \in S. \tag{2.7}$$

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a), \quad \forall\, s \in S \text{ and } a \in A(s). \tag{2.8}$$

Since $v_*(s)$ is the state-value function obtained by the optimal policy π_*, (2.7) and (2.8) can be combined to derive the Bellman optimality equation for $v_*(s)$:

$$\begin{aligned} v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right] \end{aligned} \tag{2.9}$$

This expression indicates that the state value achieved by following an optimal policy is equal to the expected discounted return when the best action is chosen for that state. Similarly, the Bellman optimality equation for $q_*(s,a)$ is:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \end{aligned} \tag{2.10}$$

Regarding the solutions of (2.9) and (2.10), it can be proven that for finite MDPs, the system of equations formed by the expression of $v_*(s)$ for each state, and for $q_*(s,a)$ for each state-action pair, has only one solution. Once $v_*(s)$ and $q_*(s,a)$ are determined, identifying the optimal policy π_* becomes straightforward because any *greedy* policy, i.e., a policy that selects actions based only in the immediate reward, with respect to $v_*(s)$ will be an optimal policy. Note that even though π_* is greedy concerning $v_*(s)$, the delayed effects have already been incorporated into the definition of $v_*(s)$. For $q_*(s,a)$ it is even simpler, as the agent should search for the action that maximizes (2.10). Note that π_* can be only computed from $v_*(s)$ if the environment dynamics $p(s', r \mid s, a)$ are known.

2.1.1.3. Policy evaluation, policy improvement, and policy iteration

As briefly mentioned in Section 1.2, RL incorporates concepts and algorithms from dynamic programming, which are used to obtain an optimal policy. In practice, dynamic programming algorithms are not typically used to solve RL problems because they assume perfect knowledge of the environment's model and involve a high computational cost. However, as demonstrated in [SB18], being familiar with some of these methods is interesting for understanding the procedures underlying RL algorithms. In the following paragraphs, dynamic programming will be employed to compute the Bellman optimality equations, (2.9) and (2.10), defined in Section 2.1.1.2.

- **Policy evaluation:** The search for the optimal policy begins by computing an arbitrary v_π. As discussed in Section 2.1.1.2, if the environment's dynamics are known, (2.6) can be solved as a system of linear equations. Nevertheless, it is often preferable to use the *Iterative Policy Evaluation* method to speed up the computation. This approach sets an arbitrary initial value for v_0 and then iterates over a sequence of approximate state-value functions $\{v_k\}$ using the Bellman equation for v_π.

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right], \quad \forall\, s \in S. \end{aligned} \tag{2.11}$$

  Consequently, the operation needed to estimate the value of $v_{k+1}(s)$, known as the *expected update*, relies on the old values of the successor states and the expected immediate return, both weighted by their respective probability. The convergence of $\{v_k\}$ to v_π as $k \to \infty$ is proven, but it is out of the scope of this dissertation.

- **Policy improvement:** After computing the state-value function for an arbitrary policy π and, by analogy, the action-value function, it is necessary to determine whether switching to a new policy π', where the action $\pi'(s) = a \neq \pi(s)$, would result in a higher value compared to the current $v_\pi(s)$. If this leads to an improvement, the policy should be

updated; otherwise, it should remain unchanged. This policy improvement process involves searching for and adopting a new policy that is better than the original by selecting greedy actions with respect to the state-value function of the original policy. The formalization of the *Policy Improvement Theorem* starts with the next relation:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \tag{2.12}$$

$$v_{\pi'}(s) \geq v_\pi(s) \tag{2.13}$$

Thus, for the action $a = \pi'(s)$, if $q_\pi(s, a) > v_\pi(s)$ it can be concluded that π' is better than π for all $s \in S$. The theorem-proof can be easily derived by substituting the definition of the Bellman equation for $q_\pi(s, a)$ into (2.12). The complete procedure is detailed in Chapter 4 of [SB18].

Instead of focusing on a single state, we can think of applying the policy improvement process to all possible states and actions. Selecting in each state the best action according to $q(s, a)$ involves following the new greedy policy π', which is defined as:

$$\begin{aligned} \pi'(s) &\doteq \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \arg\max_a \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right] \end{aligned} \tag{2.14}$$

Therefore, this policy improvement process resulted in an improved policy from an initial one by making it greedy in relation to the value function of the original policy. If the new policy π' is as good as π, then $v_{\pi'} = v_\pi$ and, according to the based definition of π' for all $s \in S$:

$$\begin{aligned} v_{\pi'}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma v_{\pi'}(s')\right] \end{aligned} \tag{2.15}$$

which is the meaning and form of (2.9). Therefore, if $v_\pi = v_{\pi'} = v_*$ then π and π' are optimal policies. The policy improvement procedure results in the optimal policy unless the initial policy is already one of the set of optimal policies.

- **Policy iteration and value iteration** Concatenating the above processes where a given policy $\pi(s)$ is first evaluated by calculating the state-value function (*policy evaluation*) and then improved to a better policy $\pi'(s)$, unless $\pi(s)$ is already optimal (*policy improvement*) results in a complete *policy iteration* methodology. For finite MDPs, this method is guaranteed to converge to an optimal policy $\pi_*(s)$ and, consequently, an optimal state-value function $v_*(s)$.

  In addition to the iteration between these two operations, there is also an internal loop within the policy evaluation step, which can increase computational cost and convergence time. To mitigate this, the *value iteration* method can be used as an alternative without losing convergence guarantees. The main difference is that, in value iteration, the policy evaluation is stopped after one state update for all $s \in S$. This allows evaluation and

improvement to occur simultaneously until the state-value function changes fall below a certain threshold, as the following expression shows:

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\right] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right], \quad \forall\, s \in S. \end{aligned} \tag{2.16}$$

Another way of improving the convergence speed is by using asynchronous dynamic programming techniques, which allow policy evaluation and policy improvement to be performed without waiting for the other to complete. This idea is known as *Generalized Policy Iteration (GPI)*, where the key point is that both processes are carried out simultaneously and interact with each other. The iteration finishes when both processes stabilize and have negligible changes, as in the previous procedures, resulting in an optimal policy and state-value function. Figure 2.3 illustrates the dynamics of GPI in a two-dimensional space. The two black lines represent the paths toward two different goals, each with its own set of possible solutions. However, in the end, both paths converge to a common optimal result.

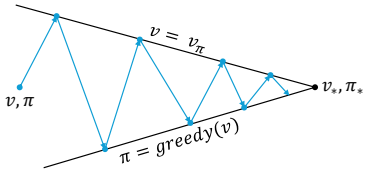


Figure 2.3. Generalized Policy Iteration (GPI) diagram, where policy evaluation and policy improvement interact iteratively to converge towards an optimal policy.

2.1.1.4. Model-based and model-free reinforcement learning

When solving sequential decision-making problems using RL, the environment's model may or may not be available. The RL algorithms that can be applied if the environment's dynamics are known are tagged as *model-based*. Given a state-action pair, the agent predicts the transition probabilities and, thus, the next state and the expected reward. Model-based approaches are mainly characterized by *planning* while model-free, by *learning*. According to [SB18], *planning* is the process where a policy is built or enhanced using the model as input. The optimal policy can be searched through the state space, taking action and computing the value functions. These methods often require fewer samples to learn since they can generate *simulated experience* from the known model. However, building the model is computationally expensive and not always feasible. For more detail on model-based solutions, refer to [SB18] Chapter 8.

Due to these limitations and the nature of the problems this dissertation solves, the focus will be on *model-free* approaches. Instead of deriving the model, the agent learns from interactions with the environment by trial and error, updating the value function or policy based on the states and rewards observed. Although they require more samples, model-free methods are more computationally efficient than model-based. The next point briefly defines some of the most relevant model-free algorithms.

2.1.1.5. Value-based and policy-based methods

Finding an optimal policy, π_*, can be achieved either by learning $v_*(s)$, or $q_*(s, a)$, that implicitly lead to π_*, or by learning how to approximate directly π_* [Aru+17]. The former methodologies are *value-based*, whereas the latter are *policy-based*.

Value-based methods learn π_* by iterating over $V(s)$, which represents the array that estimates $v_\pi(s)$, or $Q(s,a)$, which represents the array that estimates $q_\pi(s,a)$, and choosing the action that maximizes the state-action value. They require exploration strategies over the state-action space, such as ε-greedy. Conversely, policy-based methods seek to explicitly find the approximation of π_*. Generally, they have better convergence properties, even though eventually, they can converge to a local instead of a global optimum.

Before describing some of the popular algorithms according to this classification, it is necessary to point out the difference between those methods that use *tabular* strategies and those which use *function approximation* to search for the representation of $V(s)$, $Q(s,a)$, or π. Tabular methods are appropriate for low-dimensional state spaces since they use vectors and matrices to represent the functions. Nonetheless, this approach is not affordable when the problem complexity increases. The functions should be parameterized to represent a general expression over all states, actions, or policies compactly to solve this. In function approximation, the objective is to find the set of parameters $\theta_1, \theta_2,$ and θ_3 that maximize the return. Hence, now the estimated functions are described as $\hat{V}(s;\theta_1)$, $\hat{Q}(s,a;\theta_2)$, or $\hat{\pi}(s;\theta_3)$. A frequent way to perform the approximation is using Neural Networks (NNs) or Deep Neural Networks (DNNs) [LBH15; GBC16]. When a NN architecture is used not only to approximate values but also to automatically extract the Markov signal for the MDP (i.e., to learn the state representation), Deep Reinforcement Learning (DRL) emerges, combining the frameworks of RL and deep learning.

With the growth in the popularity of RL and DRL, the number of available methods has notably increased in recent years. Since the detailed explanation of all of them is out of scope, the algorithms defined here are some of the ones appropriate for tackling DRL problems, which is the nature of the challenge addressed. As value-based approaches stand out Deep Q-Network (DQN) [Mni+15], which extends the use of Q-learning by introducing DNN to approximate $Q(s,a)$; Double Deep Q-Network (DDQN) [vHGS15], which uses two networks to decouple the action selection from the value estimation to solve the overestimation bias in the DQN; Dueling Deep Q-Network (Dueling DQN) [Wan+16b], which improves the learning efficiency of DQN by separating $Q(s,a)$ into two streams, one for estimating the value of a state, and other for the advantage of the action; and Rainbow [Hes+18], which combines DQN, Dueling DQN, Prioritized Experience Replay [Sch+15a], Distributional DQN [BDM17], and Noisy DQN [For+17] into a single learning agent.

By contrast, some of the highlighted policy-based methods for DRL problems are Proximal Policy Optimization (PPO) [Sch+17], which improves the efficiency of policy gradient methods by preventing large policy updates; Trust Region Policy Optimization (TRPO) [Sch+15b], which avoids huge deviations from the current policy by enforcing a trust region constraint; and REINFORCE [Wil92], a foundational RL algorithm which enables learning in environments with continuous actions or large action spaces.

As Figure 2.4 shows, in the intersection of value-based and policy-based methods, there is a set of algorithms that approximate at the same time $V(s)$ and $\pi(s)$. These are the *Actor-Critic* approaches. The term *Actor* refers to the policy part, whereas the *Critic* is used for the state-value function estimation. Some of the main advantages resulting from the merger are that, in general, they tend to be more sample-efficient than the other alternatives. They are also more stable due to the variance decrease thanks to the baseline provided by the critic during the policy updates. Moreover, they are mainly online methods, which means that they update $V(s)$ and $\pi(s)$ during the interaction with the environment following the same $\pi(s)$, something that enables a rapid

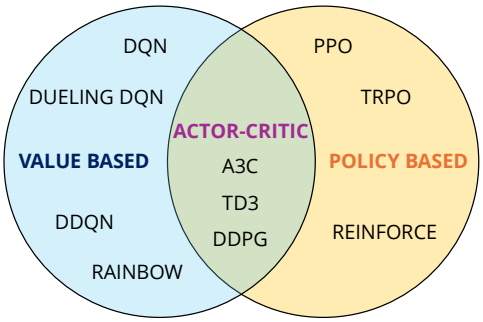


Figure 2.4. DRL algorithm classification diagram. Value-based methods estimate $V(s)$ or $Q(s,a)$; policy-based methods directly optimize π; and actor-critic methods combine both strategies by using an actor to determine actions and a critic to evaluate them.

adaptation to changes. Although the best DRL algorithm will depend on the problem, actor-critic approaches offer, in general, a more effective approach by leveraging the strengths of value-based and policy-based methods while reducing some of their limitations. Some of the most relevant actor-critic methods are the Deterministic Policy Gradient (DPG) [Sil+14], and its deep version (DDPG) [Lil+16], which is suitable for continuous action spaces and extend the DPG theorem to update the policy parameters, and Twin Delayed Deep Deterministic Policy Gradient (TD3) [FvHM18], an algorithm that uses Q-networks to reduce the overestimation bias, improving thus the original DDPG. Advantage Actor Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C) are two actor-critic approaches that learn simultaneously the $\pi(s)$ (actor) and the $V(s)$ (critic) to estimate the advantage function, denoted as $A(s,a)$, which measures how much better it is to take action a in state s compared to the average performance expected from that state, represented by $V(s)$. In other words, the advantage function quantifies the relative benefit of choosing a specific action a over simply following the policy. It is calculated as the difference between the action-value function $Q(s,a)$ and the state-value function $V(s)$. A2C and A3C [Mni+16] use bootstrapped returns for updating the networks, but A3C parallelizes the training through different agents that perform the update asynchronously, resulting in a higher sample efficiency and scalability than A2C. A3C is explained in more detail on Section 4.1 since it is the algorithm that will be implemented. The choice of A3C is based on the advantages presented above and the fact that it is a very well-known DRL algorithm within the research community due to its results in discrete action spaces [Gro+12; Bab+17].

2.2. Simulators and physics engines

Most DRL research relies on virtual environments and physics engines to gather the experience an agent needs to learn. The alternative will use real experience collected by many identical assets, like a "farm" of robotic arms picking pieces, which is usually unaffordable due to its economic cost and the time required.

We understand a virtual environment as a virtual representation in a simulator that allows the virtualization of a scenario and its elements, representing them in a visually simplified or photorealistic manner. Conversely, a physics engine is software that simulates the physical behavior of the scenario and its components. Not all simulators have an integrated physics engine, yet most have APIs that allow integrating many physics engines. Several software options are available to deploy virtual environments in DRL. Although specialized software can be found depending on the field, this literature review focuses on those that are more common for robotic control in the research community.

Multi-Joint dynamics with Contact (MuJoCo) [TET12], from Google DeepMind[1], is an open-source virtual environment framework with an integrated physics engine. It supports accurate and robust simulation of complex dynamic systems, making it suitable for various applications, including robotic manipulation and control tasks. MuJoCo specializes in modeling and simulating articulated rigid bodies, joint dynamics, contact forces, and frictions. It is considered one of the most efficient simulators, allowing real-time performance analyses. Models and scenarios are completely customizable through an XML-based domain-specific language. The supported programming languages to interact with the simulator are C++ and Python. It also offers great compatibility with the most popular DRL frameworks.

Unity [Haa14], from Unity Technologies[2], is a proprietary game engine born to design and develop video games and AR/VR applications. Still, it is used in many other sectors, such as manufacturing, automotion, and architecture. One of its strengths is that it offers almost photorealistic 3D environments, with the option of customizing lighting effects, textures, materials, etc., using built-in or third-party tools. It includes the possibility of integrating the NVIDIA PhysX[3] physics engine if the scenario is 3D, while if it is 2D, it comes with an integrated engine based on Box2D[4]. The NVIDIA PhysX is a high-performance and accurate simulator that can work on complex systems like rigid or soft body dynamics, positioned-based dynamics, and custom geometries. Unity supports multiple programming languages like C# and JavaScript, and it provides a machine learning toolkit and an open-source framework to train DRL algorithms.

Gazebo [KH04], from the Open Source Robotics Foundation (OSRF)[5], is an open-source software focused on robot control simulation. This virtual environment offers four integrated physics engines: ODE (Open Dynamics Engine), Bullet Physics, Simbody, and DART. Gazebo provides 3D visualization tools and a GUI, making the environment and model design phase easier. It also offers 3D rendering in real-time. One of its main advantages is that it supports the integration of various sensors used in robotics like LiDAR, IMUs, GPS, etc., which enables sensor fusion techniques that make the simulation even more similar to reality. Since it is focused on robotics, Gazebo is integrated with Robotic Operating System (ROS)[6], a framework for robotic software development. It can be programmed in C++, Python, and uses XML for the robot models.

CoppeliaSim [RSF13], from Coppelia Robotics[7], is a proprietary software for robotic simulation development. CoppeliaSim can be integrated with five physics engines: ODE, Bullet, MuJoCo, Vortex Studio, and Newton Dynamics. They all allow simulating complex interactions between the elements on the scene. Like Gazebo, it has 3D visualization tools and real-time 3D rendering, and includes a GUI to design the environments. The robot models are fully customizable and can integrate different sensors. It is compatible with ROS, facilitating the deployment of algorithms in real assets. CoppeliaSim is compatible with programming languages like Lua, C, C++, and Python.

[1] https://mujoco.org/
[2] https://unity.com/
[3] https://www.nvidia.com/en-us/drivers/physx/physx-9-19-0218-driver/
[4] https://box2d.org/
[5] https://gazebosim.org/home
[6] https://www.ros.org/
[7] https://www.coppeliarobotics.com/

Finally, Gymnasium, previously known as OpenAI Gym [Bro+16], from the Farama Foundation[8], is an open-source virtual environment toolkit for developing and evaluating DRL algorithms. It incorporates a wide collection of environments that serves as some of the most important benchmarks of the research community. Each environment is a Python class with a series of methods, some of which are common to all Gymnasium environments, and others are particular to specific environments. This standardization and abstraction allow researchers to focus on the development of the agent model and forget about the definition of the environment. Apart from the predefined scenarios, Gymnasium offers the possibility to create custom environments or modify existing ones, inheriting all their integration capabilities. Depending on the environment, the physics engine is MuJoCo or Bullet Physics.

Table 2.1. Simulation software comparison.

Software	Main Research Applications	Physics Engine	Render Capabilities	Programming Language	License	Support
MuJoCo	Robotics Biomechanics	Proprietary	Advanced 3D visualization	C++ Python	Open-source	Documentation GitHub forums Large user community
Unity	Gaming AR/VR	NVIDIA PhysX (for 3D) Proprietary (for 2D)	High-quality 3D graphics	C# JavaScript Python	Proprietary (free license under conditions)	Documentation Official forums Community support
Gazebo	Robotics Autonomous systems	ODE Bullet Physics Simbody DART	Realistic 3D rendering Environmental effects	C++ Python	Open-source	Documentation Community forums
CoppeliaSim	Robotics	ODE Bullet MuJoCo Vortex Studio Newton Dyn	3D visualization	Lua C C++ Python	Proprietary (free license under conditions)	Documentation Community forums Commercial support
Gymnasium	Robotics Testing algorithms	MuJoCo Bullet Physics	Basic 2D rendering 3D visualization	Python	Open-source	Documentation Community forums

Table 2.1 gathers the simulators described and their most essential characteristics. Although all simulators are suitable for the requirements of this thesis, we argue that being open-source is a critical factor to consider since it may correlate with the size of its community, and the results can be shared for its validation. We chose MuJoCo because it allows fast low detail rendering and can also be integrated with Unity if a more photorealistic render is needed, combined with a customized Gymnasium environment. These are widespread frameworks in the research community that offer accurate simulation results and enough rendering capabilities.

2.2.1. Digital twins, digital shadows and digital models

Nowadays, it is fairly common to hear about the use of digital twins in some industries to enhance maintenance tasks, predict the remaining life of assets, or improve the control of a process based on data that mimics actual behavior. After describing the virtual environment simulators, it is essential to distinguish between a digital or virtual model, our proposal, and a digital twin.

A *digital twin* is a virtual representation that mimics a real-time process or asset. Its simulation relies on the data collected by sensors and models that perfectly simulate the

[8] https://gymnasium.farama.org/

behavior to achieve a high accuracy level. These data are constantly improving the model. Hence, the communication is bidirectional, from the digital twin to the real world and vice versa. Some of the main drawbacks of digital twins are the economic cost, sensorization, data management, and model building, making it unaffordable for most companies. Another similar approach is the *digital shadow*. It uses a more simplified representation than a digital twin to provide a high-level abstraction of the system's key features. It captures the essential aspects, deleting unnecessary details for the process control or maintenance. Like in digital twins, data flows from the real system to the virtual in real-time, which can be used to improve the shadow model. Nonetheless, whereas digital twins aim to enhance the operation, reduce production times, and optimize resources, digital shadows only capture meaningful insights and information from the system status and behavior.

Finally, a *digital model* is a more general concept. It can be described as a digital representation that does not depend on real-time data and is built upon pre-existing knowledge. Its virtualization does not need to be as accurate as in a digital twin. In this case, the communication only flows from the model to the real scenario, even though, asynchronously, real data or experience can be used to improve the digital model. It is cheaper than digital twins and shadows since there is no need for sensorization or complex communication platforms. Table 2.2 gathers the commented differences between the three alternatives. Based mainly on economic reasons and seeking an easy-to-deploy solution suitable for any industry, this thesis relies on digital models.

Table 2.2. Digital twin, digital shadow, and digital model differences.

Solution	Model representation accuracy	Communication
Digital twin	Exact replica	Real-time Real system ↔ Digital system
Digital shadow	Simplified representation with high accuracy	Real-time Real system → Digital system Asynchronous Real system ← Digital system
Digital model	Representation built upon pre-existing knowledge with an adequate accuracy	Asynchronous Real system ↔ Digital system

2.3. Transfer learning in DRL

The use of models trained through DRL algorithms in virtual environments mitigates the sample efficiency problem, but forces to develop techniques to transfer the synthetic experience to the real assets. As the differences between these two domains increase, the *sim-to-real* gap widens.

[ZLZ20] explores several types of transfer learning techniques. In *reward shaping*, the reward structure of the target domain is modified using external knowledge to guide the agent's learning process by assigning auxiliary rewards to beneficial state-action pairs. Another alternative is *policy transfer*, either through policy distillation, where knowledge from a teacher agent is transferred to a student agent, or through policy reuse, where source policies are directly applied to the target task. *Inter-task mapping* is a method for establishing mappings between the state or action spaces of the source and target domains, allowing knowledge transfer across different but related tasks. Finally, *representation transfer* shares feature representations or latent

spaces across domains, often leveraging neural networks to share invariant features between tasks. Policy and representation transfers are covered in Section 2.3.3 and Section 2.3.4, respectively.

According to the level of refinement the virtually-trained agent needs once in the real environment, there are three types of transfer. From the lowest to the highest degree of adjustment, in the first place is *zero-shot* transfer [Kir+23], where the agent can be directly deployed in the real scenario, maintaining the performance achieved in the virtual domain. In other words, the agent is transferred from the source to the target domain without domain-specific experience. In the case of DRL agents, this implies that if the model input, i.e., the observation, is an image, the virtual environment must be sufficiently similar to have few visual discrepancies. This can be achieved either with a photorealistic simulator or by processing the image before the agent sees it. Besides, the dynamics of the virtual model must be accurate enough so that the agent's actions respond similarly to the real controlled system. DRL agents that achieve zero-shot transfer have learned to successfully extract the key features and representations of the environment so they can generalize the knowledge learned and apply it to other related domains. The main drawback of zero-shot transfer is that in model-free DRL approaches achieving this similarity on the visual field and model's dynamics can be challenging and computationally expensive.

Secondly, in *few-shot* learning the agent is refined with a small amount of domain-specific experience when deployed in the real scenario. However, the training span is shorter than that of the agent trained in the virtual environment. While the model-trained parameters remain frozen in zero-shot, here they serve as an initial point from which the agent starts to refine and adapt its knowledge with real experience. Another alternative commonly used in few-shot learning is to freeze some weights of the architecture and retrain only specific layers.

Lastly, when the amount of new experience needed to adjust the agent's performance to the target domain is large, we talk about *fine-tuning*. The term fine-tuning can be found in the literature when referring to different processes. It can be applied to the adjustment of the model's hyperparameters according to the new context, or to the process where the model's weights change according to the new experience gathered by the agent in the target domain. In the second case, the initialization is performed as in few-shot, but the difference is that the amount of real experience needed by the DRL agent is larger. The disadvantage of fine-tuning is that, although it is very simple, it is a very demanding methodology for complex tasks, requires tedious engineering tuning efforts, and there is no guarantee of success. Table 2.3 summarizes the differences between these three transfer learning types.

Table 2.3. Comparison of transfer learning approaches.

Transfer Type	Experience gathering	Performance refinement
Zero-shot	Directly deployment in the real environment. No real-world experience is required.	No refinement. Accurate virtual simulations.
Few-shot	Additional but limited real-world experience.	The agent refines its virtually learned knowledge, using a few real-world interactions.
Fine-tuning	Extensive real-world experience for full adjustment, retraining all parameters in the target environment.	Full retraining is done using real-world data, ensuring complete adaptation.

After discarding the fine-tuning option as being too complex for the problem at hand, having no guarantee of success, and being the worst alternative if the transfer has to be efficient, several techniques can be used to achieve zero-shot or few-shot transfer. These are Domain Randomization (DR), Domain Adaptation (DA), Knowledge Distillation, Progressive Neural Network (PNN), Semantic Knowledge or Object-Centric Approaches, Meta Reinforcement Learning, and Imitation Learning.

2.3.1. Domain Randomization

DR [Tob+17] is one of the most widespread transfer learning techniques in the community due to its ease of implementation and its adaptability to the majority of DRL problems.

It relies on training an agent in a wide distribution of virtual environments with randomized features. This way, when the agent faces the real scenario, it will be more likely to act as if this new scenario was another stage of the virtual setup, being able to generalize the learned knowledge accurately. As stated in [Pen+18b] "discrepancies between the source and target domains are modeled as variability in the source domain."

[Tob+17] shows that training models in simulated environments with enough variability can make the agent agnostic to the workspace. Their task is an image-based object detector where DR is applied to multiple environment characteristics such as the texture and material of the scene objects, the camera position (the most relevant feature according to the results), and light conditions. [Ren+19] continues this research line by developing a pose estimation using DR that, considerably improves the state-of-the-art results. As an extension of [Tob+17], [MJD18] trains a robotic arm that manipulates deformable objects in a virtual scenario applying DR to some visual characteristics and the camera pose. Despite the high success rate in the virtual setup, the agent's performance in the real scenario decreases between 30 % and 70 %, depending on the task.

Without using environment images as input, [Pen+18b] tackles the reality gap by applying DR to the robot dynamics in the simulator. They show that the robotic arm can generalize policies learned, keeping a similar success rate in the real environment without further training. [TAK23] introduces DROPO (Domain Randomization Off-Policy Optimization) to estimate the DR distributions. They use a likelihood-based approach that recovers dynamic parameter distributions using pre-collected offline datasets. They utilize a probabilistic metric to fit dynamics parameter distributions to the gathered human demonstration or to pre-trained policies. Then, these optimized dynamics distributions are used to train the policies in simulation and transfer them in a zero-shot approach to the real world. [Hub+23] study the correlation between DR-based quality criteria and sim-to-real transferability. They generate and deploy a dataset of robot grasping trajectories using Quality-Diversity (QD) methods, combining object state, joint state, and friction perturbations to overcome real-world uncertainties. They identify DR variants that reliably discriminate transferable grasps. However, despite these advancements they could not achieve successful sim-to-real transfer without manual setup adjustments and encountered limitations due to physical simulation inaccuracies.

[And+20] combines distributed DRL and DR to learn optimal policies for dexterous in-hand manipulation. The agent is virtually trained with randomized physical and appearance features. Since they use Unity during training due to its high render capacity and MuJoCo for the physics simulation and rendering during evaluation, they argue that a previous sim-to-sim stage might be convenient. Their outcomes show that despite the direct transfer of the learned policies to

the real scenario, there is a deterioration in the agent's performance. [Moz+20] improves the agent's generalization capability using DR during training to perform adversarial attacks. They add random unrealistic perturbations and implement an intervention-based invariant transfer learning method (IBIT).

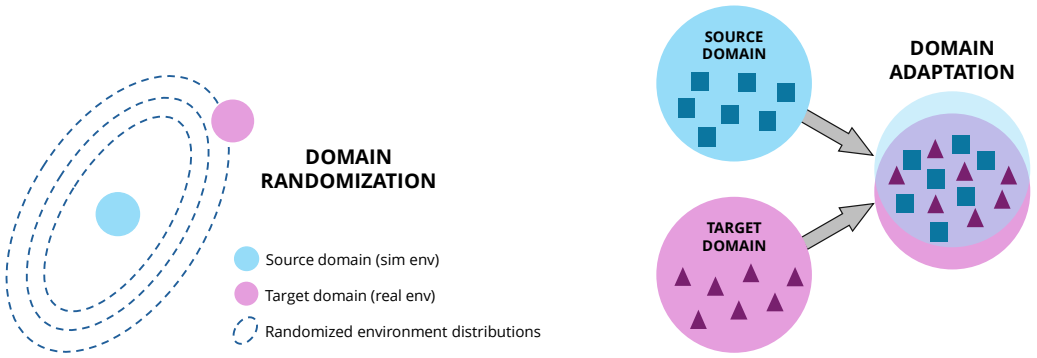
Nonetheless, these proposals have one main drawback. Obtaining high-quality observations that capture light or texture details in the objects and the scenes, requires significant computational and hardware resources. Even though the outcomes presented in [AJ21] suggest that using a small number of high-quality images is preferable to using a large number of low-quality ones, this finding may not apply to tasks involving DRL, where the amount of experience is crucial. Therefore, if rendering observations is time-consuming, training times are bound to rise. In addition, when gathering the observations in the real setup, the hardware equipment should be good enough to capture these details in the scene, which might not happen in an industrial setup.

An open question in DR is how many features should be randomized. Even though it would be technically possible to randomize all the parameterizable characteristics and cover all the scenario configurations, the training time and experience required to learn under this setup would be unaffordable. The alternative to this vanilla DR approach is to choose which attributes are sampled. This partially-guided DR saves computational resources, but might introduce some bias into the model. Besides, it would not be possible to ensure that the real environment is a sample from the training distribution. Hence, before deciding which characteristics should be randomized, it may be interesting to analyze the most relevant features to balance the trade-off between the number of randomized parameters and the feasibility of the implementation.

Within this idea of bounding the randomized features and their possible values, [RPF19] and [Mur+21] propose a Bayesian approach to select the parameters instead of using a uniform distribution that might result in sub-optimal outcomes. [Meh+20] implements an active DR that looks for the most relevant environment variations by leveraging the discrepancies between the current policy outcomes in the randomized scene and a baseline instance. Conversely, [Pra+19] suggests using a structured DR alternative that uses scene information to generate synthetic data more similar to the real scenario, thus avoiding unfeasible or unrealistic situations. In this line, [Yue+19] and [Che+19] advocate for the inclusion of knowledge about the real setup to adapt randomized images during the virtual training. While [Yue+19] utilizes real images processed with an image segmentation model to create an initial dataset to which DR is applied, [Che+19] adapts the randomization in simulation according to the results obtained after a few roll-outs in the real environment. Finally. [Du+21] auto-tunes the simulator parameters to match the real-world images, converting it into a search problem. Their Search Param Model (SPM) takes a sequence of observations and actions, a set of parameters, and predicts if their value should be higher or lower to match the current observations. They perform sim-to-sim and sim-to-real experiments showing an improvement over the vanilla DR.

2.3.2. Domain Adaptation

DA [WD18] approaches translate data from a source domain to a target domain. Whereas DR seeks to train the virtual agent across a wide distribution of environments with randomized features, expecting that the target domain will fall into one of them, DA tries to unify data from both domains, adapting or completing the source experience to make it similar to the one from the target environment. Figure 2.5 depicts the difference between both techniques.



(a) (b)

Figure 2.5. Comparison between Domain Randomization (a) and Domain Adaptation (b), illustrating the transfer from the source domain (simulated environment) to the target domain (real environment).

Some DA techniques tackle the fusion by searching a common latent space between domains that translates experience from one domain to another. [SA20] addresses transfer learning through DA in continuous RL problems by learning source domain skills at a high level, then merges this data with the target data to help the agent during the discovery of the state-action mapping between domains and finally, transfers the acquired knowledge along the map to speed up training in other tasks or setups. [Hig+17] implements a multi-stage RL agent, DARLA, that learns a disentangled representation of the observed environment, the vision task, and then acquires policies, the acting part, in the source domain that can be directly used in the target domain. [Xin+21] presents a two-stage procedure where they first learn a Latent Unified State Representation (LURS) that is consistent across domains and then combine it with the RL policy learned in a source domain. Within this line, [Li+22a] also proposes a two-stage model where DA is implemented to align the domain-invariant state representations in a latent space across domains. These state representations are then used during the DRL agent training. [Che+21] suggests a cross-modal DA approach with sequential structure (CODAS) that focuses on learning a mapping function from low-dimensional states in the source domain to high-quality images in the target domain. [Jeo+20] develops a self-supervised sim-to-real method that also leverages the temporal nature of the DRL experience, as [Che+21], to adapt a latent representation learned in simulation to the real domain.

Other DA methods propose an image-to-image translation between domains [Mur+18; Pan+22b]. To perform this visual adaptation, Generative Adversarial Networks (GANs) [Goo+14] are widely used. GANs are a powerful framework for generative modeling that integrates deep learning methods. The adversarial training between the two networks, i.e., the generator and the discriminator, aims to generate synthetic samples that are indistinguishable from the original. The *generator* (G) has to fool the discriminator by generating new samples similar to the real data. It takes a random noise vector z from a prior distribution and produces a sample $G(z)$ that should resemble samples from the target domain. The *discriminator* (D) has to classify if the input data comes from the real dataset or the generator distribution. The training of both networks is simultaneous in a minimax game where D tries to maximize the objective function while G tries to minimize it. The typical loss function is

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.17}$$

where the first term of the sum represents the expected value of the log-probability that D correctly classifies the real sample x over the real data distribution $p_{\text{data}}(x)$, and the second term

is the expected value of the log-probability that D correctly classifies the generated samples of $G(z)$ over the noise distribution $p_z(z)$.

Several adaptions have been developed to overcome some of the limitations of GANs. In the image-to-image translation field, Cycle-Consistent Generative Adversarial Networks (CycleGANs) [Zhu+17] are designed specifically for problems with unpaired datasets. It uses two GANs, one per domain, to learn the bidirectional transformation (from the source to the target domain and vice-versa). Each of these GANs has its adversarial loss similar to Equation (2.17). CycleGAN introduces the cycle consistency loss to ensure that an image translated to the target domain and back to the source domain remains the same. This loss is defined as:

$$\begin{aligned} \mathcal{L}_{\text{cyc total}} &= \mathcal{L}_{\text{cyc}}(G, F) + \mathcal{L}_{\text{cyc}}(F, G) \\ &= \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1] \end{aligned} \tag{2.18}$$

where G is the generator that maps from domain X to domain Y, $G : X \to Y$, and F maps from Y to X, $F : Y \to X$.

Regarding the robotics grasping task, [Jam+19] proposes the Randomized-to-Canonical Adaptation Networks (RCANs) to bridge the reality gap. They translate randomized simulated images into their non-randomized canonical versions, allowing real-world images to be translated into the same canonical simulated format. Hence, training the agent in simulation with the canonical observations allows its zero-shot transfer with 70 % accuracy and 91 % accuracy in a few-shot transfer if the agent is fine-tuned with 5,000 real-world grasps. [Ho+21] presents a GAN-based method, RetinaGAN, that preserves object structure and texture consistency in the adapted images. They utilize a pre-trained object detector, EfficientDet [TPL20], that provides the object consistency loss added to the CycleGAN. The RetinaGAN is trained using simulated and 135,000 real-world samples and requires fine-tuning the camera pose. Overall, they improve the state-of-the-art grasping, pushing, and door-opening results. [Bou+18] solves the sim-to-real transfer by combining DR and a pixel-level DA technique named GraspGAN that modifies synthetic images generated with ShapeNet [Cha+15] to make them more realistic. GraspGAN implements various loss functions, like content similarity loss and task loss, to maintain the semantic information of the observation. During training they used simulated samples and real-world grasping data, achieving almost 77 % success rate in grasping real objects. Following the research line of [Bou+18], [Rao+20] introduces RL-CycleGAN, which incorporates an RL-scene consistency loss that ensures the retention of task-relevant features. It tailors the image translation to the needs of the RL task. It uses off-policy real-world data, simulated data, and the CycleGAN-adapted images during training. The outcomes show 95 % accuracy in single-bin grasping. Nonetheless, the method does not handle the physics-based discrepancies between domains, requiring some fine-tuning with real-world data.

Moving into robotic vision-based assembly tasks, [Yua+22] and their latest work [Shi+23], solve the sim-to-real problem using CycleGANs and force control adaption. In simulation, the action space is the robot position along the x, y, and z axes. The orientation of the end-effector is fixed. They experiment using three observation spaces: 64x64 RGB images, 64x64 greyscale images, and a 128x1 latent space generated by an encoder. The simulated outcomes show 0 % accuracy with the image observations and 96 % accuracy when the model input is the latent space. The policy trained with the latent space is transferred to the real setup using the CycleGAN and a force control adaptation that multiplies the original position action by a gain K

and uses the product C_{real} as the control command for the real robot controller. The success rate in the real world is 86 %. Besides, [Che+22] presents a Real2Sim policy adaption with minimal infrastructure. They propose a sim2real and real2sim pipeline enhanced with a CycleGAN and DR, mixing learning in different sub-tasks. During inference, the physical robot acts as if it operates in the simulated environment using the translated observations from the CycleGAN. The success rate achieved is 92.3 %. However, this solution heavily depends on the fixed camera pose. There is an important feature loss during domain transfer, and the current observation network, a vanilla CNN, is not deep enough and cannot handle sequential problems.

In robot locomotion and navigation, [Zha+19] proposes an image-to-image translation approach for real-to-sim domain adaptation that reverses the flowchart. Instead of going from simulation to reality, they translate real-world images back to the synthetic domain where it was trained during deployment "to make the robot feel at home". Their architecture combines the CycleGAN losses, a semantic loss, and a shift loss that constrains the consistency between subsequent frames. [Jia+21] introduces SimGAN, an adversarial RL framework that learns to identify a hybrid physics simulator that matches simulated trajectories to target policies using a learned discriminative loss. This hybrid simulator combines neural networks and traditional physics simulation and can be used to refine target policies. Finally, [Jan+24] proposes a two-level sim-to-real approach using a modified CycleGAN that translates simulated depth images to real ones and has specific loss functions. The simulated and CycleGAN images are encoded into a shared latent space that maps domain-invariant and task-invariant features. They achieve 74 % success in navigation during the simulation and 90 % rate in the real scenario, although they struggle with inference time and sample-time delays in the control.

While many domain adaptation techniques successfully map experience between domains by learning a common latent space or translating images, a significant drawback is that this process often introduces added complexity, requiring the design and training of additional models to align domain representations, which can be computationally expensive and time-consuming, particularly in tasks with high-dimensional state spaces. Hence, using DA may be justified in some cases but not in others.

2.3.3. Knowledge Distillation

Knowledge or policy distillation is a procedure in which a "teacher" model transfers its learned knowledge to a "student" network, which is typically smaller and lighter than the teacher [Rus+16a]. In the sim-to-real problem, the teacher is trained in the virtual environment, and the student learns to replicate the teacher's behavior by mimicking its policy actions in the real setup. However, the student should be able to refine the teacher's decisions according to the new environment. Therefore, it is trained using the teacher's output probabilities, i.e., soft targets, as a guide. Its loss function has to include a distillation loss term that measures the discrepancy between the student and teacher outputs. This loss is usually defined as the Kullback-Leibler (KL) divergence [KL51] between the teacher's action distribution $\pi_T(a|s)$ and the student's action distribution $\pi_S(a|s)$:

$$\mathcal{L}_{\text{distill}} = \mathbb{E}_{s\sim\mathcal{S}}[D_{\text{KL}}(\pi_T(a|s)\|\pi_S(a|s))] \tag{2.19}$$

where $\mathbb{E}_{s\sim\mathcal{S}}$ is the expectation over states s sampled from S.

Therefore, knowledge distillation falls into the few-shot transfer techniques since, in most cases, the student must adjust its weights with little experience from the target domain.

[Cza+19] researches different ways to improve learning, varying the distillation technique depending on the task.

[Tra+19] works on a continual learning approach applying policy distillation to robot navigation. The agent has to learn two tasks in simulation and then merge its knowledge into a single policy that will be deployed in the real scenario. Their approach uses learned features instead of raw images as input. [Kad+23] proposes a Cyclic Policy Distillation (CPD) method that addresses the DR issue of the number of randomized parameters by generating several sub-domains with local policies. These policies transition between sub-domains during the learning process, and, in the end, they are distilled into a global policy that will be transferred to reality. They achieve better sample efficiency and performance outcomes than other state-of-the-art approaches in the robot ball-dispersal task.

2.3.4. Progressive Neural Networks

Based on knowledge distillation, [Rus+16b] propose PNNs. PNNs transfer the knowledge between the teacher and the student networks as learned representations. Each agent is a (deep) neural network model, also known as columns, solving a particular MDP. The knowledge transfer between agents takes place through learnable lateral connections that link the columns' layers. Figure 2.6 shows the general architecture of PNNs.

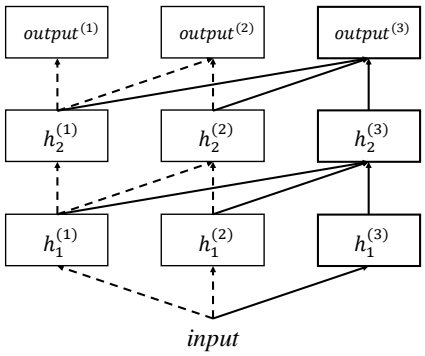


Figure 2.6. PNN general architecture. Each column refers to a DRL agent trained on a task or environment. They transfer the knowledge to the next columns through the learnable lateral connections. The dashed lines represent the agents and connections that have already been trained. In this case, columns 1 and 2 are sharing their representations with column 3 during its training.

PNNs are framed in the continual learning field [Par+19; CL18], which involves mastering a task and being able to learn other tasks by reusing the previously acquired knowledge. We separate them from knowledge distillation because instead of distilling knowledge, PNNs make an "ensemble" and reuse it. The learning mechanism consists in training an agent to solve a particular task and then training a second agent in a related task using a new lighter network. During the training process of the second agent, the learnable parameters are the weights of the second network and the weights of the lateral connections. Conversely, the weights of the first network remain frozen. This approach helps address catastrophic forgetting, i.e., the tendency of an agent to lose previously learned knowledge or skills when it learns new tasks or improves existing knowledge, by preserving the knowledge encoded in the first column. However, this does not entirely prevent the second agent from forgetting some of the previous knowledge learned.

Within the robot manipulation control field, [Rus+17] applies PNNs to the sim-to-real problem. With a robotic arm, they reach a target within a 10 cm tolerance using a 64x64 RGB

observation. They successfully bridge the reality gap in a few-shot attempt, needing just 60,000 steps in the real world to train the student column. More recently, [Luo+24] adapts the original PNNs architecture to learn the state transition distribution in a model-based approach using the orientation of the joints instead of images as observations.

2.3.5. Semantic Knowledge

Semantic knowledge or object-centric approaches [Dev+18] aim to simplify the state representation of complex environments by enabling abstract reasoning from low-level perceptual features that capture their compositional properties. Integrating information about the scenario elements as an extra input can be regarded as a way of giving some "common sense" capability to the agent.

[Dev+18] proposes a framework that leverages object-centric priors from pre-trained visual models and a two-stage attention mechanism, one pre-trained task-independent and the other task-specific, to enable robotic skill learning. The task-specific attention is fine-tuned with demonstrations, and the robot's control policy is trained using RL with the salient object features as input. It successfully generalizes policies to different objects and cluttered environments, demonstrating a robust performance despite the presence of distractors. However, the visual representation is limited to image-space object coordinates, which may not be sufficient for tasks requiring a detailed understanding of object configurations. [Loc+20] presents the Slot Attention method, able to interface with perceptual representations and build a set of abstract task-dependent representations. These sets are the slots that gather object-centric representations and achieve generalization in unseen compositions. [Kan+17] proposes the Schema Networks, an object-oriented generative physics simulator that identifies and separates different causes of events. It can also reason backward from a desired outcome to determine the necessary steps to achieve that goal. They compare Schema Networks with PNNs in Atari environments [Tow+23], obtaining slightly better results, with the difference that [Kan+17] also performs zero-shot transfer. [DL20] integrates into the Rainbow model [Hes+18] simple feature-engineered object representations, showing that there is a high impact on the agent's performance and identifying that some Atari environment states are more sensitive to these representations than others.

Additionally, Large Language Models (LLMs) have been proven capable of providing this semantic knowledge to DRL agents. [Pte+24] defines three study classes based on the interaction between the two models. One of them, the *LLM4RL* studies, gathers those that "use an LLM to supplement the training of an RL model that performs a general task that is not inherently related to natural language". LLMs may improve the efficiency of the agents' training process. For example, they can facilitate exploration [QSK23; Du+23] or enabling policy transfer [RYG22]. Concerning the grasping task, [Li+24] demonstrates that generating human grasp trajectories needs not only information about the object geometry but also semantic information. Their semantic-based grasp generation method, SemGrasp, generates static human grasp poses with semantic information in the representation.

Unfortunately, integrating a complete LLM in a DRL framework might be excessive if the only contribution expected is the semantic information. To overcome this, semantic knowledge can be stored in graph embeddings and manipulated using Knowledge Graph Embedding (KGE) techniques. [Mia+23] shows that building a semantic representation framework based on a knowledge graph is a suitable method for gathering robotic manipulation data from different

sources and can guide planning as well as generate semantic representations of entities and relations in the knowledge base.

2.3.6. Other approaches

Other solutions bridge the reality gap using Imitation Learning [Hus+17] by leveraging samples from a human expert or other policies. [Nai+18] employs behavior cloning to reduce the agent exploration phase in environments with sparse rewards. [Pen+18a] applies Inverse Reinforcement Learning [NR00; AD21] to learn motion across different assets and scenarios. [Zhu+18] uses Generative Adversarial Imitation Learning (GAIL) [HE16], which merges imitation learning and GANs, to successfully train end-to-end visuomotor policies in a virtual robotic arm and transfers them to a real robot. [Lee+21] uses imitation learning through vision-based interactive policy distillation to teach stacking policies that are then transferred from simulation to reality. The primary drawback of Imitation Learning is the need for an expert to gather demonstrations or policies because, depending on the problem, task, or setup, it might not be available.

Finally, Meta Reinforcement Learning (Meta-RL) [Wan+16a] emerged from the success of recurrent neural networks in a supervised learning context and their application to meta-learning. It focuses on training agents to learn how to learn. Instead of learning a single policy on a single task, a Meta-RL agent learns a meta-policy that can adapt quickly to new tasks by leveraging previous experiences and obtaining great generalization capabilities. For example, [Arn+20] exploits this "learning to learn" idea combined with DA to train a policy under changing dynamic conditions in a virtual environment and then test it on the real scenario. Meta-RL algorithms tend to be much more complex than classical DRL methods. Therefore, depending on the application and task solved, it might not be the most suitable option to implement.

2.3.7. Concluding remarks and selection of techniques

Figure 2.7 presents the sim-to-real transfer techniques this literature review highlights. Domain Randomization (DR) is one of the most popular approaches due to its simplicity and applicability. It works by training agents in a range of randomized virtual environments to ensure that they can generalize well when deployed in real-world scenarios. However, the key disadvantage of DR is that it requires extensive variability in simulations, which makes it computationally expensive, and still, there is no guarantee that the real environment will match any of the virtual configurations, limiting its effectiveness in highly constrained industrial settings.

Domain Adaptation (DA) techniques attempt to map experiences between domains by learning a common latent space or by translating features, offering a more targeted approach to aligning the virtual and real domains. DA methods often involve the additional complexity of designing and training extra models, which introduces significant time and computational costs. This complexity should be considered in industrial applications where strict performance requirements and hardware limitations are common.

Knowledge distillation is commonly used to adapt policies from a "teacher" trained in a simulated environment to a "student" deployed in a real-world scenario. The student learns by replicating the teacher's actions, guided by a loss function that minimizes the difference between their respective output distributions.

Progressive Neural Network (PNN), based on knowledge distillation, is another approach to transferring learned knowledge from one domain to another through lateral connections between network layers. While PNNs can facilitate efficient transfer between tasks and domains, the primary drawback is their few-shot approach which involves gathering some experience in the real environment to fine-tune the student's behavior.

Semantic knowledge aims to simplify complex environments by abstracting relevant information from perceptual features, allowing agents to focus on essential elements. While these methods enhance generalization, they can be limited by simplistic object representations, often lacking the detail required for tasks that involve intricate object configurations. Moreover, scaling these approaches to dynamic environments remains a challenge, despite advances like knowledge graph embeddings and pre-trained models.

Other techniques, such as Imitation Learning and Meta-Reinforcement Learning (Meta-RL), have also shown promising results in transferring high-level skills and generalizing across tasks. Imitation Learning can guide agents in learning optimal behaviors more quickly by leveraging demonstrations from experts or previously trained policies. However, this dependency on expert demonstrations can be limiting. Meta-RL focuses on preparing agents to adapt quickly to new tasks with minimal real-world data but entails a huge algorithmic challenge that might be a disproportionately complex solution to address relatively simple problems.

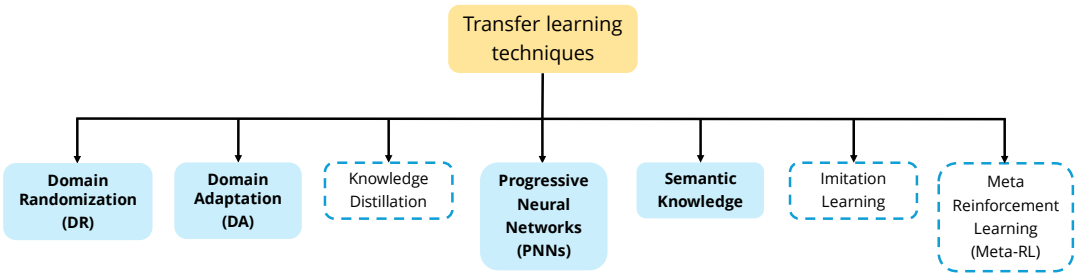


Figure 2.7. Transfer learning techniques. Those analyzed to address the sim-to-real problem in this dissertation appear with the blue background.

In conclusion, while these methods demonstrate effective solutions for sim-to-real transfer in controlled or research-based environments, they often fail to address the specific constraints of industrial settings. Developing methods without relying on highly accurate simulations, real-time adaptation, and minimal computational overhead remains unresolved in the literature. Most solutions involve complex models and additional training steps in the real setup that increase both design effort and time costs, making them less practical for deployment in real industrial scenarios, where reliability, efficiency, and simplicity are crucial. This highlights the necessity for transfer learning methodologies that can align with the stringent requirements of industrial applications, ensuring a balance between computational efficiency and transfer effectiveness.

3

Problem Description

Understanding is a two-way street.
Eleanor Roosvelt (1884–1962)

This chapter outlines the task definition and the experimental frameworks for the research, detailing the setups for two different robotic arms and describing each asset and virtual and real environment configuration.

3.1. Task definition

The task addressed to explore the effectiveness of the different transfer learning techniques is object approaching, the first part of one of the most common operations in industrial environments, such as pick-and-place. More specifically, we solve the approaching step for an object placed arbitrarily within the workspace of a robotic arm. The task is modeled as an episodic MDP in which, the initial position of the robotic manipulator and the target are randomly defined. The robot is considered to have reached the target if the distance between the arm gripper and the center of the object, a 6 cm-side cube, is less than 5 cm during training and 10 cm during evaluation. We argue that training in more challenging conditions than those actually required leads to better performance. This is commonly known as *overlearning*. Each episode lasts 50 steps maximum and finishes earlier if the end-effector reaches the goal within the rewarding distance.

Even if it might seem like a simple task already solved by classical control algorithms, we introduce two elements that considerably increase the complexity. First, the random placing of the target within the robot's workspace considers the possibility that the industrial environment is not fixed, forcing the agent to acquire generalization capabilities. Second, the input to the DRL agent is a 64x64 RGB image from an externally mounted camera without additional proprioceptive information. The agent has to learn to infer the robot and the target pose in a 3D world using 2D observations. This decision is based on the fact that we are looking for a sim-to-real solution that can be applied to various manipulators and even different industrial problems. Therefore, using a low-quality image from an externally mounted camera without depending on the robot or the scenario characteristics makes our algorithmic proposal, on the one hand, easier to implement in real industrial environments and, on the other hand, adaptable to different tasks.

3.2. IRB120 experimental setup

3.2.1. IRB120 description

The asset used to develop the pipeline capable of transferring DRL experience between a virtual and a real agent is the IRB120 industrial robotic arm from ABB [ABB19]. The IRB120 is compact and lightweight, making it easy to integrate into small spaces. It is compatible with ABB's IRC5 controller and is programmed in RAPID, a high-level language designed explicitly for controlling industrial robots manufactured by ABB. The IRB120 supports various communication interfaces for seamless integration into existing automation systems.

This 6 degree-of-freedom (DoF) manipulator offers a repeatability of ± 0.01 mm, ensuring precise and consistent operations. It can handle a payload of up to 3 kg, with a reach of 580 mm, and can operate at high speeds performance with optimized acceleration and deceleration. Figure 3.1 shows the IRB120 joints rotation axes and Table 3.1 displays their range and velocity.

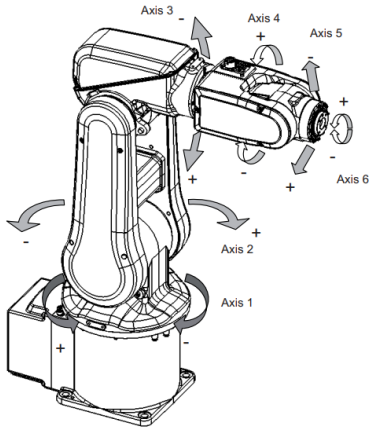


Figure 3.1. IRB120 axes and their rotation (*Source: ABB [ABB19]*).

Table 3.1. IRB120 joints' working range and velocity.

Axis motion	Working range	Velocity
Axis 1 (rotation)	-165 to +165°	±250 °/s
Axis 2 (arm)	-110 to +110°	±250 °/s
Axis 3 (arm)	-110 to +70°	±250 °/s
Axis 4 (wrist)	-160 to +160°	±320 °/s
Axis 5 (bend)	-120 to +120°	±320 °/s
Axis 6 (turn)	-400 to +400°	±420 °/s

According to its specifications, the IRB120 workspace is defined by the area of a semicircular annulus, with its center at the robot's base. The inner and outer circumferences have radii of 25 cm and 55 cm, respectively. Outside of this area, it is highly likely that the robot will reach a singular point, i.e., a configuration from which the inverse kinematics cannot be computed, thus preventing the robot's movement.

3.2.2. IRB120 virtual environment

As presented in Section 2.2, MuJoCo is the software selected to train the DRL agent in a virtual environment due to its physics simulator, rendering capabilities, open-source license, and widespread use in the research community.

The scenario virtualization is performed by designing a MuJoCo .xml file. This file has a hierarchical structure with concatenated elements that can inherit properties from their parents. These are its most relevant sections:

- **Compiler and Option**: set the main simulator and physics engine parameters such as the angle units, the total mass of the asset, the timestep of the simulator, the directory of the meshes, and if the inertia of the bodies should be inferred from the described geometry or it is explicitly provided, among others.

- **Asset**: the elements within this tag define the external resources, like the meshes and textures used in the simulation. The IRB120 virtual model is defined based on the .stl files available for each joint and component.
- **Worldbody**: contains all the descriptions of all the physical objects of the simulation and their properties, as well as the rendering parameters like the camera position and the scene light. Physical objects are defined as bodies, geometries, joints, and sites. Each has its own customizable parameters, like color, position, orientation, axis angle, size, range, and mesh file. We first describe the camera and light settings, then the ground plane and the target object, a 6 cm cube, and finally, the IRB120 joints and how they are linked. This XML section is the core of the MuJoCo arena description and is critical for a correct simulation behavior.
- **Actuator**: describes the control type (e.g., motor torque, position, velocity, etc.), which joints are controlled, and their control range.

The IRB120 environment prepared has a background with a green ground plane and a grey sky. The IRB120 is placed in the middle of the scene with the base on the ground. The 6 cm red cube target is positioned on the ground within the IRB120 reach. Its coordinates are within the intervals [0.2, 0.4] m for the x-axis and [–0.3, 0.3] m for the y-axis (Figure 3.2). The camera pose is set in the middle of the scene, with an orientation of 180° around the z-axis, and a 30° inclination with respect to the horizontal plane around the y-axis to ensure the complete scene is captured (Figure 3.3). This placement gives a viewpoint with a perspective in which the resulting 2D image includes visual depth cues, enabling better interpretation of the spatial arrangement of objects within the scene. We argue that, although this setup considerably differs from an industrial environment, it is appropriate to focus on the pipeline definition for the sim-to-real transfer without adding extra complexity and distractors to the main objective. Besides, the design is inspired and aligned with state-of-the-art works like [Rus+17]. Figure 3.4 shows the IRB120 scenario with high resolution and the 64x64 pixel observation fed to the agent. Table 3.2 gathers the RGB codes of the main environment elements.

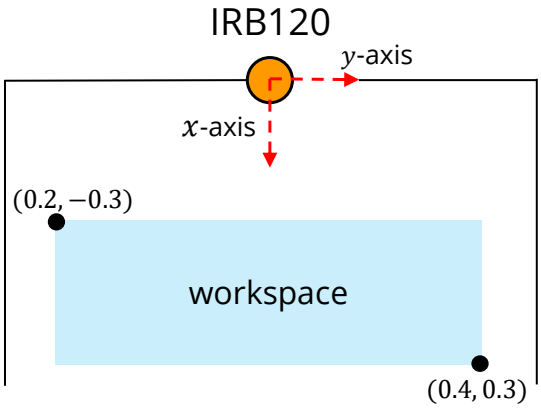
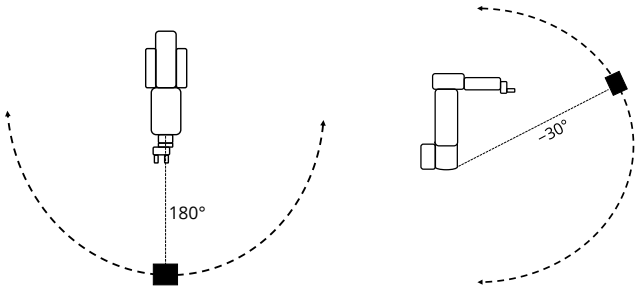
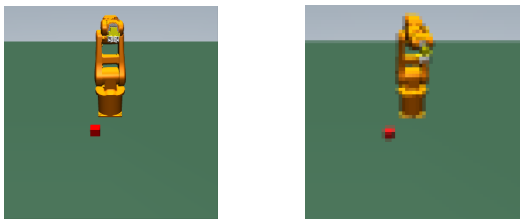


Figure 3.2. IRB120 workspace and axes location. The target coordinates should be within the intervals [0.2, 0.4] m for the x-axis and [–0.3, 0.3] m for the y-axis.



(a) (b)

Figure 3.3. Camera pose around the z-axis (a) and camera pose around the y-axis (b) in the IRB120 environment.



(a) (b)

Figure 3.4. IRB120 virtual observations with 640x640 pixel resolution (a) and 64x64 pixel resolution (b), which is the one received by the agent.

Table 3.2. IRB120 baseline environment colors. The RGB code is provided to supplement the color name.

Background	Robot	Gripper	Target
Green (ground) (0.19, 0.30, 0.23)	Orange (1.0, 0.5, 0.0)	Yellow (connector) (1.0, 1.0, 0.0)	Red (1.0, 0.0, 0.0)
Gray (sky) (0.67, 0.71, 0.75)	Black (joint 6) (0.0, 0.0, 0.0)	White (fingers) (1.0, 1.0, 1.0)	

3.2.3. IRB120 real environment

The real environment layout was built to replicate the virtual scenario. The IRB120 is positioned in the middle of the composition with the help of a metallic structure, and the floor and the background planes are made of green and grey sheets, respectively. The camera used is the Intel® RealSense™ Depth Camera D435[1]. It is equipped with a pair of depth sensors, an RGB sensor, and an infrared projector, although only the RGB information is used. The D435 camera offers an 85.2°x58° field of view and an RGB resolution of 1920x1080 at 30 fps. The camera and PC communication is done using our own drivers based on the `pyrealsense2` Python library[2]. This camera completely meets the requirements we need, has an affordable price (around 350€), and enables using the depth channel in future work.

The IRB120-computer connection, i.e., the agent-environment pair, is established via LAN and handled through a socket. While the communication methods on the computer are written in Python, in the robot the program has been developed in RAPID. When the communication flows from the agent to the environment, the functions developed involve requesting the current

[1] https://www.intelrealsense.com/depth-camera-d435/
[2] Available at https://github.com/IntelRealSense/librealsense

orientation of the robot's joints, retrieving the gripper's (x, y, z) coordinates relative to the robot base, and sending the updated joint orientations. Conversely, when the communication is from the robot to the agent, the functions include transmitting the joint angles, the gripper's coordinates, and applying the new joint orientations to update the robot's position. Figure 3.5 summarizes the real setup configuration and communication flow. The computer used in the real setup runs Ubuntu 20.04 LTS, Python 3.8, and PyTorch[3] (v1.13.1) [SAV20]. It is equipped with an Intel® Core™ i7-12700 processor at 2.10 GHz, 128 GB of DDR5 RAM, an NVIDIA GeForce RTX 3070 GPU, and a 1 TB M.2 SSD.

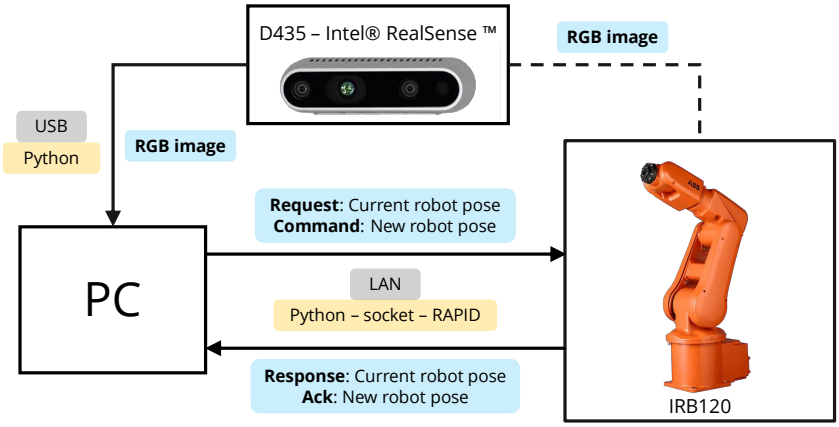
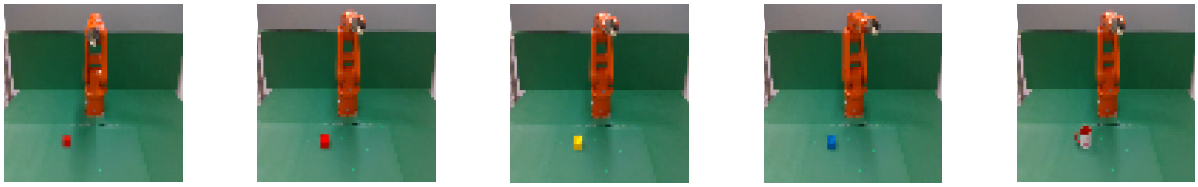


Figure 3.5. IRB120 real setup configuration diagram. The hardware components are connected via LAN and USB connections. The environment observation captured by the camera is sent directly to the computer. The bidirectional communication between the IRB120 controller and the computer is handled through sockets. While the computer can request the current pose or command the new pose, the IRB120 can respond with the current pose or acknowledge the reception and achievement of the new pose.

Before experimenting with real objects, Augmented Reality (AR) is used to display the targets, speeding up the real training process and easing deployment. Since the object can be placed anywhere in the workspace at the beginning of the episode, building another framework to fulfill this will add extra cycle time. Hence, we pre-saved offline different positions and, during training and evaluation, artificially rendered the target once the camera captured the image and before passing it to the agent. The sites' poses are gathered using ArUco markers [Gar+14], square and binary patterns with a unique identifier encoded that computer vision algorithms can easily detect and decode, and can provide reference axes that facilitate the placement of objects. With this approach, experiments are efficiently performed in the real environment. The markers are within the limits of the virtual target, [0.2, 0.4] m for the x-coordinate and [–0.3, 0.3] m for the y-coordinate. In total, 125 target positions are saved. Towards the end of the thesis, models are also evaluated with four real targets to corroborate the results obtained with AR. They are a red LEGO® cube, a yellow LEGO® cube, a blue LEGO® cube, and a red and white mug. Figure 3.6 displays the image with the augmented reality target and the real targets.

Despite the purpose and our efforts to replicate the virtual arena, the differences between both setups are apparent. Additionally, one has to consider the noise introduced by the camera, which is not considered in the virtual simulator. These discrepancies between images have been of great importance to assess the different transfer learning techniques, since the image is the only input for the agent.

[3] https://pytorch.org/



(a) (b) (c) (d) (e)

Figure 3.6. IRB120 64x64 pixel observations in the real setup with the AR target (a), the red (b), yellow (c), and blue (d) LEGO® cubes, and the white and red mug (e).

3.3. UR3e experimental setup

3.3.1. UR3e description

The UR3e from Universal Robots (UR) [Uni20] is used to measure and quantify the generalization capabilities of the sim-to-real methodology developed with the IRB120. The UR3e is a collaborative, versatile, lightweight robotic arm suitable for compact spaces. It is compatible with the Universal Robots control software and can be programmed using either Polyscope, the graphical interface, or URScript, the programming language tailored for controlling UR robots. As the IRB120, the UR3e supports various communication protocols and can be integrated with third-party accessories, making it highly adaptable to different automation environments.

The UR3e is also a 6 DoF arm. It has a repeatability of ± 0.03 mm, slightly higher than the IRB120. The UR3e can handle a payload of up to 3 kg, with a reach of 500 mm, 80 mm less than the IRB120. The UR3e morphology completely differs from the IRB120, as it has some offsets between the joints that increase the difficulty when understanding its movement. Its collaborative features include force sensing, enabling safe interaction with humans and the environment. Besides, the robot operates efficiently with smooth acceleration and deceleration, providing consistent performance across applications. Figure 3.7 depicts the UR3e joint locations, while Table 3.3 presents their range and velocity. In this case, the end-effector, a two-finger gripper, is directly mounted in the last joint without needing extra parts.

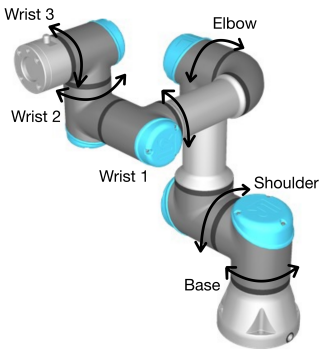


Figure 3.7. UR3e axes and their rotation

Table 3.3. UR3e joints' working range and velocity.

Axis motion	Working range	Velocity
Axis 1 (base)	$\pm 360°$	$\pm 180\,°/s$
Axis 2 (shoulder)	$\pm 360°$	$\pm 180\,°/s$
Axis 3 (elbow)	$\pm 360°$	$\pm 180\,°/s$
Axis 4 (wrist 1)	$\pm 360°$	$\pm 360\,°/s$
Axis 5 (wrist 2)	$\pm 360°$	$\pm 360\,°/s$
Axis 6 (wrist 3)	Unbounded (∞)	$\pm 360\,°/s$

The UR3e workspace is defined by the area of a semicircular annulus, with its center at the base of the robot. The inner and outer circumferences have radii of 10 cm and 55 cm, respectively, similar to the IRB120. Outside of this area, it is highly likely that the robot will reach a singular point. Although its joints offer greater freedom of movement compared to the IRB120, its movements have been slightly restricted due to the real scenario constraints.

3.3.2. UR3e virtual environment

To validate the pipeline designed with the IRB120, we reused the same virtual baseline environment defined in Section 3.2.2. The scene is composed of a green ground and a grey sky, with the UR3e placed in the center. The target is a 6 cm red cube whose coordinates range from [–0.5, 0.5] m in the x-axis and [–0.5, 0.0] m in the y-axis (Figure 3.8). Note that in the UR3e, the y-axis origin is at the robot base and extends to negative values as the target moves further away. The camera pose is the same as in the IRB120 (Figure 3.3) with the only difference that, due to the change in the reference system, the orientation in the z-axis is 0°. Figure 3.9 depicts the UR3e virtual arena and the 64x64 pixel observation the agent receives as the environment state. Table 3.4 defines the RGB colors of the main scenario components.

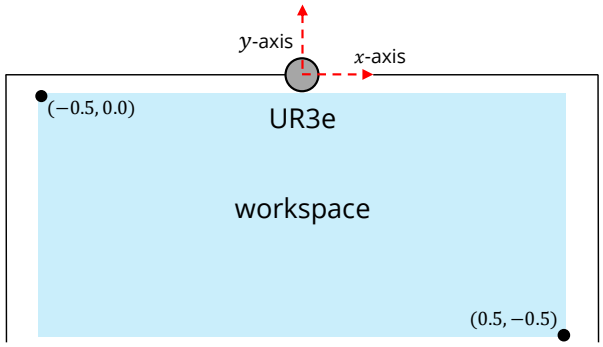
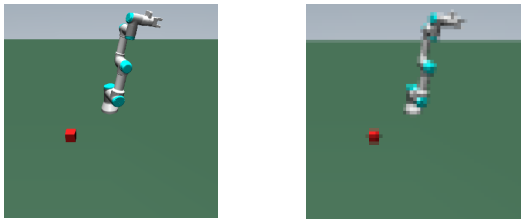


Figure 3.8. UR3e workspace and axes location. The target coordinates should be within the intervals [–0.5, 0.5] m for the x-axis and [–0.5, 0.0] m for the y-axis.



(a) (b)

Figure 3.9. UR3e virtual observations with 640x640 pixel resolution (a) and 64x64 pixel resolution (b), which is the one received by the agent.

Table 3.4. UR3e baseline environment colors. The RGB code is provided to supplement the color name.

Background	Robot	Gripper	Target
Green (ground) (0.19, 0.30, 0.23)	Gray (0.7, 0.7, 0.7)	White (fingers) (1.0, 1.0, 1.0)	Red (1.0, 0.0, 0.0)
Gray (sky) (0.67, 0.71, 0.75)	Blue (0.33, 0.67, 0.82)		

3.3.3. UR3e real environment

As in the IRB120, the real environment was designed according to the virtual setup. The camera used is also the Intel® RealSense™ Depth Camera D435[4], thus, the communication with the PC is

[4] https://www.intelrealsense.com/depth-camera-d435/

established through our drivers based in the `pyrealsense2` Python library[5]. The UR3e-computer attachment is configured like in the IRB120, with a LAN connection handled with the RTDE (Real-Time Data Exchange) protocol. The methods programmed on the computer to solve the communication are implemented in Python and are more complex than the ones developed for the IRB120 due to the number of signals and threads that must be controlled. The computer and the robot exchange data in a structured way using defined "recipes" in the configuration. These recipes specify the data fields that are sent back and forth between the computer and the robot. The recipe with the key `state` handles the transmission of robot state data from the robot to the computer. This includes variables like the robot's joint positions (`actual_q`), joint velocities (`actual_qd`), the Tool Center Point (TCP) speed, and pose, among others. The controller also sends register values. On the other hand, the recipe with the key `setp` manages the data sent from the computer to the robot. This allows the computer to control the robot by specifying target positions, velocities, or other commands. Lastly, the `watchdog` recipe ensures the safety and synchronization of the communication.

On the UR3e side, the programming was done using Polyscope instructions. Figure 3.10 shows the described configuration and the communication diagram. Regarding the instructions sent from the agent (i.e., the computer) to the environment (i.e., the UR3e), we reused the same methods as in the IRB120, except for the aforementioned protocol changes. The UR3e is connected to a computer running Ubuntu 20.04 LTS, Python 3.8, and PyTorch[6] (v1.13.1) [SAV20]. It is equipped with an Intel® Core™ i7-3770 CPU running at 3.40 GHz, and 8 GB of RAM.

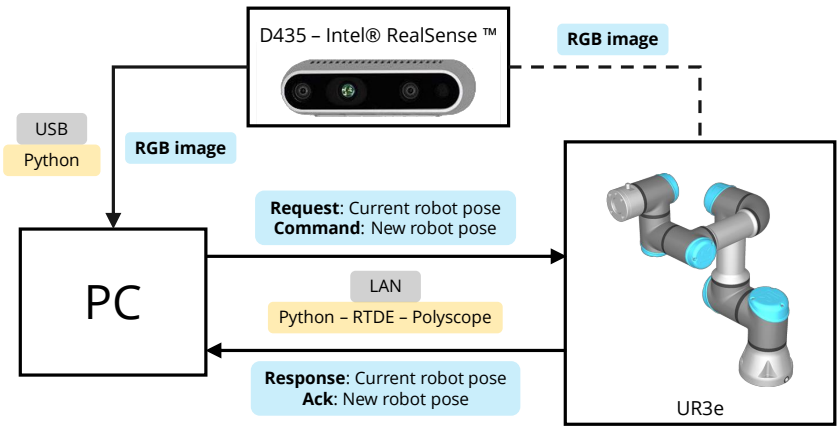


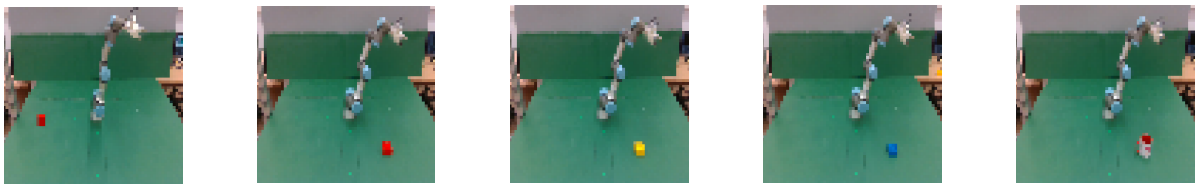
Figure 3.10. UR3e real setup configuration diagram. The hardware components are connected via LAN and USB connections. The environment observation captured by the camera is sent directly to the computer. The bidirectional communication between the UR3e controller and the computer is handled through RTDE. While the computer can request the current pose or command the new pose, the UR3e can respond with the current pose or acknowledge the reception and achievement of the new pose.

It should be noted that the configuration diagrams for both the IRB120 and the UR3e setups are very similar in their basic structure, a design choice made intentionally to ensure that the developed methodology would be as generalizable as possible. Both setups rely on a connection between a computer and the respective robot, utilizing USB and LAN for communication and incorporating an Intel® RealSense™ D435 camera. Commands and data are exchanged via Python, with the IRB120 using socket communication and RAPID, while the UR3e employs RTDE

[5]Available at https://github.com/IntelRealSense/librealsense
[6]https://pytorch.org/

and Polyscope. This equivalent configuration is key to ensuring that the same control principles can be seamlessly applied across both systems, facilitating a more universal approach.

Regarding the experiments, the approach is the same as in the IRB120. First, the agent is evaluated with AR targets whose poses are saved using the same ArUco markers as in the previous setup. In the UR3e, the limits are [–0.5, 0.5] m for the x-coordinate and [–0.5, 0.0] m for the y-coordinate, the same as in the virtual environment. In total, there are 400 target poses saved. This number is much larger than in the IRB120 because the scenario is placed more horizontally, and as a result, the camera can distinguish better the ArUco tags. The real targets used to validate the outcomes with the AR goals are also a red LEGO® cube, a yellow LEGO® cube, a blue LEGO® cube, and a red and white mug. Figure 3.11 displays the image with the augmented reality target and each of the real targets.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 3.11. UR3e 64x64 pixel observations in the real setup with the AR target (a), the red (b), yellow (c), and blue (d) LEGO® cubes, and the white and red mug (e).

4

Problem Modeling

Every great dream begins
with a dreamer.
Harriet Tubman (1822–1913)

This chapter puts forward the problem formalization by introducing the DRL algorithm used in all the experiments. After describing the algorithm's architecture, the MDP design process is outlined, presenting the action space and reward configurations tested. The chapter explains the training and post-training evaluation procedures used in the experiments and concludes with the MDP evaluation results and a discussion.

4.1. Asynchronous Advantage Actor-Critic

The Asynchronous Advantage Actor-Critic (A3C) [Mni+16] is one of the most suitable DRL algorithms due to the advantages described in Section 2.1.1.5, such as the variance reduction achieved through the use of error estimators and its efficient parallelization. This results in more stable learning and faster convergence compared to other methods. Since it operates with discrete action spaces, even though the robot joints can be commanded with continuous actions, we designed a discrete action space.

The A3C framework includes an *actor*, responsible for the optimization of the policy $\pi(s)$, and a *critic*, who estimates the state-value function $V(s)$. To measure the goodness of taking an action a in state s with respect to the goal, A3C uses the Advantage Function (4.1), which computes the difference between $Q(s,a)$, and $V(s)$.

$$A(s,a) = Q(s,a) - V(s) \tag{4.1}$$

The Advantage Function can be approximated using the TD error (4.2), which updates $V(s)$ by predicting the current state value and reward plus the discounted value of the next state.

$$A(s_t,a_t) \approx \delta_t = r_t + \gamma V(s_{t+1};\theta_1) - V(s_t;\theta_1) \tag{4.2}$$

where t is the current time step, δ_t is the TD error at time step t, r_t is the reward at time step t, γ is the discount factor, and $V(s_t;\theta_1)$ is the estimated value of state s_t given the parameters θ_1.

A3C uses the General Advantage Estimator (GAE) (4.3) to smooth the advantage estimation provided by the TD error, reduce the variance and stabilize learning. This smoothing is achieved by considering multiple TD errors, i.e., multiple future rewards and transitions.

$$A_{\text{GAE}_t} = \gamma\lambda A_{\text{GAE}_{t+1}} + \delta_t \tag{4.3}$$

where t is the current time step, A_{GAE_t} is the Generalized Advantage Estimator at time step t, γ is the discount factor, λ is the trace decay factor that controls the bias-variance trade-off, and δ_t is the TD error.

Regarding the loss that should be minimized, A3C defines the total loss as the sum of the *policy loss* plus the *value loss* (4.4).

$$\text{Loss} = \text{Policy}_{\text{loss}} + \text{Value}_{\text{loss}} \tag{4.4}$$

The policy loss (4.5) has two main components: the policy gradient, the first term in the summation, that adjusts the policy parameters θ_2 to maximize the expected return –ensuring that actions leading to higher advantages are more likely to be chosen in the future– and the entropy regularization, the second term, that encourages exploration by preventing the collapse to a completely deterministic policy.

$$\text{Policy}_{\text{loss}} = -\sum_i \left(\log \pi(a_{ti}|s_{ti};\theta_2) \cdot A_{\text{GAE}_t} - \beta H(\pi(s_t;\theta_2))\right) \tag{4.5}$$

$$H(\pi(s_t;\theta_2)) = -\sum_a \pi(a|s_t;\theta_2) \log \pi(a|s_t;\theta_2) \tag{4.6}$$

where i iterates over actions at time step t, $\pi(a_{ti}|s_{ti};\theta_2)$ is the probability of taking action a_{ti} given the policy parameters θ_2, A_{GAE_t} is the GAE at time step t, β is the entropy regularization weight, and $H(\pi(s_t;\theta_2))$ is the entropy of the policy (4.6).

The value loss (4.7) is responsible for training the critic, and is obtained as the least squares error between the n-step return G_i and $V(s)$.

$$\text{Value}_{\text{loss}} = \frac{1}{2}\sum_i \left(G_i - V(s_i;\theta_1)\right)^2 \tag{4.7}$$

$$G_i = \sum_{k=0}^{n-1} \gamma^k r_{i+k} + \gamma^n V(s_{i+n}) \tag{4.8}$$

where i iterates over time steps, $V(s_i;\theta)$ is the estimated value of state s_i given the parameters θ_1. G_i is the n-step return at time step i computed as an estimation of the total discounted reward from state s_i (4.8), where r_{i+k} is the reward received k steps after s_i. Note that in this case, n is the number of steps in the episode.

A3C differs from its non-asynchronous version, A2C, because it incorporates asynchronous updates. This means that multiple agents can learn in parallel on different environment instances. Each agent interacts with its environment copy locally, computes its gradients, and asynchronously updates a shared global network, ensuring an efficient and stable

learning process, reducing experience correlation, and, as a consequence, leading to better performance.

4.1.1. A3C architecture

The virtual agent architecture is deeply based on [Rus+17] since the task solved is the same and PNNs are one of the sim-to-real techniques analyzed and implemented in this thesis. However, the MDP definition, robots, and scenarios differ.

Figure 4.1 presents the model architecture. The 64x64 RGB image passes through two Convolutional Neural Networks (CNNs) [LBH15; GBC16; Li+22b] with 16 8×8 filters with stride 4 and 32 5×5 filters with stride 2, respectively. Their goal is to learn the features that allow for understanding the key aspects of the image. After each CNN layer, a non-linearity is introduced by applying the Rectified Linear Unit (ReLU) function, $f(x) = \max(0, x)$, to the outputs. The resulting tensor is flattened, and the hidden activations go through a Fully Connected (FC) layer.

After the FC, there is a Long-Short Term Memory (LSTM) [LBH15; GBC16] cell with 128 hidden states, whose role is to help capture the sequential nature of the problem by storing information from previous steps for later use. Therefore, it receives the new activations from the FC and has saved the state of the previous step. Finally, the output consists of six heads (FC layers) for the A3C actor, one per robot joint, plus one head, the A3C critic, that estimates $V(s)$. A softmax function is applied to translate the output activations into the probability assigned to each joint action, resulting in each joint policy π_n. Their output size depends on the number of actions defined in the MDP.

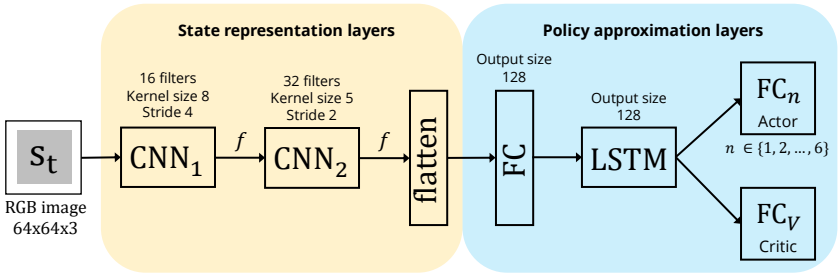


Figure 4.1. A3C architecture implemented. The input consists of a 64x64 pixel RGB image, processed through CNN layers with 16 and 32 filters, which learn the state representation. The activations pass through the FC and the LSTM for temporal state information before producing the policies for the robot joints, computed with a softmax after obtaining the FC activations, and the state-value estimation. f stands for the ReLU activation function.

4.2. MDP design

In a MDP setting, an MDP is formed by the set of states S, actions A, and rewards R. With the state observation already defined as the 64x64 pixel observation, we seek to determine an appropriate action set and reward function. Regarding the actions, we chose a discrete rather than a continuous space due to the algorithm selection, as the A3C works with discrete action spaces, the results obtained in several DRL works in the field of robotic manipulation and control [Rus+17] and because, according to several studies [KSH20; TA20; Pan+22a], it is shown that having continuous or very granular discrete spaces does not always lead to better results, apart from increasing computational complexity.

When defining the reward it is essential to bear in mind that it should not tell the agent *how to* perform the task, since this would introduce a prior bias that could rule out better solutions. The ideal reward configuration would be sparse, just a positive score delivered when the objective is met. The only feedback given during the rest of the steps would be either 0 or -1. However, this format does not always lead to successful learning. As the agent receives infrequent feedback, it is harder to discern which actions are promising, making the learning process inefficient. This delay between actions and rewards can significantly increase training times. The configurations proposed for training the various agents have been designed using deterministically selected values. These values provide the agent with feedback on its distance from the object at each step, with higher penalties for farther positions and compensatory rewards that offset accumulated negative values upon reaching the target. Some experiments were conducted with variable rewards based on the number of steps taken to reach the goal, aiming to incentivize the agent to find more efficient trajectories. The validation of the different positive values chosen is carried out experimentally.

Considering these insights, we defined and tested seven different action sets and reward function pairs. Table 4.1 gathers the MDP definitions for these models. The discrete actions are set according to the Maximum Position Increment (MPI) of each joint. This variable is obtained by multiplying the joint range with a 0.3 scale value if actions are applied to the joint torques and 0.015 if they command the joint angles.

4.2.1. Training and post-training evaluation procedure

Each of the MDPs in Table 4.1 corresponds to an agent trained and tested under the architecture explained in Section 4.1.1. As stated in Section 3.1, the rewarding distance during training was 5 cm and during post-training evaluation, 10 cm. At the beginning of each episode, the first and second joints of the robot and the target are set randomly sampling from a uniform distribution. For the robot, the distribution is defined as $U(+15\%$ WRLL, -15% WRUL), where WRLL and WRUL stand for Working Range Lower Limit and Working Range Upper Limit, respectively (i.e., the lower and upper limit for each joint angle). The target position in the XY-plane follows two uniform distributions where $x \sim U(0.2, 0.4)$ m and $y \sim U(-0,3, 0.3)$ m. Therefore, the target moves within a 20x60 cm area. In this set of experiments, the actions command the torque of the joints.

The training process lasts 70 M steps, and for every 50 k steps there is an interim evaluation of the last model shared in the A3C global network. This interim evaluation consists of 40 episodes with fixed initial robot and target poses so we can compare the outcomes between evaluations. The starting weights for the model are randomly chosen following an orthogonal initialization. The learning rate, which controls the step size during the optimization process, is set to $1e^{-4}$. The discount factor is 0.99, and the entropy regularization weight is 0.01. The optimizer used is RMSprop, with a decay factor of 0.99. Additionally, gradient clipping is applied with a maximum gradient norm set to 40 to avoid excessively large updates, further stabilizing the training process.

The virtual agents were trained using two computers to streamline the development of experiments, both running Ubuntu 20.04 LTS, Python 3.8, and PyTorch[1] (v1.13.1) [SAV20] as the deep learning framework, one with an Intel® RealSense™ i9-10900KF processor at 3.70 GHz, 64 GB of DDR4 RAM, an NVIDIA GeForce RTX 2080 Ti GPU, and a 2 TB M.2 SSD, and another

[1] https://pytorch.org/

Table 4.1. Definition of the MDPs tested. The chosen alternative is highlighted in gray.

Model	Action space[a]	Positive reward if target reached[b]	Negative reward per step[c]
M1	0 ±MPI ±MPI/2	+70	$-(2 \cdot dist)^2$
M2	0 ±MPI ±MPI/10 ±MPI/100	+70	$-(2 \cdot dist)^2$
M3	0 ±MPI ±MPI/10 ±MPI/100	+90 if ep_length ∈ [0, 30] +70 if ep_length ∈ (30, 35] +50 if ep_length ∈ (35, 40] +30 if ep_length ∈ (40, 50]	$-(2 \cdot dist)^2$
M4	0 ±MPI ±MPI/10 ±MPI/100	+100 if ep_length ∈ [0, 30] +50 if ep_length ∈ (30, 35] +30 if ep_length ∈ (35, 40] +20 if ep_length ∈ (40, 50]	$-(2 \cdot dist)^2$
M5	0 ±MPI ±MPI/2 ±MPI/4 ±MPI/16 ±MPI/64 ±MPI/128	+70	$-(2 \cdot dist)^2$
M6	if distance > 1.5*reward_dist ±(0, MPI, MPI/2, MPI/4) if distance < 1.5*reward_dist ±(0, MPI/10, MPI/50, MPI/100)	+70	$-(2 \cdot dist)^2$
M7	if distance > 1.3*reward_dist ±(0, MPI, MPI/2, MPI/4) if distance < 1.3*reward_dist ±(0, MPI/4, MPI/10, MPI/25)	+90 if ep_length ∈ [0, 30] +70 if ep_length ∈ (30, 35] +50 if ep_length ∈ (35, 40] +30 if ep_length ∈ (40, 50]	$-(2 \cdot dist)^2$

[a] MPI stands for Maximum Position Increment and reward_dist for rewarding distance.
[b] ep_length is an abbreviation for episode length.
[c] $dist$ is the distance between the gripper and the target.

with an Intel® Core™ i9-14900KF processor, 256 GB of DIMM RAM, two NVIDIA GeForce RTX 3090 GPUs, and a 4 TB WBC SSD.

Once training is over and the best model is selected, considering both the agent's performance and the time needed to achieve it, we carry out a post-training evaluation. It consists of 1,000 episodes with random initial robot and target poses and a different seed from the one used during training. This ensures a more reliable comparison between models without trusting solely the average return obtained during training. To compare the agents' performance in the post-training evaluation, we mainly use their accuracy, defined as the percentage of episodes in which the agent reaches the goal (4.9). We also consider other metrics like the episode length mean and standard deviation (std. dev.), the average return mean and std. dev., and the failure distance mean, std. dev., and maximum.

$$\text{Accuracy (\%)} = \frac{\sum_{i=1}^{N} \mathbb{I}(\text{dist}_i \leq 10\,\text{cm})}{N} \times 100 \tag{4.9}$$

where N is the total number of evaluated episodes, dist_i is the relative distance between the gripper and the target in the i^{th} episode, and $\mathbb{I}$ is the indicator function. Table 4.2 collects the hyperparameters used during training and post-training evaluation. The hyperparameter selection was guided by the recommendations provided in [Rus+17], despite the absence of specific values in their work, and was informed by established knowledge of typical values that have been shown to perform effectively in similar problem settings. A sensitivity analysis for hyperparameter optimization was not conducted, as the obtained results were deemed satisfactory and aligned with the objectives of this thesis. Future work may focus on exploring potentially optimal hyperparameter configurations to further enhance performance.

Table 4.2. A3C agent training hyperparameters.

Hyperparameter	Value
Seed	123 (training) 803 (post-training evaluation)
Training steps	70 M
Episode length	50 steps or target reached
Success distance	5 cm (training) 10 cm (post-training evaluation)
Evaluation interval	50 k steps
Evaluation episodes	40 episodes (training) 1,000 (post-training evaluation)
Discount factor (γ)	0.99
RMSProp learning rate	$1e^{-4}$
RMSprop decay	0.99
Entropy weight (β)	0.01
Trace decay factor (λ)	1

4.2.2. Results

M1 has the same action space as [Rus+17] but a different reward function. Instead of giving a +1 if the agent reaches the goal and staying there until the end of the episode, we provide +70, a high positive value, which is approximately the absolute value of the total return the agent would receive if the robot remains stays still in the initial position, and finish the episode. Moreover, we also reinforce the agent during the episode with a negative reward that depends quadratically on the relative distance between the gripper and the target. In the end, the positive reward compensates for the negative cumulative return received during the episode. With this approach, we aim to reduce the complexity and learning time associated with sparse rewards.

Model M2 has a logarithmic action set that allows the agent to approach the target faster when it is far away and operate smoothly in the surroundings of the goal. The accuracy obtained increased by almost 10 % with respect to the results of M1. With M3, we considered varying the positive reward based on the number of steps needed to reach the goal with the purpose of promoting a faster approach maneuver. The intervals are set according to the outcomes of M2 (e.g., in M2, the average episode length was 30.31, so in M3, the +70 reward corresponds to the interval (30, 35]). With this reward configuration, M3 presents a 12 % improvement with respect to M1. The last model with the logarithmic action space, M4, differs from M3 in the positive reward values, which try to encourage a reduction in the number of steps. The success rate drops almost 8 % with respect to M3. We suggest that this might happen because of the lower reward values for all the intervals except for the first one. The agent struggles to achieve

the goal in less than 30 episodes where the reward value was increased, and in consequence, the rewards obtained were lower than in M3.

M5 has the same reward configuration as M1 and M2 but with a binary logarithmic action set. Accuracy decreases considerably with respect to M2 and M3, and it only exceeds M1 by 3 %. The deterioration could be explained by the increased complexity of the action set, which might make finding the optimal policy difficult. In M6, the rewards remain as in M5, but two action sets were implemented depending on a constant threshold defined by the rewarding distance times 1.5. Therefore, if the gripper distance to the target is higher than 7.5 cm, the action set allows larger movements, so it can get close faster, while if the distance is lower, the agent has more granularity to act and approach the goal. The results were 7 % better than in M1, but around 2 % and 5 % worse than M2 and M3, respectively. Lastly, M7 proposes a similar action space as M6 but with a different distance threshold, 6.5 cm, and values for the second action set interval. This design builds upon the hypothesis that the action change in the M6 agent happened too far from the target and, as a consequence, the fine movements were not enough to reach the goal. Besides, we modify the reward configuration as in M3 where the scalar varies depending on the number of steps. The accuracy obtained is 6 % higher than in M1, but on average, 3 % lower than in the rest of the models evaluated. Please refer to Table B.1 to consult the details of the results for these models.

Regarding these results, we conducted a more exhaustive evaluation of models that performed better to unequivocally identify the best among them. For each model, we repeated the 1,000 episodes post-training evaluation with 10 different seeds and averaged those results. In addition, we assessed the accuracy with the 5 cm threshold, which was used during training and in the interim evaluations, and the 10 cm original post-training rewarding distance.

Table 4.3 collects the average outcome of the experiments. M2 and M3 stand out over the rest in the 5 cm evaluation. During the 10 cm test, M2 achieved 100 % success, while the others present failed episodes in one or more post-training evaluations. With these results, we also prove our hypothesis about the success rate improvement when training in more challenging conditions than the ones we have during the post-training evaluation.

Table 4.3. Average accuracy of the selected MDPs during the post-training evaluations. Models were evaluated across 1,000 episodes with 10 different seeds.

Model	Rewarding distance (cm)	Average accuracy (%)
M2	5	**95.7**
	10	**100**
M3	5	96.9
	10	99.7
M6	5	93.6
	10	99.6
M7	5	91.2
	10	99.9

In light of these results, and after comparing the success rates and training evolution, which are very similar between M2 and M3, M2 was chosen as the Baseline Model (BM) for the subsequent developments and experiments. The performance of the agent trained under this MDP, with a logarithmic action set and discontinuous reward function with negative reinforcement per step if the goal is not reached and a high positive otherwise, is on par with the state-of-the-art results, being slightly better than [Rus+17] in terms of the steps needed

to achieve the steady state. Figure 4.2 illustrates the BM average returns obtained during the interim evaluations. It reaches the steady state after 30 M steps. Around the 40 million and 60 million step marks, the agent appears to explore and exploit a sub-optimal policy, resulting in a temporary reduction in the average return. While model stability and convergence can only be guaranteed in tabular problems, the empirical results demonstrate no significant issues with stability during training and evaluation. This could be due to a well-known phenomenon in DRL, where internal dynamics in the target updates cause the agent to temporarily lose performance or a high learning rate value when the model is already stabilized.

After obtaining the BM with controlling the torque of the joints, the same MDP definition of the BM was tested with a joint orientation control and the MuJoCo position actuators. The actions, thus, are increments on the joints' angle. We found that the BM also masters the task with this control. Figure 4.3 training results show that the steady state is reached earlier than in the torque control (Figure 4.2), after 8 M steps rather than 30 M steps. This is a natural consequence of the fact that inferring how actions that command the joints' angle increment affect is much simpler than learning how actions on motor torque govern the robot's motion. In the post-training evaluation the agent obtained 99.6 % success on the 5 cm evaluation and 99.98 % on the 10 cm test.

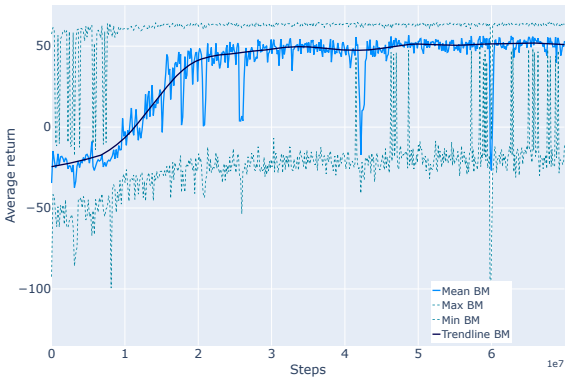


Figure 4.2. BM average returns obtained during the interim evaluations throughout the 70 M episode training with torque control. The steady state, with an average return of approximately 50, begins after 30 M steps (note the 1e7 scaling factor on the right-hand corner of the figure).

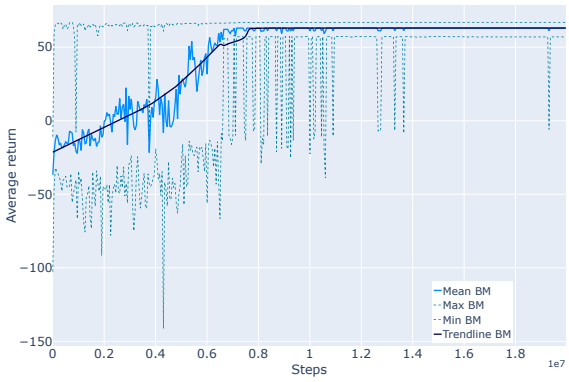


Figure 4.3. BM average returns obtained during the interim evaluations throughout the 20 M episode training with orientation control. The steady state, with an average return of approximately 65, begins after 8 M steps (note the 1e7 scaling factor on the right-hand corner of the figure).

4.2.3. Conclusion

In this chapter, the formalization of the problem has been explored, detailing the implementation of the A3C algorithm and the design of the MDP. The design and adjustment of the action space and reward configuration were crucial to improving the agent's performance. We tested various configurations of these components, aiming to ensure the agent could consistently learn. With different reward structures and action granularity, we observed significant variations in success rates across models. While the torque-based control achieved a stable state after 30 M steps, the joint orientation control reached its steady state much sooner, highlighting the differences in task complexity between the two control methods. Ultimately, M2, defined by a logarithmic action set and a negative reward per step plus a positive value if the target is reached, emerged as the best-performing model, achieving near-perfect accuracy in both 5 cm and 10 cm evaluations, thus becoming the BM for further experiments.

5

Domain Randomization

Either you fully commit to research,
or you don't commit at all.
Margarita Salas (1938–2019)

This chapter explores Domain Randomization (DR) and its application to the sim-to-real problem. First, it outlines the fundamentals of DR and continues with its objectives and contributions. Then, the set of experiments carried out following a high level DR strategy are presented. These tests consist in randomizing the environment's visual features, such as the camera pose, the target attributes, and the background. Part of the results presented in this section are published in the journal article "Learning more with the same effort: How randomization improves the robustness of a robotic deep reinforcement learning agent" [GBL22]. Furthermore, this chapter also addresses the application of low-level DR to the virtual observation with the aim of making it more similar to the real image. This is achieved by introducing Gaussian noise at a pixel level before the observation is processed by the agent.

5.1. Domain Randomization fundamentals

Domain Randomization (DR) [Tob+17] is a technique that aims to bridge the reality gap in sim-to-real problems by deliberately introducing a wide range of variability into the training virtual environments. The idea is to expose the virtual agent to a broad spectrum of scenarios during training so that the policy learned is robust enough to keep the same performance level and face the variations and mismatches of the real world. Therefore, the policy is trained on samples taken from a distribution of possible environments, learning to generalize across them. When the agent is deployed on the unseen real setup, it can perceive the scenario as another instance of the training distribution. As introduced in Section 2.3.1, several features can be parameterized, such as lighting conditions, textures, physical properties (e.g., mass, friction), or camera poses. In general, any feature of the agent's observation is a candidate for DR.

The advantages of DR are that it consistently improves the agent's generalization capability without increasing the architecture complexity or incurring in computational overheads. This

statement holds as long as the environment distribution is not too wide or the simulation has high rendering requirements. Regarding this last issue, the other advantage of DR is that there is no need for a precise virtual setup since the policy does not rely on any specific environment configuration but on its ability to handle variability and learn how to generalize.

5.2. Objective and contributions

In our search for an efficient pipeline to solve the sim-to-real problem in an industrial application, DR might not be an appropriate method to achieve a zero-shot transfer on its own due to the large visual discrepancies between real and virtual environments. However, combining DR with other techniques, such as PNNs, can potentially reduce the real experience needed to fine-tune the policy in a few-shot approach.

Therefore, we first investigate whether high-level DR, i.e., applying DR to visual features of the environment, enhances the agent's robustness and quantify its improvement in our problem and under our environment using a sim-to-sim approach. To better understand the drivers that lead to a decrease in robustness, the robotic agent is still tested in the virtual setup, following sim-to-sim approach to ensure total control over the divergences between the simulated and real models. Then, we study the effect of introducing low-level DR, i.e., applying DR at a pixel level, to enhance the sim-to-real transfer by making the RGB distributions of the virtual and real observations closer.

The main contributions of this chapter are:

- A methodology that allows the comparison with a benchmark to assess the improvement in the agents' performance.
- The quantification of the robustness gain in a first-column PNN-like agent trained with high-level DR. This outcome can be understood as an efficiency improvement in the sim-to-real process.
- The incorporation and quantification of low-level DR to enhance sim-to-real efficiency in PNN-like agents.

5.3. High-level Domain Randomization

5.3.1. Method

Figure 5.1 summarizes the methodology followed. On the top branch, we use the Baseline Model (BM), defined in Section 4.2.2, as reference. Once the BM agent is trained, a virtual test bench measures its robustness against changes in the environment settings. Finally, we proceed with the obtention of results and their analysis. Afterward, we employ the same method on the bottom branch, but in this case, with an agent whose training phase has been enriched by applying DR to certain environment features. We call this model the Domain Randomization Model (DRM).

5.3.1.1. Training and post-training evaluation procedures

The general training and post-training evaluation parameters and metrics remain as the ones explained in Section 4.2.1. Briefly, the initial robot and target poses are set according to two uniform distributions. The training lasts 70 M steps, and the interim evaluation is done every

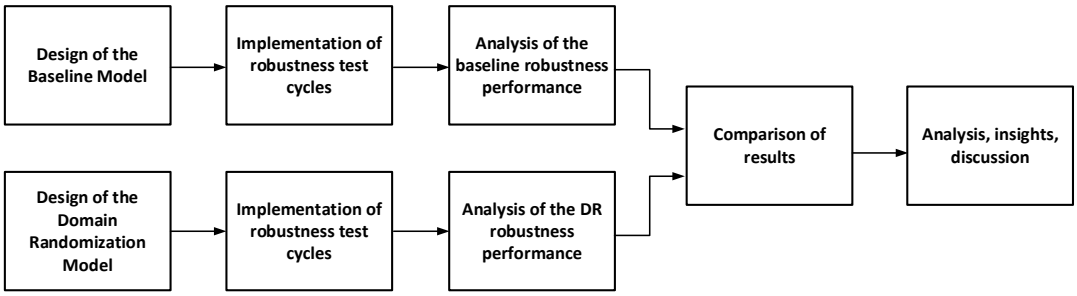


Figure 5.1. Methodology followed on the high-level DR experiments.

50 k steps for 40 episodes that have the same initial robot and target poses across them with 5 cm rewarding distance. The post-training evaluation procedure has a different seed, and it consists of 1,000 episodes with random initial robot and target configurations and a 10 cm rewarding distance.

5.3.2. Description of the experiments

We conducted several experiments to explore how the agent's performance can improve through the implementation of DR. The first set of experiments address visual changes in the observation. We determined the scenario elements and features suitable for randomization, e.g., the camera pose, the target, and the background, and how they could be modified. In addition, we considered environments in which variations took place in several features simultaneously. In such cases, the distribution is much more complex, but the resulting policy might exhibit better generalization capabilities. We did not consider the robot for DR because, even though it is the leading actor, its color representation is accurate enough, and modifying any morphological aspect might result in unfeasible solutions during the physics simulation. The experiments were performed using joint torque control on the IRB120. The brightness and texture conditions were fixed, with the robot casting a shadow on the floor due to the orientation of the light source. Figure 5.2 shows the summary of the environment distributions where agents were trained.

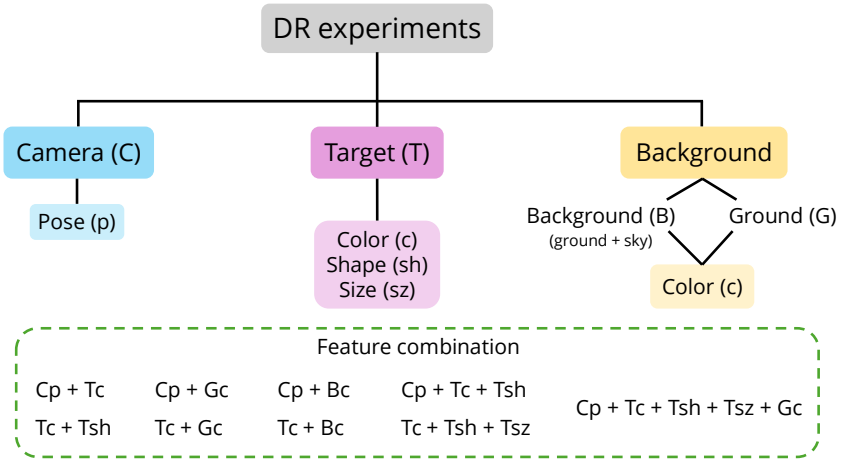


Figure 5.2. Overview of the high-level DR experiment configurations, including camera pose (Cp), target color (Tc), shape (Tsh), size (Tsz), background color (Bc), ground color (Gc), and their respective combinations for different experimental setups.

- **Camera Pose (Cp)**: The camera was set to a fixed position around the z and y axes of (180°, −30°), respectively, in the BM. When DR is applied during training, its pose

is randomly defined between episodes by a uniform distribution in the interval [160°, 200°] for the z-axis and [−40°, −20°] for the y-axis. Table 5.1 gathers the training and post-training evaluation conditions for the BM and this DRM_{Cp} when the camera pose is randomized.

For the post-training evaluation, we defined a virtual test bench that provides a highly interpretable measurement of the effectiveness of the virtual training stage and where we assess the policy robustness. To asses the generalization capability of the agent, the virtual test bench has a higher variability in the camera pose than the one used to train the DRM_{Cp}. The pose intervals during the evaluation were [140°, 220°] and [−50°, −10°] for the z and y axes, respectively, with 5° granularity. Figure 5.3 depicts the test bench grid and the post-training evaluation conditions, and Figure 5.4 shows the range of the camera positions with respect to the robot. The gray-shaded region represents the poses shown to the DRM_{Cp} agent during training while the BM was anchored in the center. The grey rectangle corresponds to only one-third of the evaluation area. This lets us investigate how the agent performs when it has to generalize its knowledge to other unseen scenarios. We evaluated 153,000 camera pose combinations, each with a random initial robot and target configurations.

According to the results presented in [Tob+17], the camera pose is the feature that most affects the agent's accuracy. For this reason, we performed an exhaustive analysis of the effect of the camera position on our setup. This work is reported in [GBL22]. Although we cannot directly compare our work in this field with [Tob+17] as the applications are different, we can consider it an extension because we infer the robot pose from the image apart from visualizing the target.

Table 5.1. Training and post-training evaluation parameters for the BM and DRM_{Cp} when DR is applied to the camera pose.

Model	Episode camera z-axis (training)	Episode camera y-axis (training)	Episode camera z-axis (post-training)	Episode camera y-axis (post-training)
BM	180°	−30°	$U(140°, 220°)$	$U(-50°, -10°)$
DRM_{Cp}	$U(160°, 200°)$	$U(-40°, -20°)$		

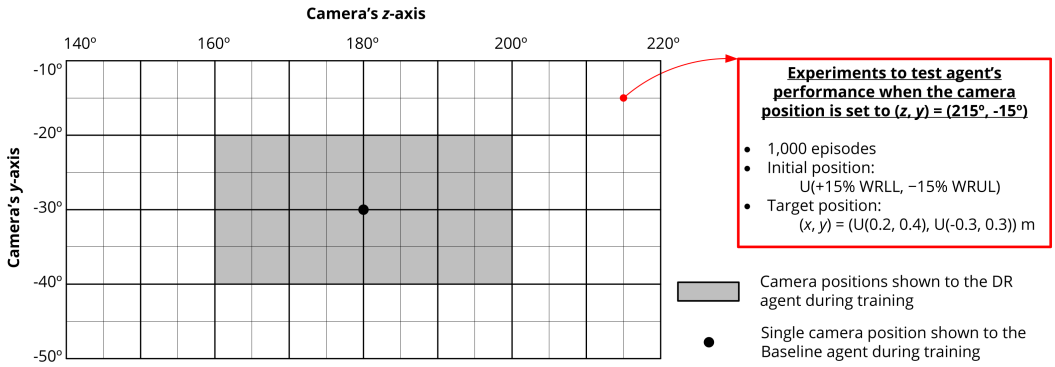
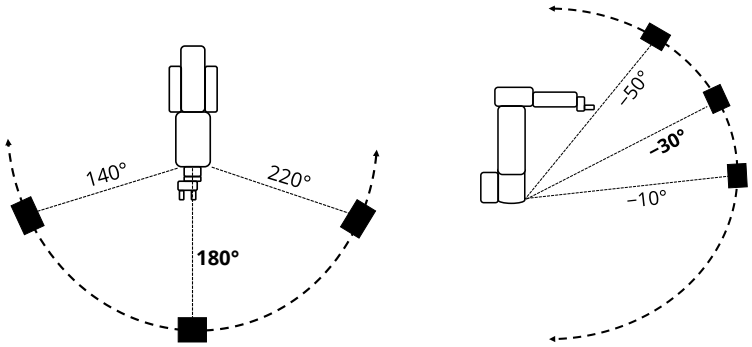


Figure 5.3. Grid designed to evaluate the models with 5° granularity. The black dot represents the camera pose used during training for the BM, while the gray area shows the orientations presented to the DR agent. WRLL and WRUL stand for Working Range Lower Limit and Upper Limit, respectively.



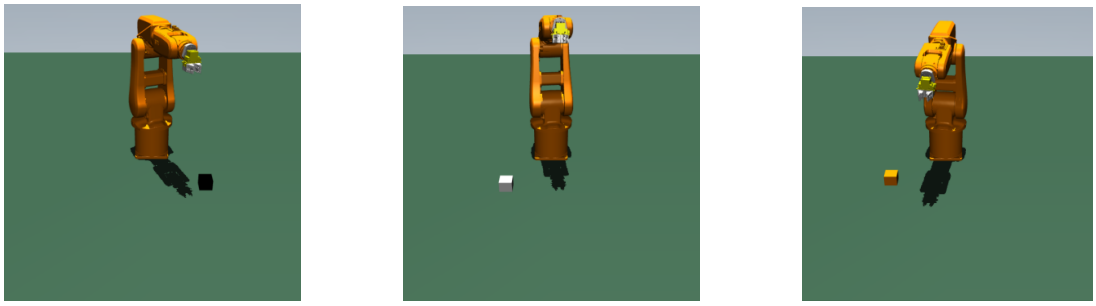
(a) (b)

Figure 5.4. Range of possible camera orientations around the z-axis (a) and y-axis (b), with a constant sphere radius of 2 m centered in the robot base.

- **Target (T)**: The baseline target used was a 6 cm red cube. It is defined as a MuJoCo geometry, so the attributes that can be modified are the color, the shape, and the size. For the **target color (Tc)**, we considered two possible sets of experiments. In the first one ($\text{DRM}_{\text{Tc discrete}}$), we deterministically defined five possible target colors that vary through episodes, e.g., red (1.0, 0.0, 0.0), blue (0.0, 0.0, 1.0), yellow (1.0, 1.0, 0.0), black (0.0, 0.0, 0.0), and white (1.0, 1.0, 1.0). We argue that this color selection is appropriate for effectively learning the representation layers filters (i.e., the CNN filters) as it covers all three channels of the CNN filters as well as the achromatic colors.

  As in the camera pose experiments, during the post-training evaluation we analyzed the $\text{DRM}_{\text{Tc discrete}}$ for 1,000 episodes in each training target color, and included evaluations in unseen environments with violet (0.5, 0.0, 0.5), orange (1.0, 0.5, 0.0) and green (0.0, 1.0, 0.0) targets to test the agent's adaptability. Figure 5.5 shows examples of environment scenarios with the randomized target color in a 640x640 pixel resolution.

  Additionally, we developed another experiment with the target color in which, instead of hand-picking the colors, the training target color was set according to a uniform distribution $U(0.0, 1.0)$ per RGB channel, hoping that the resulting agent's policy was more robust. For the post-training evaluation, we assessed the $\text{DRM}_{\text{Tc uniform}}$ in environments where the target color varied, such as in training, and where it was fixed to the selected colors from the previous experiment. Table 5.2 summarizes the training and post-training evaluation conditions for the BM, the $\text{DRM}_{\text{Tc discrete}}$, and the $\text{DRM}_{\text{Tc uniform}}$.



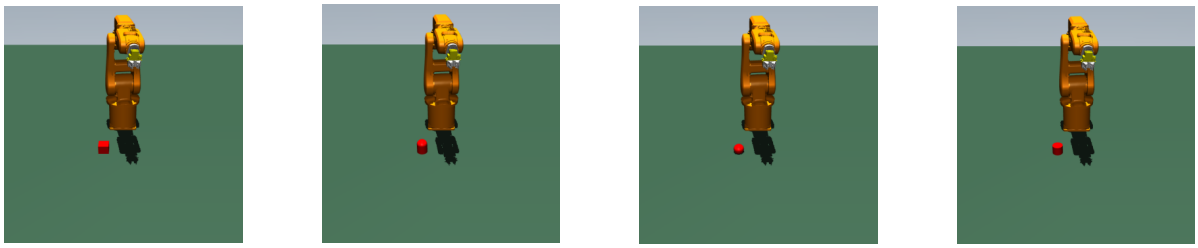
(a) **(b)** **(c)**

Figure 5.5. Environment scenarios with the target color randomized in black (a), white (b), and orange (c). The resolution of these images is 640x640 pixels to ensure better representation in the dissertation.

Table 5.2. Training and post-training evaluation parameters for the BM, the $DRM_{Tc\ discrete}$, and the $DRM_{Tc\ uniform}$ when DR is applied to the target color. The RGB code supplements the color name.

	Target colors	
Model	**Training**	**Post-training**
BM	Red (1.0, 0.0, 0.0)	Red (1.0, 0.0, 0.0) Blue (0.0, 0.0, 1.0) Yellow (1.0, 1.0, 0.0) Black (0.0, 0.0, 0.0) White (1.0, 1.0, 1.0) Violet (0.5, 0.0, 0.5) Orange (1.0, 0.5, 0.0) Green (0.0, 1.0, 0.0)
$DRM_{Tc\ discrete}$	Red (1.0, 0.0, 0.0) Blue (0.0, 0.0, 1.0) Yellow (1.0, 1.0, 0.0) Black (0.0, 0.0, 0.0) White (1.0, 1.0, 1.0)	
$DRM_{Tc\ uniform}$	Red channel $\sim U(0.0, 1.0)$ Green channel $\sim U(0.0, 1.0)$ Blue channel $\sim U(0.0, 1.0)$	

For the **target shape (Tsh)**, MuJoCo offers the possibility of choosing predefined geometries. The DRM_{Tsh} agent was trained in an environment where the target color remained red, but it could be a cube, a capsule, a sphere, or a cylinder. These shapes are depicted in Figure 5.6. The size remains constant at 6 cm per side or radius. The post-training evaluation uses the same shapes since the agent was trained using all possible geometries. Table 5.3 summarizes the training and post-training evaluation conditions for this analysis.



(a) **(b)** **(c)** **(d)**

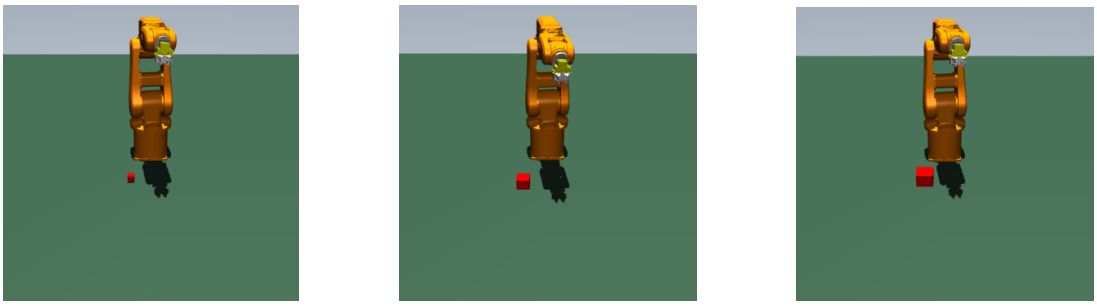
Figure 5.6. Environment scenarios with the target shape randomized in a cube (a), a capsule (b), a sphere (c), and a cylinder (d). The resolution of these images is 640x640 pixels to ensure better representation in the dissertation.

Table 5.3. Training and post-training evaluation parameters for the BM and DRM_{Tsh} when DR is applied to the target shape.

	Target shapes	
Model	**Training**	**Post-training**
BM	Cube	Cube Capsule Ellipsoid Cylinder
DRM_{Tsh}	U([Cube, Capsule, Ellipsoid, Cylinder])	

Finally, we designed an environment where the **target size (Tsz)**, DRM_{Tsz}, can vary within the [3.0, 8.0] cm interval (Figure 5.7). The size value per episode was selected using a uniform distribution. During the post-training evaluation, we assessed how the agent

behaves when the target size is within the interval values and when dealing with sizes out of the training range. Table 5.4 gathers the training and post-training evaluation conditions for this analysis.



(a) (b) (c)

Figure 5.7. Environment scenarios with the target size randomized to 3 cm (a), 6 cm (b), and 8 cm (c). The resolution of these images is 640x640 pixels to ensure better representation in the dissertation.

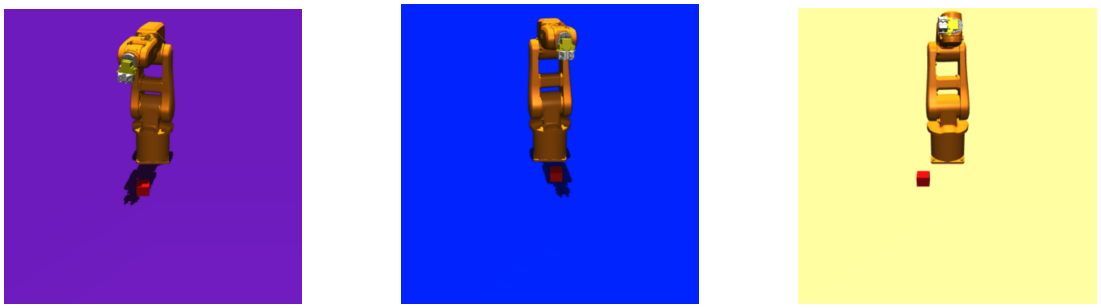
Table 5.4. Training and post-training evaluation parameters for the BM and DRM_{Tsz} when DR is applied to the target size.

	Target sizes (cm)	
Model	**Training**	**Post-training**
BM	6.0	2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
DRM_{Tsz}	$U(3.0, 8.0)$	

- **Background (B) and Ground (G)**: The baseline background has a green (0.19, 0.3, 0.23) floor plane, the ground, and a gray (0.67, 0.71, 0.75) sky. We analyze how DR affects the agent's performance when it is applied to the complete background or the ground.

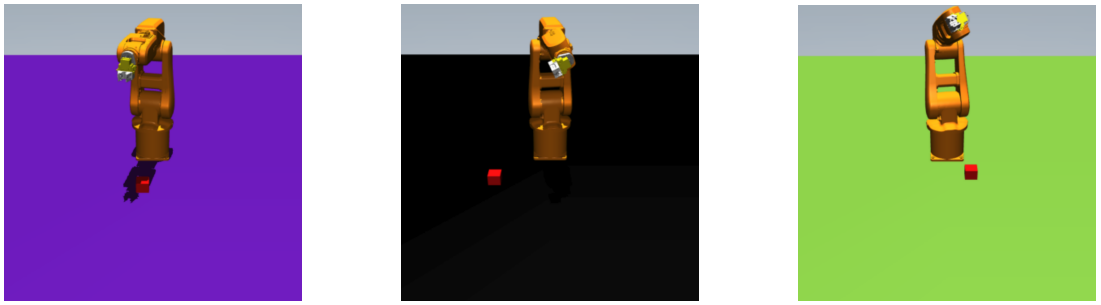
  Since the color will be modified, the process is as in the target color experiments, where a DRM was trained with five colors chosen deterministically, and then evaluated on those setups plus three more colors. For the complete background and the ground only, we chose green (0.19, 0.3, 0.23), violet (0.5, 0, 0.5), blue (0.0, 0.0, 1.0), black (0.0, 0.0, 0.0), and white (1.0, 1.0, 1.0), as the training colors, and yellow (1.0, 1.0, 0.0), orange (1.0, 0.5, 0.0) and light green (0.0, 1.0, 0.0) for the post-training assessment. Afterward, another DRM was trained with a uniform distribution that sets the color on each RGB channel. This model was assessed on the previous eight colors. Figure 5.8 presents some examples of the color modifications in the background, while Figure 5.9 shows some ground changes. On the other hand, Table 5.5 collects the conditions of these sets of experiments, which are equal for the background and the ground.

- **Feature combination**: The agents were trained in the same randomization conditions as their counterparts, i.e., if the camera and target color are simultaneously randomized, the agent is taught with a camera pose of [160°, 200°] for the z-axis and [−40°, −20°] for the y-axis, and the discrete set of target colors (red, blue, yellow, black, and white). Regarding the post-training evaluation, its complexity has increased due to the number



(a) (b) (c)

Figure 5.8. Environment scenarios with the background color randomized to violet (a), blue (b), and a color set by the uniform distribution (c). The resolution of these images is 640x640 pixels to ensure better representation in the dissertation.



(a) (b) (c)

Figure 5.9. Environment scenarios with the ground color randomized to violet (a), black (b), and a color set by the uniform distribution (c). The resolution of these images is 640x640 pixels to ensure better representation in the dissertation.

Table 5.5. Training and post-training evaluation parameters for the BM, the $DRM_{Bc\ discrete}$, the $DRM_{Gc\ discrete}$, the $DRM_{Bc\ uniform}$, and the $DRM_{Gc\ uniform}$ when DR is applied to the background or the ground color. The RGB code supplements the color name.

	Background and ground colors	
Model	**Training**	**Post-training**
BM	Green (0.19, 0.30, 0.23) Gray (sky) (0.67, 0.71, 0.75)	Violet (0.5, 0, 0.5) Blue (0.0, 0.0, 1.0) Green (0.19, 0.3, 0.23) Black (0.0, 0.0, 0.0) White (1.0, 1.0, 1.0) Yellow (1.0, 1.0, 0.0) Orange (1.0, 0.5, 0.0) Light Green (0.0, 1.0, 0.0)
$DRM_{B/Gc\ discrete}$	Violet (0.5, 0, 0.5) Blue (0.0, 0.0, 1.0) Green (0.19, 0.3, 0.23) Black (0.0, 0.0, 0.0) White (1.0, 1.0, 1.0)	
$DRM_{B/Gc\ uniform}$	Red channel $\sim U(0.0, 1.0)$ Green channel $\sim U(0.0, 1.0)$ Blue channel $\sim U(0.0, 1.0)$	

of possible combinations that exist if we proceed as in the individual studies. For this reason, we carry out a general evaluation in which the agent is tested under the same

conditions as in training and, depending on the case and based on the results obtained in the previous individual studies, specific situations are analyzed.

5.3.3. Results and discussion

5.3.3.1. DR applied to the camera pose

Figure 5.10 depicts the average returns obtained during the interim evaluation of the training process for the BM and the (DRM_{Cp}). The BM agent reaches the steady state in around 30 M steps, while the DRM_{Cp} takes 40 M steps. This increment and the fact that the DRM_{Cp} plot is more noisy are probably consequence of the increase in the problem's complexity when DR is applied. The DRM_{Cp} selected for the post-training evaluation is obtained around the 46^{th} M step. Both learning graphs show brief and occasional deteriorations in the agent's performance. We suggest that this is because, in those moments, a suboptimal policy is explored and exploited due to the internal dynamics of the agent in the target estimation or a high learning rate. Thus, a large variation of the model weights might still occur even though they are already optimized and performing well. Although the models' stability and convergence can only be guaranteed in tabular problems, the empirical results show no relevant steadiness issues during training and evaluation.

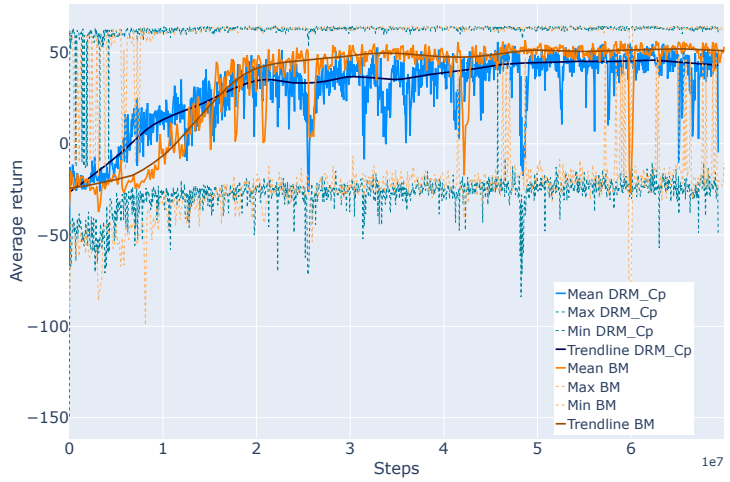


Figure 5.10. Average returns in the interim evaluation for the BM (orange) and DRM_{Cp} (blue) over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure). The BM reaches the steady state, with an average return of around 50, at approximately 30 M steps. In contrast, the DRM_{Cp}'s training curve shows a more noisy pattern, with the steady state starting around 40 M steps and an average return of 40. The noise in the DRM_{Cp} curve is likely caused by the increased problem's complexity.

We use figures inspired by the orthographic projection system to analyze the agents' robustness. As shown in Figure 5.11 for the BM and Figure 5.12 for the DRM_{Cp}, the floor view is a heatmap matrix with the average accuracy obtained after evaluating the agents through all the z and y-axes combinations. The top graph displays the average accuracy along the z-axis when y is fixed to $-30°$. Analogously, the left graph represents the average accuracy for the y-axis when z is constant at 180°.

Figure 5.11 shows that, even though the BM was trained with a fixed camera pose, (180°, $-30°$), the success rate is above 90 % for the interval [165°, 195°] in the z-axis and [$-35°$, $-25°$] in the y-axis. It achieves 100 % in the training viewpoint and the two contiguous spots along the $-30°$ height. The worst agent's performance occurs when the camera is positioned at

the highest position, $-50°$, where neither the position of the robot nor the depth variable can be inferred from the observation.

The accuracy in the z-axis is almost symmetrical along its range, except for the values within the intervals $z \in$[140°, 170°] and $z \in$[190°, 220°] for $y = -40°$. This might be due to the robot's shadow, which reduces the contrast between the target and the floor. Although with less intensity, this effect also happens for those z intervals when $y = -10°$ and $y = -15°$. The agent works better when observations come from the robot's left side, where the shadow is behind the target. Unsurprisingly, the distribution along the y-axis is not symmetrical, as the worst-case scenarios are much better when the camera is on the lowest viewpoint at $-10°$, where the agent can still see the robot joints although it cannot infer the depth, than when it is at $-50°$ where it can barely see the robot.

Analyzing the DRM_{Cp} results in Figure 5.12, the agent can perform above 90 % for almost all orientations between $z \in$[155°, 210°] and $y \in$[$-40°$, $-25°$]. This range is clearly above the one obtained in the BM. Besides, the agent achieves 100 % success rate in eleven poses, most of them within $z \in$[165°, 200°] and $y = -30°$. The symmetry around the z-axis notes that the agent learned to manage better the effect of the shadow than in the BM, despite the 20 % and 15 % drop at $-45°$ and $-15°$, respectively. Besides, the Figure 5.12 z-axis projection shows that the curves are parallel. Around the y-axis, it is interesting to note that the agent's behavior completely changes with respect to the BM. The DRM_{Cp} accuracy decreases at $-10°$ and increases at $-50°$. We argue that, since the number of training steps is the same in both models and in the DRM_{Cp} the experience samples have a wider distribution than in the BM, the DR agent sees more scenarios but less frequently. This might explain why the BM can handle the lack of depth better. Hence, unlike the BM, the lack of symmetry of the DRM_{Cp} around the y-axis suggests that, if there is not enough experience, dealing with partially hidden joints is easier than understanding depth.

Figure 5.13 presents the accuracy increments of the DRM_{Cp} with respect to the BM. In poses near (180°, $-30°$), both agents behave almost perfectly. In the rest of the poses, except for the $y = -10°$ row, the DRM_{Cp} exceeds the BM by 25 % on average, and up to 80 % in some cases. The y-axis projection shape denotes the poor DRM_{Cp} performance when $y = -10°$ and its great results for $y = -50°$. With this final analysis, we can safely state that, in general, the DRM_{Cp} outperforms the BM within the same training steps (i.e., the same effort) despite being exposed to a more variable and, therefore, more complex problem.

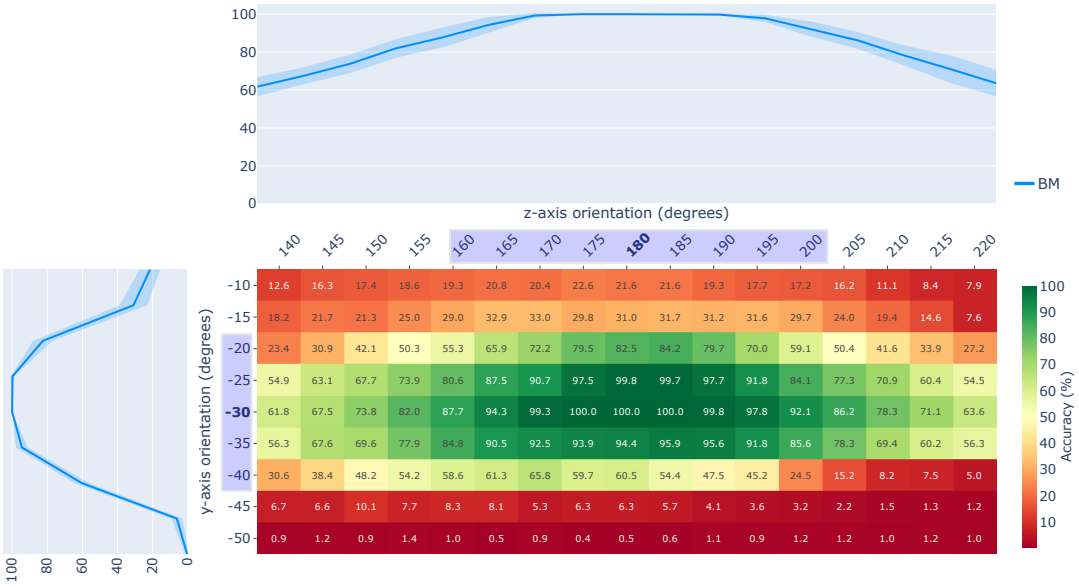


Figure 5.11. Heatmap of the BM accuracy when DR is applied to the camera pose. Each percentage represents the average performance over 1,000 episodes for a given camera pose. The bold orientations on the axes indicate the camera positions used during training. The top graph illustrates the average accuracy along the z-axis, showing the heatmap's projection at a plane perpendicular to the y-axis at $-30°$. The left graph depicts accuracy along the y-axis, corresponding to the projection at a plane perpendicular to the z-axis at 180°. The shaded areas around the lines represent ± 1 std. dev..

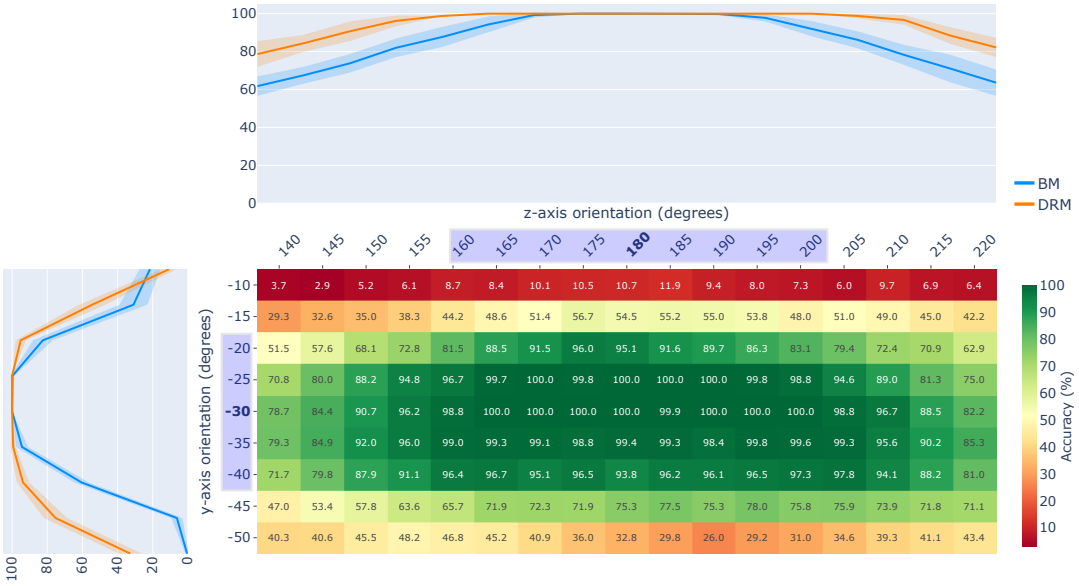


Figure 5.12. Heatmap of the DRM_{Cp} accuracy when DR is applied to the camera pose. Each percentage represents the average performance over 1,000 episodes for a given camera pose. The orientations shaded in purple on the axes indicate the camera positions used during training. The top graph illustrates the average accuracy along the z-axis, showing the heatmap's projection at a plane perpendicular to the y-axis at $-30°$. The left graph depicts accuracy along the y-axis, corresponding to the projection at a plane perpendicular to the z-axis at 180°. The shaded areas around the lines represent ± 1 std. dev..

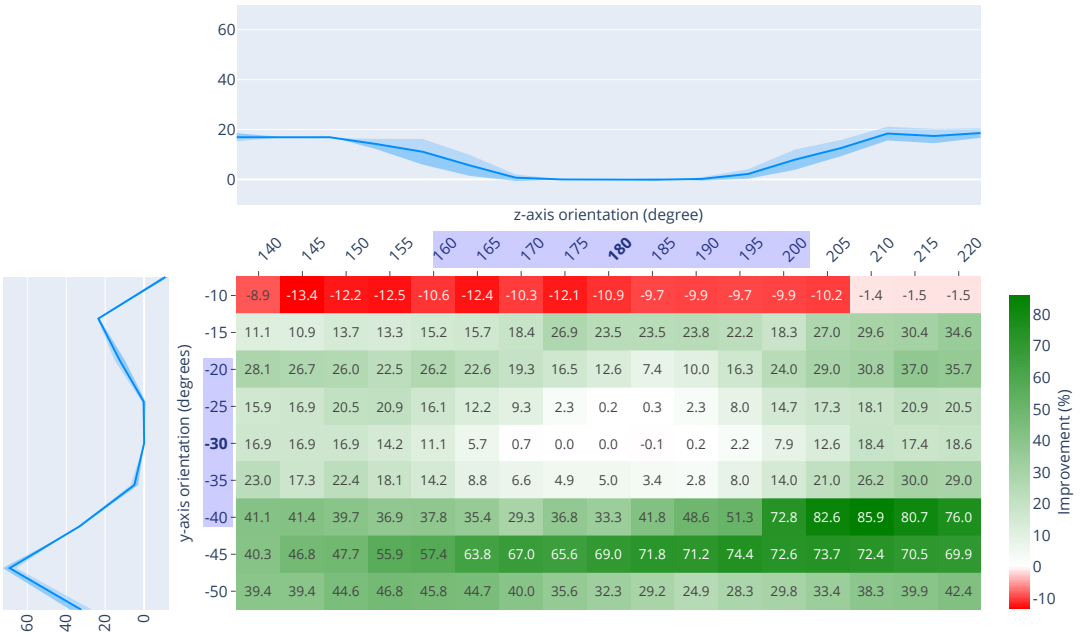


Figure 5.13. Heatmap of the improvement in the average accuracy of the DRM_{Cp} over the BM models. The values displayed represent the accuracy difference over 1,000 test episodes. The orientations shown to the DRM_{Cp} agent are marked in purple, while the single orientation used in the BM is highlighted in bold. The top plot shows the increase in average accuracy along the z-axis, representing the heatmap's projection at a plane perpendicular to the y-axis at $-30°$. The left plot illustrates the accuracy increase along the y-axis, representing the projection at a plane perpendicular to the z-axis at 180°. The shaded area around the line indicates ± 1 std. dev. of the accuracy distribution for each orientation pair.

5.3.3.2. DR applied to the target

5.3.3.2.1. Target color

Figure 5.14 presents the average returns of the training processes for the BM and the DRM where the target color is set according to a predefined list ($\text{DRM}_{\text{Tc discrete}}$), and the DRM where the color is defined by a uniform distribution per RGB channel ($\text{DRM}_{\text{Tc uniform}}$). The $\text{DRM}_{\text{Tc discrete}}$ steady state is reached at 30 M steps, as in the BM, and the best model is in the 33$^{\text{rd}}$ M step, while for the $\text{DRM}_{\text{Tc uniform}}$ this stable period takes place in 30 M steps and its top model is in the 35$^{\text{th}}$ M step. The BM model demonstrates a smoother and more stable training trajectory, reaching a higher average return earlier than the DRM_{Tc} models. Both the $\text{DRM}_{\text{Tc discrete}}$ and $\text{DRM}_{\text{Tc uniform}}$ agents exhibit more variability in their returns, with noticeable fluctuations throughout the training process due to the increase in the environment complexity. However, the trendlines for the DRM_{Tc} models indicate that their overall learning progress positively. The discrete and uniform configurations show similar patterns in the transitory phase, being both slower than the BM.

Table 5.6 gathers the post-training evaluation results. The $\text{DRM}_{\text{Tc discrete}}$ and $\text{DRM}_{\text{Tc uniform}}$ agents achieve accuracy percentages above 90 % in all the colors seen during training and in the ones unseen, except for the green in the case of the $\text{DRM}_{\text{Tc discrete}}$. This might be because, although they are different greens, it is still difficult to distinguish the target from the ground, and during training the agent has not been exposed to enough experience with low contrast between elements. The worst performance in both models is achieved for black, where the $\text{DRM}_{\text{Tc discrete}}$ agent has approximately 92 % accuracy and the $\text{DRM}_{\text{Tc uniform}}$, 94 %. This might happen due to the partial lack of contrast between the green ground and the target, plus the shadow effect. Since both colors are dark and the shadow is black, the agent can lose focus on

the goal as it approaches. As for the maximum failure distances, the $DRM_{Tc\ uniform}$ has lower values than the $DRM_{Tc\ discrete}$, which means that even though the agent fails, it performs a better approach and ends closer to the target. When DR is applied to the target color varying according to uniform distributions, the problem becomes harder as the environment distribution is wider. Still, no prior biases are introduced when choosing the colors, and the agent could better generalize its policy to unseen scenarios. However, even the $DRM_{Tc\ uniform}$ is able to achieve higher returns than the BM, which clearly cannot generalize its knowledge in colors that are far from red, despite achieving 100 % success for violet and orange, which might activate the same CNN filters as the red target. These outcomes show that the target color is an important feature that should be randomized to enhance the agent's ability to deal with varying environments.

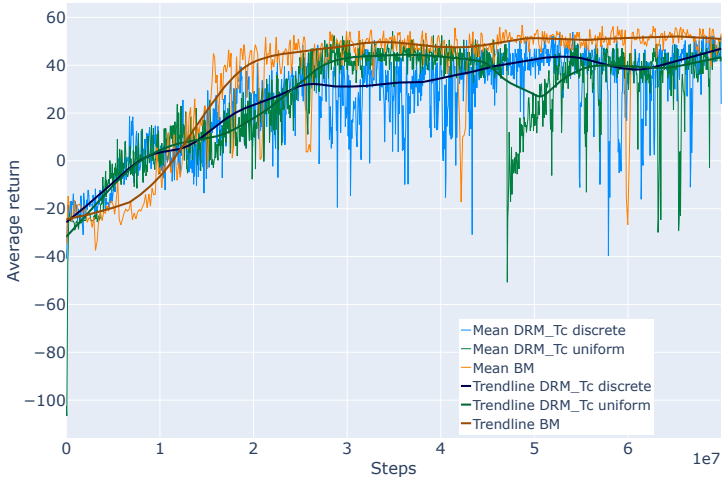


Figure 5.14. Average returns in the interim evaluation for the BM (orange), the $DRM_{Tc\ discrete}$ (blue), and the $DRM_{Tc\ uniform}$ (green) over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure). The $DRM_{Tc\ discrete}$ and $DRM_{Tc\ uniform}$ steady state starts after 30 M steps, as in the BM, although both DRMs achieve lower average returns and are noisier than the BM.

Table 5.6. BM, $DRM_{Tc\ discrete}$, and $DRM_{Tc\ uniform}$ accuracy when DR is applied to the target color. The RGB code supplements the color name.

	Accuracy (%)		
Target colors	**BM**	**$DRM_{Tc\ discrete}$**	**$DRM_{Tc\ uniform}$**
Red (1.0, 0.0, 0.0)	**100**	99.2	98.4
Blue (0.0, 0.0, 1.0)	42.8	**99.4**	98.7
Yellow (1.0, 1.0, 0.0)	25.0	98.8	**99.7**
Black (0.0, 0.0, 0.0)	71.4	91.8	**94.3**
White (1.0, 1.0, 1.0)	72.5	**99.6**	99.5
Violet (0.5, 0.0, 0.5)	**100**	99.0	99.1
Orange (1.0, 0.5, 0.0)	**100**	99.2	99.2
Green (0.0, 1.0, 0.0)	14.3	62.7	**99.3**

5.3.3.2.2. Target shape

Figure 5.15 shows the training outcomes for the BM and the model with the target shape randomized (DRM_{Tsh}). The DRM_{Tsh} curve reaches the steady state in 20 M steps, which is a reduction with respect to the time achieved with the BM, and its leading model is around the 21st M step, 20 M steps before the BM. The DRM_{Tsh} curve, though following a similar overall trend, takes longer to reach a steady state, with more pronounced dips and peaks, indicating more frequent deviations from the optimal performance. These fluctuations are likely attributed to the increased complexity of the training conditions, which result in a noisier learning pattern.

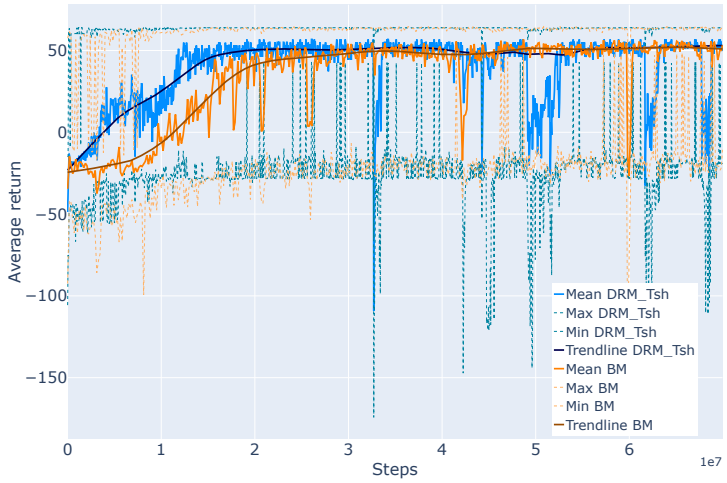


Figure 5.15. Average returns in the interim evaluation for the BM (orange) and DRM_{Tsh} (blue) over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure). The DRM_{TSh} reaches the steady state starting around 20 M steps and an average return of 50, which means an earlier steady state than the BM but with more instabilities.

Table 5.7 summarizes the results from the post-training evaluation. Regarding the DRM_{Tsh} and BM results, which almost all have a 100 % success rate, we can state that randomizing the target shape with the predefined MuJoCo geometries is not a big challenge for the agents. Beyond the particular shape, this agent is probably learning to identify where the object is inside the working area by focusing on its color.

Table 5.7. BM and DRM_{Tsh} accuracy when DR is applied to the target shape.

	Accuracy (%)	
Target shapes	**BM**	**DRM_{Tsh}**
Cube	**100**	99.9
Capsule	100	100
Ellipsoid	100	100
Cylinder	**100**	99.7

5.3.3.2.3. Target size

Figure 5.16 exhibits the target size randomized model, DRM_{Tsz}, and BM training curves. The DRM_{Tsz} steady state is reached in the 45th M step approximately, although the top model is in the 30th M step. The noise and the drop in performance that happens between the 32nd M step and the 42nd M step can again be attributed to the internal dynamics during the learning process or an excessively high learning rate. This can lead to large changes in the model weights

even when they are already optimized. The DRM_{Tsz} model exhibits more variability, with larger deviations between its minimum and maximum values. Despite this, both models ultimately converge toward similar performance levels.

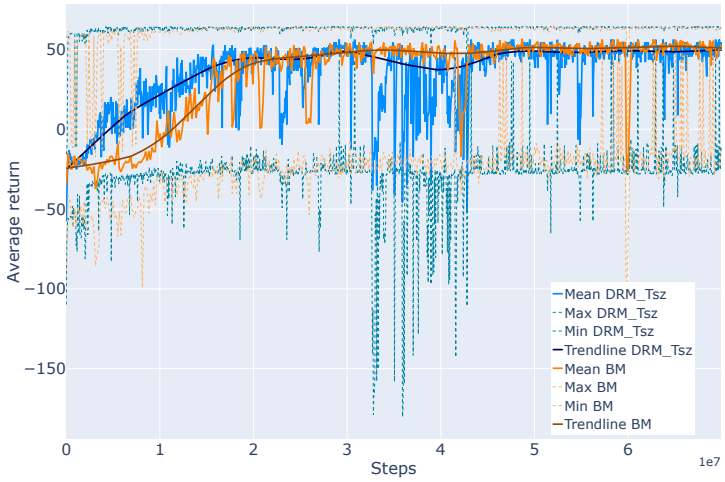


Figure 5.16. Average returns in the interim evaluation for the BM (orange) and DRM_{Tsz} (blue) over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure). The DRM_{Tsz} reaches the steady state starting around 45 M steps, later than the BM, and an average return of 50.

The post-training evaluation outcomes from Table 5.8 indicate that the minimum size the DRM_{Tsz} agent can accurately handle is 4 cm, as the 3 cm cube presents a success rate deterioration that becomes more noticeable in the 2 cm evaluation, which has a performance decrease of about 20 %. Conversely, the BM agent achieves better accuracy for smaller sizes. This suggests that, as with the shape experiments, randomizing the target size it is not as useful as randomizing other features.

Table 5.8. BM and DRM_{Tsz} accuracy when DR is applied to the target size.

	Accuracy (%)	
Target sizes (cm)	**BM**	**DRM_{Tsz}**
2.0	**85.7**	77.4
3.0	**98.9**	97.7
4.0	100	100
5.0	**100**	99.8
6.0	100	100
7.0	100	100
8.0	**100**	99.9
9.0	**100**	99.7

5.3.3.3. DR applied to the background and ground

In this set, randomization is applied separately to the complete background and to the ground. Figure 5.17 plot displays the training process of two agents with the background color randomized, $DRM_{Bc\ discrete}$ and $DRM_{Bc\ uniform}$, as well as the BM. $DRM_{Bc\ discrete}$ and $DRM_{Bc\ uniform}$ steady states are located in the 20^{th} M and 30^{th} M step, respectively. The learning of the $DRM_{Bc\ discrete}$ is less noisy than the $DRM_{Bc\ uniform}$ curve, possibly because the environment distribution is smaller. The $DRM_{Bc\ uniform}$ average returns are much lower than the $DRM_{Bc\ discrete}$ and the BM due to its increased environment complexity. The best models are found in steps 30 M for the discrete color distribution and 31 M for the uniform distribution.

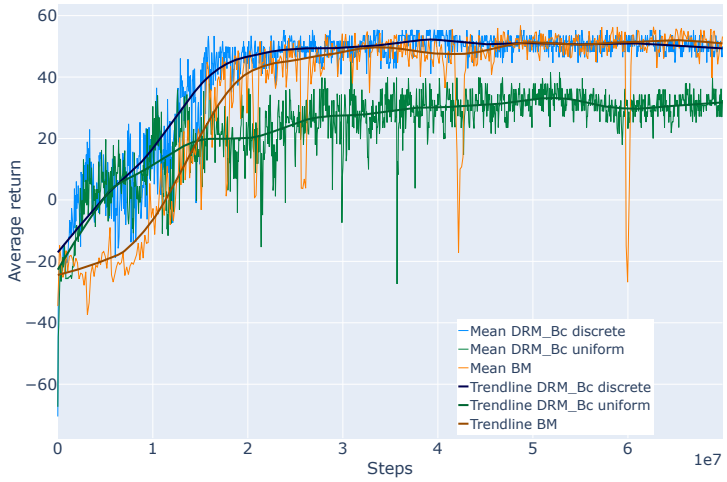


Figure 5.17. Average returns in the interim evaluation for the BM (orange), the $\text{DRM}_{\text{Bc discrete}}$ (blue), and the $\text{DRM}_{\text{Bc uniform}}$ (green) over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure). The $\text{DRM}_{\text{Bc discrete}}$ and $\text{DRM}_{\text{Bc uniform}}$ steady state starts after 20 M and 30 M steps, although $\text{DRM}_{\text{Bc uniform}}$ remains at a lower performance level than the others.

Regarding the background modification, Table 5.9 gathers the accuracy results for the three models under the post-training evaluation conditions. Overall, $\text{DRM}_{\text{Bc discrete}}$ shows consistent high accuracy across most background colors, outperforming both BM and $\text{DRM}_{\text{Bc uniform}}$. For instance, in the violet, blue, and green backgrounds, $\text{DRM}_{\text{Bc discrete}}$ achieves accuracies above 96 %, while BM struggles. The $\text{DRM}_{\text{Bc uniform}}$ model performs better than the BM in some cases, such as blue and black backgrounds, but generally lags behind $\text{DRM}_{\text{Bc discrete}}$. Note that in the yellow and orange backgrounds, none of the models achieve particularly high accuracy, which might be because the robot's color overlaps with the background, making it challenging for the agent to distinguish the joint poses.

Table 5.9. BM, $\text{DRM}_{\text{Bc discrete}}$, and $\text{DRM}_{\text{Bc uniform}}$ accuracy when DR is applied to the background color.

	Accuracy (%)		
Background colors	**BM**	**$\text{DRM}_{\text{Bc discrete}}$**	**$\text{DRM}_{\text{Bc uniform}}$**
Violet (0.3, 0.0, 0.5)	32.4	**96.6**	68.5
Blue (0.0, 0.0, 1.0)	64.6	**96.7**	78.7
Green (0.19, 0.30, 0.23)	25.2	**96.6**	60.3
Black (0.0, 0.0, 0.0)	88.0	**97.2**	76.4
White (1.0, 1.0, 1.0)	80.0	**96.0**	78.8
Yellow (1.0, 1.0, 0.0)	**56.5**	48.0	41.7
Orange (1.0, 0.5, 0.0)	36.0	**36.6**	32.6
Light Green (0.0, 1.0, 0.0)	36.1	0.0	**68.7**

Examining the effect of the ground color change, Figure 5.18 shows the training curves of the randomized agents with the discrete and uniform color selection, $\text{DRM}_{\text{Gc discrete}}$ and $\text{DRM}_{\text{Gc uniform}}$, respectively, and the BM. The steady state for the $\text{DRM}_{\text{Gc discrete}}$ starts at 35 M steps, while for the $\text{DRM}_{\text{Gc uniform}}$ it begins in 25 M steps. However, $\text{DRM}_{\text{Gc uniform}}$ has a more unstable learning process as, unlike the previous cases with the sudden and brief performance

decrease, the agent seems to forget the already acquired knowledge around the 45th M step, and it has to re-learn it again. This might be caused by either the internal dynamics of the learning process or the learning rate value. The best models for the $DRM_{Gc\ discrete}$ and the $DRM_{Gc\ uniform}$ are both in their 30th M step.

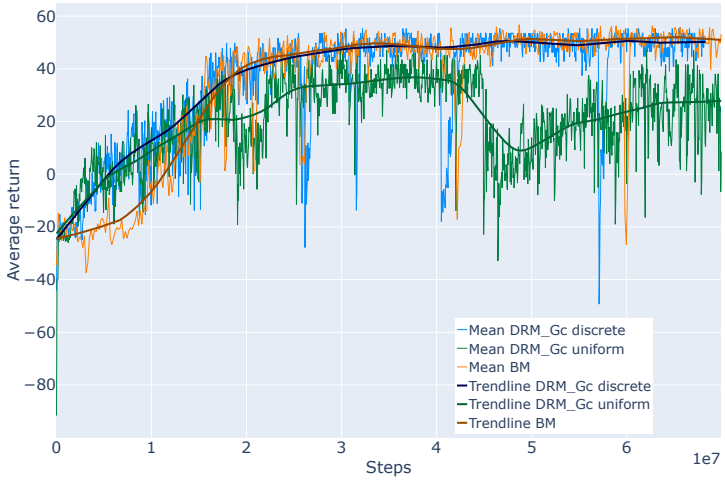


Figure 5.18. Average returns in the interim evaluation for the BM (orange), the $DRM_{Gc\ discrete}$ (blue), and the $DRM_{Gc\ uniform}$ (green) over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure). The $DRM_{Gc\ discrete}$ and $DRM_{Gc\ uniform}$ steady state starts after 35 M and 25 M steps, although $DRM_{Bc\ uniform}$ remains at a lower performance level than the others. The $DRM_{Gc\ discrete}$ and the BM reach the same steady state average return, but the former is trained in a more complex environment.

Finally, Table 5.10 presents the evaluation accuracy obtained in the post-training evaluation. The $DRM_{Gc\ discrete}$ excels in all the seen colors but is unable to generalize, as the results for the unseen environments are very poor. By contrast, despite its unstable learning process, the $DRM_{Gc\ uniform}$, which was exposed to more scenarios during its training, achieves success rates above 97 % for all colors except for the black. It might be the consequence of the absence of the robot's shadow, since in this scenario, it cannot be distinguished from the ground. We consider it clear evidence that the agent relied on the shadow for part of its acting decisions, thus the shadow was removed from the virtual environment hereafter.

Table 5.10. BM, $DRM_{Gc\ discrete}$, and $DRM_{Gc\ uniform}$ accuracy when DR is applied to the ground color.

	Accuracy (%)		
Ground colors	**BM**	**$DRM_{Gc\ discrete}$**	**$DRM_{Gc\ uniform}$**
Violet (0.3, 0.0, 0.5)	2.5	**100**	99.6
Blue (0.0, 0.0, 1.0)	12.5	99.1	**99.8**
Green (0.19, 0.3, 0.23)	97.5	**99.8**	99.5
Black (0.0, 0.0, 0.0)	57.8	**100**	47.4
White (1.0, 1.0, 1.0)	56.2	**100**	99.2
Yellow (1.0, 1.0, 0.0)	2.5	18.8	**98.8**
Orange (1.0, 0.5, 0.0)	0.0	10.4	**97.0**
Light Green (0.0, 1.0, 0.0)	15.7	55.6	**99.1**

5.3.3.4. DR applied to a combination of features and elements

After the previous results obatained with a single environment modification per model trained, we designed different combinations of features that could change within the same training process. Figure 5.2 presents the combinations made on this second approach. In this section, only the most relevant insights will be commented on, leaving to Appendix A the presentation of figures with the average returns obtained in the interim evaluations. In general, in these curves the learning process is generally slower than in the models with one environment modification due to the increase in the difficulty of the problem. Besides, even though the graphs still show sudden valleys, they are not as noticeable as in the previous scenarios because the curves are noisier as a consequence of the high variance in the environment.

With respect to the success rates, the DRM_{CpTc} and $\text{DRM}_{\text{CpTcTshTszGc}}$ agents are below the minimum reached by the other models, which are around 97 %. In the case of the DRM_{CpTc}, the evaluations carried out with the camera pose randomized and each discrete color fixed achieve about 90 % accuracy, being the worst scenarios the ones with the black (77.6 %), and green (67.8 %) target. These performance decreases due to the black and green colors had also been noticed in Section 5.3.3.2 when the target color effect was analyzed individually and in all the DRM where this feature is involved. The $\text{DRM}_{\text{CpTcTshTszGc}}$ has the lowest performance of all the trained models, 87.8 %. This is consistent with the fact that it is the most challenging environment as five features are randomized. According to its training curve, we argue that increasing the learning time in this model would not necessarily lead to better outcomes since it has already reached the steady state within the 70 M steps.

To conclude, these results might suggest that one of the best approaches to tackle environment randomization would be to randomize one or two elements, in case they are not related, i.e., the camera and the ground, or even three if they are, as in the $\text{DRM}_{\text{TcTshTsz}}$ where the complete randomization is about the target, and transfer the knowledge between different agents. This way, the scenario complexity is contained and the high success rates agents can achieve in simpler environments are preserved. This knowledge share can be achieved, for example, by means of the architecture proposed with PNNs, which is discussed in more detail in Chapter 6.

5.4. Low-level Domain Randomization

5.4.1. Method

Applying high-level DR to visual elements is an appropriate approach for addressing transfer learning between environments, provided that the images start with a reasonable similarity. In other words, it can be useful for fine-tuning or enhancing the agents' ability to generalize in response to the variability of specific elements within the scenario. However, if the problem is inherently visual and the two domains exhibit significant discrepancies from the outset, this strategy is not valid. As demonstrated, applying DR to multiple elements simultaneously can deteriorate the agents' performance. Therefore, in another attempt to leverage the benefits of DR, its application has been explored at a lower level, where randomization is introduced as Gaussian noise to the image pixels. This approach builds on the idea that using purely raw synthetic observations in artificial vision often yields suboptimal results. The objective is to align the distributions of each RGB channel of the virtual and real observations without explicitly modeling the real observation noise, ensuring that the agent does not perceive the two domains as vastly different.

Figure 5.19 depicts the methodology followed. We first performed a zero-shot with the BM trained with the orientation control. We then analyzed the scenario mismatches through a histogram analysis of the real and virtual observations. Afterward, we designed, trained, and evaluated different agents in the virtual environment applying DR as Gaussian noise, bringing the histograms closer. Finally, we completed a zero-shot transfer per each DRM trained to quantify the gains obtained.

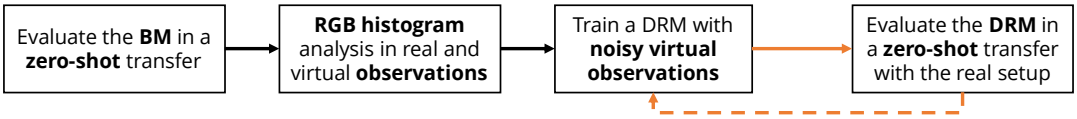


Figure 5.19. Methodology of the zero-shot transfer experiments with low-level DR. First, we evaluate the BM performance in the real environment in a zero-shot transfer. Then, the RGB histograms of the real and virtual observations are analyzed and, based on these results, a set of DRM agents are designed and virtually trained with noisy observations. Their zero-shot evaluation in the real setup determines the changes to be made in the following agents. The process continues until the point in which adding more noise to the image does not improve the results.

5.4.2. Description of the experiments

Virtual and real image differences are analyzed using the histograms of each RGB channel separately. To modify the RGB pixel distributions in the virtual environment, instead of modeling the real noise distribution, which is time-consuming and might vary depending on external factors, such as illumination changes, we define a Gaussian noise tensor per episode step with the same dimensions as the image and add it to the raw virtual observation. The resulting pixel values are within the interval [0, 255] to avoid infeasible color codes. The Gaussian noise is determined by a random normal distribution with a mean of 0 and a standard deviation (std. dev.) that is adjusted between experiments.

Additionally, instead of randomizing the target position in the virtual scenario according to the two uniform distributions (Section 4.2.1), we use 125 saved ArUco positions available in the real setup (Section 3.2.3). The rest of the considerations and hyperparameters for the virtual training step and the post-training evaluation in the virtual setup are the same (Table 4.2). Shortly, we train for around 30 M steps and do the interim evaluation every 50 k steps for 40 episodes with fixed initial robot and target poses and a 5 cm rewarding distance. The post-training assessment is done with another seed for 1,000 episodes with random initial robot and target poses and a 10 cm rewarding distance.

After the post-training evaluation in the virtual setup, we performed a zero-shot deployment to check if the trained models with noisy images improved the success rate in the real environment with respect to the BM. With this zero-shot we were not looking for very high success rates or a completely successful transfer. We seek to prove the hypothesis that adding Gaussian noise to the virtual image makes it easier for the agent to re-use its learned policy in the real setup. Hence, when the integration of these randomized models with the approximation proposed by the PNNs is done, the probability of having a successful few-shot transfer with fewer samples should be increased. This zero-shot test was performed following the same conditions as a post-training evaluation. We evaluated 1,000 episodes with random initial robot and target poses per episode and a 10 cm rewarding distance.

5.4.3. Results and discussion

Figure 5.20 exhibits the image histograms for the BM and the real scenario. The RGB channels from the raw virtual observation fit an almost identical bimodal distribution, with negligible noise levels. Conversely, the real images show noisy RGB pixel distributions and do not preserve the same shape between channels. Apart from the discrepancies in the virtual and real scene colors, we must consider that the camera is a source of noise. Therefore, it becomes clear that applying only the high-level DR strategy will not be enough to raise the possibilities of a successful transfer. Instead, we should modify the complete virtual image by modifying its pixel distributions.

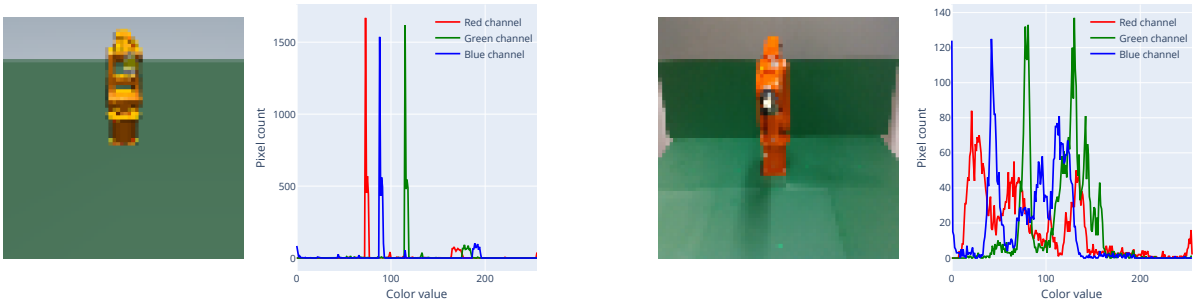


Figure 5.20. RGB histograms for the raw virtual (a) and real (b) observations.

Table 5.11 presents the outcomes for the noisy models trained in the virtual environment and their accuracy in the zero-shot evaluation on the real setup. Starting from the success rate achieved by the BM, the objective is to surpass the 15.8 % accuracy obtained in its zero-shot. To achieve this, Gaussian noise is added to the models' observations progressively, assuming that at some point, no matter how much noise is added, the efficiency will start to decline. Adding noise with low std. dev. already improves the agent's performance in zero-shot 5 % while mastering the task in the virtual environment. As we keep increasing the std. dev., the high accuracies in the virtual scenario remain, and the zero-shot success rates rise until a Gaussian noise with a std. dev. of 100 is reached. The next agent with a std. dev. of 120 decreases its yield to the level of the model with $\sigma = 80$.

Table 5.11. Accuracy when DR is applied as Gaussian noise to the raw virtual observation. The first column indicates the standard deviation of the Gaussian distribution used to introduce the noise.

	Accuracy (%)	
Model std. dev.	**Post-training evaluation (virtual environment)**	**Zero-shot transfer (real environment)**
BM ($\sigma = 0$)	99.8	15.8
$\sigma = 10$	99.6	21.2
$\sigma = 20$	100.0	20.7
$\sigma = 40$	100.0	25.3
$\sigma = 60$	99.1	24.5
$\sigma = 80$	99.9	28.7
$\boldsymbol{\sigma = 100}$	**99.8 %**	**34.1 %**
$\sigma = 120$	98.9 %	28.4 %

For the $\sigma = 100$ model, the zero-shot accuracy is 34.1 %, more than twice the percentage obtained with the BM. Figure 5.21 depicts the observation seen by the agent with the $\sigma = 100$

noise and the comparison of the new histogram with respect to the one from the real observation. Even though the distributions are still not that similar since modeling the real noise using a normal distribution does not accurately reproduce what happens in the real environment, we achieved a higher level of correspondence with respect to what was initially available with the BM. Moreover, as discussed in the previous experiments with DR, this is a gain that is obtained without additional computational cost and, therefore, with the same effort. This can be observed in Figure 5.22, which presents the agent's training curve compared with the BM. Note that in this process, the steps are lower as the actions command the joints' orientation, not the torque. The training of the BM concluded earlier due to the fact that it reached a highly stable and robust steady state. In contrast, the training of the DRM is more unstable, which is likely a result of the increased environmental complexity in its setup. Despite these variations in training dynamics, both models eventually converged to almost the same level of performance.

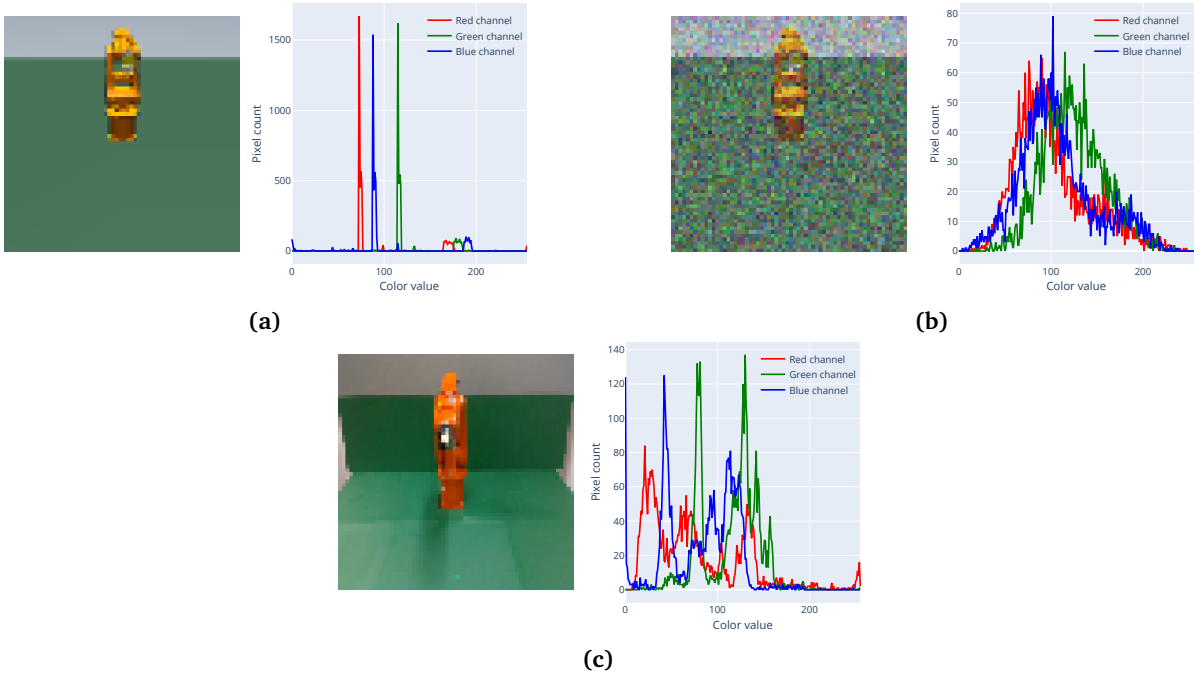


Figure 5.21. RGB histograms for the raw virtual (a), noisy virtual (b), and real (c) observations.

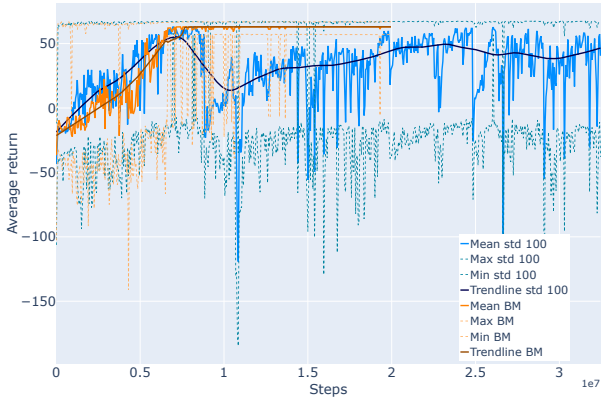


Figure 5.22. Average returns in the interim evaluation for the BM (orange) and DRM agent (blue) trained with noisy observations with std. dev. of 100 for almost 30 M steps (note the 1e7 scaling factor on the right-hand corner of the figure). The BM training was stopped earlier as it achieved a very robust steady state. Due to the environment complexity, the DRM training is more unstable than in the BM.

5.5. Conclusion

This chapter has explored the use of DR as a possible technique to bridge the reality gap. First, we have studied in a sim-to-sim approach how applying DR to different high-level environment features improves the generalization capability of the agents. Through a detailed analysis, we have identified key high-level environment features –the camera pose, and the target and background color– that influence the most the agent's performance. The study showed an important limitation: there might be a practical limit to the number of high-level features that can be randomized simultaneously without compromising the agent's ability to generalize effectively. If the number of randomized features increases too much, it may complicate or destabilize the training process, given a fixed model capacity, preventing the agent from correctly approximating what it perceives. In our experiments, when multiple complex features are varied together, the agent's performance tends to decline, revealing that while DR can be powerful, it might not be a one-size-fits-all solution. This insight highlights the need for a balanced approach when applying DR in an environment where several features can be randomized.

In addition to high-level DR, we also experimented with low-level DR, where noise was introduced at pixel level using Gaussian distributions. This approach aimed to shrink the gap between virtual and real observations by aligning the RGB pixel distributions more closely. The results indicated that, while low-level DR can improve the agent's robustness, there is a threshold beyond which additional noise degrades performance. The experiments suggest that although DR can reduce discrepancies between simulated and real environments, it is insufficient on its own for achieving a perfect zero-shot transfer, especially in cases with significant mismatches between the real and virtual domains.

In conclusion, while DR enhances agents' performance without increasing computational cost or training time in the virtual environment, its effectiveness can be limited in practice by the complexity of the features being randomized and the extent of the discrepancies between virtual and real-world setups. Thus, DR is a valuable tool for narrowing the reality gap, but it should be part of a broader transfer learning strategy, integrating complimentary techniques, like Progressive Neural Networks (PNNs) or Domain Adaptation (DA), to ensure successful deployment in the real scenarios.

6

Progressive Neural Networks

Your reward will be the widening
of the horizon as you climb.
Cecilia H. Payne-Gaposchkin
(1900–1979)

This chapter focuses on the use of PNNs to bridge the reality gap. It begins by setting their fundamentals and describing the PNN architecture implemented. Then it analyzes the PNNs' learning dynamics through a series of experiments in both sim-to-sim and sim-to-real approaches. It evaluates how PNNs handle knowledge retention, catastrophic forgetting, the adaptation to new, more complex environments, and their integration with DR. The results of these experiments are gathered in a paper named "Evaluating the perception, understanding, and forgetting of Progressive Neural Networks: a quantitative and qualitative analysis," which is under review at the moment of writing.

The following section defines and presents the results of the methodology that has to be followed to bridge the reality gap in a few-shot approach using a PNN architecture, analyzing in detail how the agents perform in the real environment and assessing their generalization capabilities.

6.1. Progressive Neural Networks fundamentals

Progressive Neural Networks (PNNs) [Rus+16b] are a knowledge transfer technique that consists in sharing the representations learned by one or more teacher networks with a smaller student network. The networks, also known as columns due to their graphical depiction, represent a DRL agent solving an MDP that is somehow related to the MDPs from the other columns. Knowledge transfer is handled through learnable lateral connections that join the networks' layers.

PNNs are based on knowledge distillation [Rus+16a] and framed into the continual or lifelong learning area as new agents can be trained in other tasks or setups by reusing previously acquired knowledge. Hence, avoiding catastrophic forgetting of the previous know-how is critical. PNNs attempt to solve this issue explicitly with their architecture design since the

teachers' weights are frozen during the student learning process. Nevertheless, it is not guaranteed that the student agent is as skilled as its teachers in all previous tasks.

6.1.1. Architecture and implementation

The PNNs architecture can be used to tackle the sim-to-real problem if we think of a teacher trained in the virtual environment and a student that learns with little experience from the real setup plus the knowledge shared by the teacher. Hence, PNNs fall into the few-shot category since the agent deployed in the real scenario needs some samples to refine its behavior. Figure 6.1 presents a schematic diagram of our implemented architecture. It is deeply based on the proposal of [Rus+17] but with some changes in its representation, as we have added the output layer lateral connections from the first-column LSTM, which were mentioned but not depicted[1]. The first stream corresponds to the agent trained in the virtual environment, and the second to the agent trained in the real scenario. Although both have the same number of layers, the student is smaller than the teacher (i.e., it has fewer learnable parameters) as it will leverage the teacher's knowledge.

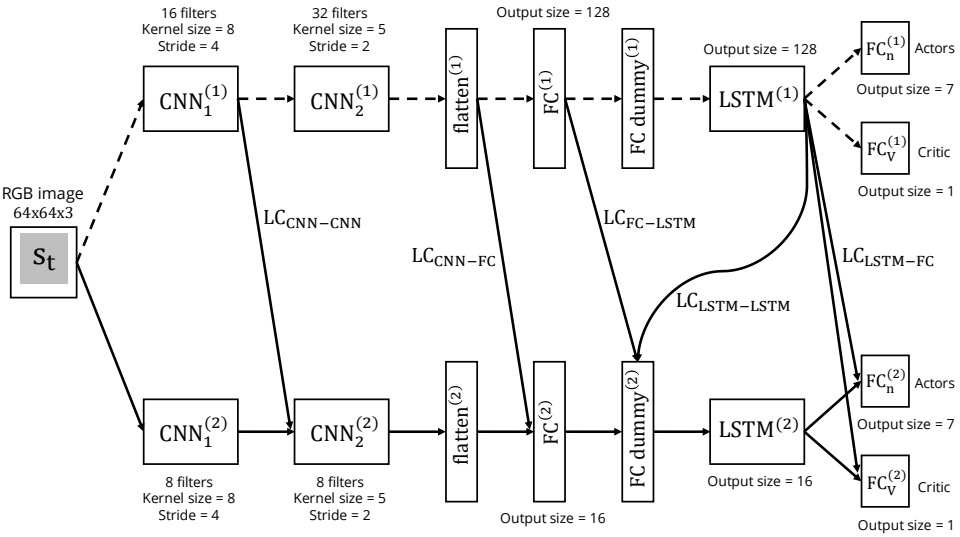


Figure 6.1. Schematic diagram of the PNN architecture implemented. The first stream corresponds to the first (top) column or teacher, and the second (bottom) is the student. LC stands for lateral connection. The flatten layer resizes the CNN_2 outputs to row vectors. The dummy layers only handle the LSTM lateral connections. Dashed lines indicate frozen parameters during the training phase of the student. In our problem $n \in \{1, 2, ..., 6\}$ as the action commands the six robot joints.

We argue that the PNN state representation layers correspond to the first two convolutional layers, CNN_1 and CNN_2. The following blocks, the Fully Connected (FC), the LSTM, and the FC from the output layers, learn the optimal policy. The columns' output is divided into the actor and the critic FC layers as we implement an A3C as the DRL algorithm (Section 4.1). The actor FC blocks approximate the policy that has to follow each robot joint, six policies in total, while the critic FC block estimates the state-value function. The first convolutional layer applies 16 filters for the teacher and 8 for the student with a kernel size of 8 and a stride of 4, followed by a second layer with 32 and 8 filters, respectively, a kernel size of 5 and a stride of 2. These layers extract essential features, reducing the dimensionality of the image data while retaining critical spatial information. The output from these convolutional layers is flattened into a row vector and fed into FC layers, where the first produces an output size of 128 and 16,

[1]We appreciate the detailed answers Andrei A. Rusu and his co-authors gave to our questions.

respectively, which is then passed through a LSTM to finally reach the last FC layer that will determine each joint policy and the value of the state value function. The dummy layer before the LSTM is defined as a FC with weights fixed at one, as it only facilitates the management of the LSTM lateral connections.

Equation (6.1) presents the general formulation of the hidden activations h for layer i and column k [Rus+16b]. Biases are omitted for clarity. $h_i^{(k)}$ is a sum of two terms: the weight matrix, $W_i^{(k)} \in \mathbb{R}^{n_i \times n_{i-1}}$, times the hidden activation $h_{i-1}^{(k)}$ of the previous layer, plus the sum for all previous columns of the learnable lateral connection matrices $U_i^{(k:j)} \in \mathbb{R}^{n_i \times n_j}$ times the hidden activation $h_{i-1}^{(j)}$ from column j in the previous layer. f is an element-wise non-linearity applied to the result of the above addition. For intermediate layers, it is defined as a Rectified Linear Unit (ReLU) $f(x) = \max(0, x)$.

$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)}\right) \tag{6.1}$$

Figure 6.2 shows the details of the diagram presented in Figure 6.1 with our PNN architecture and our formalization of Equation (6.1). In Figure 6.2, tensors and vectors named $B_n^{(k)}$ correspond to the first term of the sum in Equation (6.1), and $A_n^{(j)}$ to the second. Hence, we can particularized Equation (6.1) as:

$$h_i^{(2)} = f\left(W_i^{(2)} h_{i-1}^{(2)} + U_i^{(1)} h_{i-1}^{(1)}\right) = f\left(B_i^{(2)} + A_i^{(1)}\right) = f\left(M_i^{(2)}\right) \tag{6.2}$$

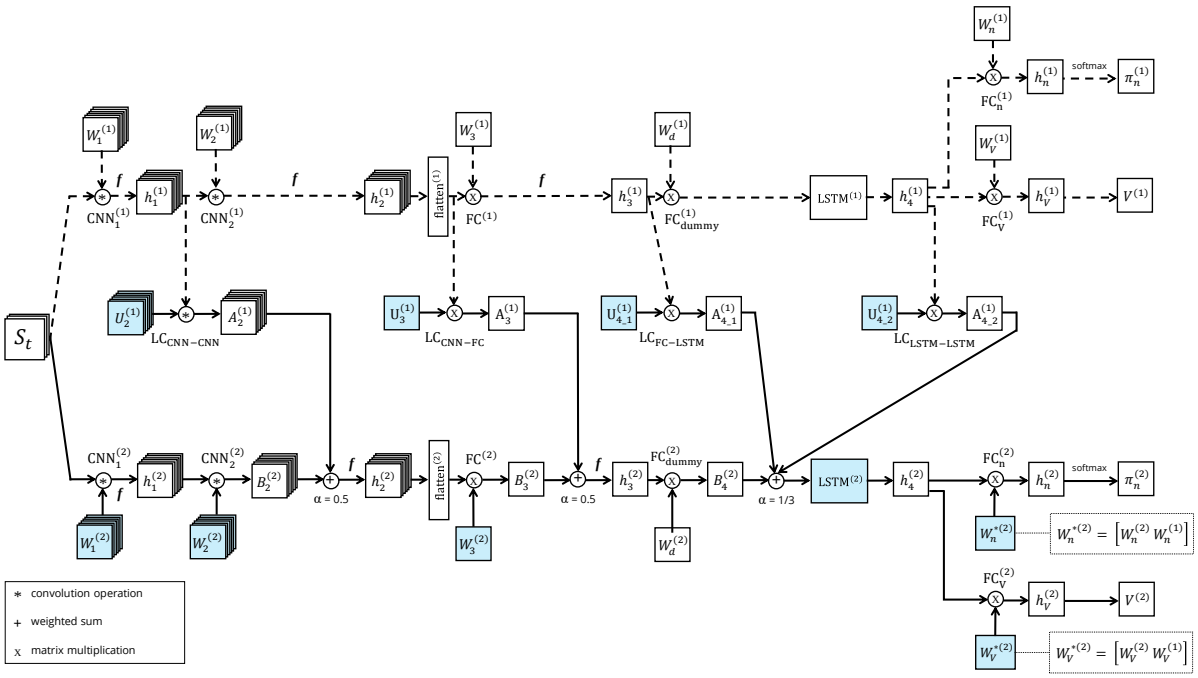


Figure 6.2. Detailed diagram of the PNN architecture implemented. The top stream corresponds to the first column or teacher, and the bottom is the student. There are 4 explicit lateral connections plus the connections in the output layers that are made through weight concatenation. All the connections, except the last one, are implemented as weighted sums. The dashed lines represent frozen parameters during the training phase of the second-column layers. The blue blocks are learnable parameters.

$W_d^{(1)}$ and $W_d^{(2)}$ are the dummy tensors that have no effect on the learned policy, as they are filled with ones, and are only added to handle the two LSTM lateral connections. Lateral connections are implemented as a weighted sum of features. If the block has one connection, tensors are weighted by $\alpha = 0.5$, while if it has two, they are weighted by $\alpha = 1/3$. We argue this is a plausible decision since it gives the same importance to features from both columns, and we avoid scalability issues if the number of lateral connections increases. The actor and critic lateral connections do not follow Equation (6.1) as there is no trainable matrix $U^{(1)}$. Their weight matrices, $W_n^{*(2)}$ for the actors, and $W_V^{*(2)}$ for the critic, are a concatenation of the learnable parameters from column 2, $W_n^{(2)}$ and $W_V^{(2)}$ respectively, with the $\text{FC}_n^{(1)}$ and $\text{FC}_V^{(1)}$ weight matrices, $W_n^{(1)}$ and $W_V^{(1)}$. This layout allows the initialization of the second-column output with the policies and state-value function from the first column. These initial values provide a crucial guide to the agent in its learning process. We apply the softmax function to translate the FC outputs' activations into the probability that defines the policy $\pi_n^{(k)}$.

The first lateral connection, $\text{LC}_{\text{CNN-CNN}}$, links $\text{CNN}_1^{(1)}$ with $\text{CNN}_2^{(2)}$ (Figure 6.3). Tensor $B_2^{(2)}$ sets the size of tensor $A_2^{(1)}$ as both will be weighted and added together. Hence, the size of tensor $U_2^{(1)}$ has to be calculated according to the dimension of $h_1^{(1)}$ after $\text{CNN}_1^{(1)}$. Figure 6.4 shows the next lateral connection $\text{LC}_{\text{CNN-FC}}$, which joins $\text{CNN}_2^{(1)}$ with $\text{FC}^{(2)}$. The flatten layer resizes features into a vector, and as in the $\text{LC}_{\text{CNN-CNN}}$, matrix $U_3^{(1)}$ adapts the dimensions to enable the posterior weighted sum of $A_3^{(1)}$ and $B_3^{(2)}$. The following two lateral connections belong to the LSTM layer, $\text{LC}_{\text{FC-LSTM}}$ and $\text{LC}_{\text{LSTM-LSTM}}$. Equation (6.2) is then slightly modified as there are two $A_i^{(1)}$ terms. As Figure 6.5 presents, the $\text{LSTM}^{(2)}$ input is obtained from the weighted sum of the $\text{FC}^{(2)}$ output $h_3^{(2)}$, with the lateral connection from $\text{FC}^{(1)}$, whose resize is done through $U_{4\_1}^{(1)}$, and the lateral connection from $\text{LSTM}^{(1)}$, whose learnable matrix is $U_{4\_2}^{(1)}$. In this case, the weighting parameter is $\alpha = 1/3$. $W_d^{(j)}$ are dummy tensors filled with ones to manage the multiple connections. Finally, Figure 6.6 depicts the lateral connections from the LSTM to each FC output layer, $\text{LC}_{\text{LSTM-FC}}$. In this case, they do not follow Equation (6.1). There is no trainable matrix $U^{(1)}$, but weight matrices, $W_n^{*(2)}$, where $n \in \{1, 2, ..., 6\}$, and $W_V^{*(2)}$ for the actors and the critic respectively. Their elements are a concatenation of the learnable parameters $W_n^{(2)}$ and $W_V^{(2)}$ with the frozen weight matrices from column one $W_n^{(1)}$ and $W_V^{(1)}$. As aforementioned, with this implementation, we can initialize the second-column output with the policies and state-value function from the first column and guide the student agent during the first steps before the first network update.

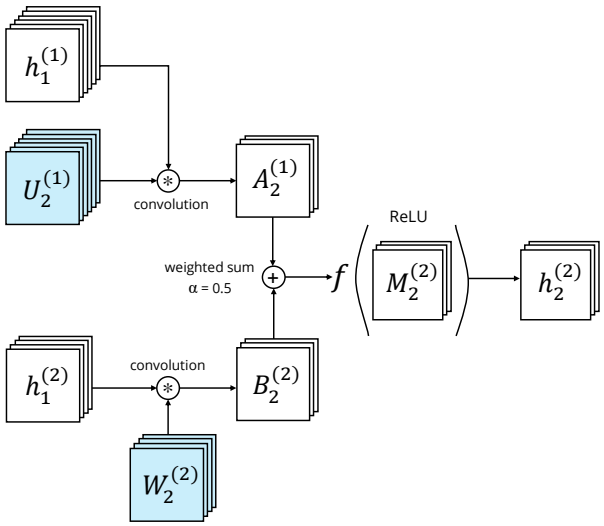


Figure 6.3. CNN-CNN lateral connection. $U_2^{(1)}$ is the learnable tensor from the lateral connection. After the weighted sum, a ReLU function f introduces a non-linearity. The blue blocks are learnable parameters.

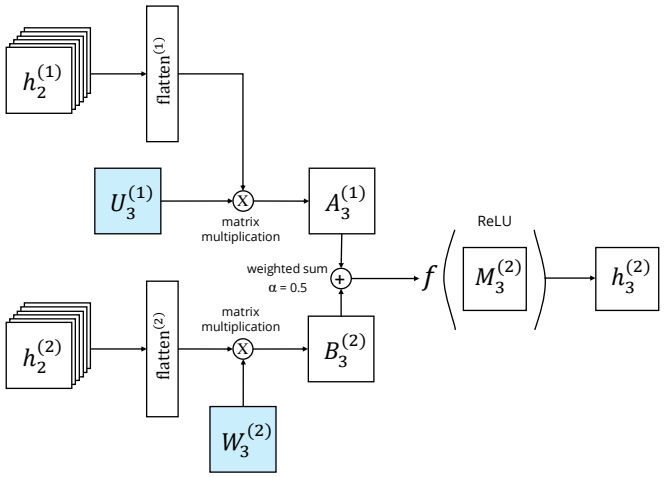


Figure 6.4. CNN-FC lateral connection. $U_3^{(1)}$ is the learnable tensor from the lateral connection. After the weighted sum, a ReLU function f introduces a non-linearity. The blue blocks are learnable parameters.

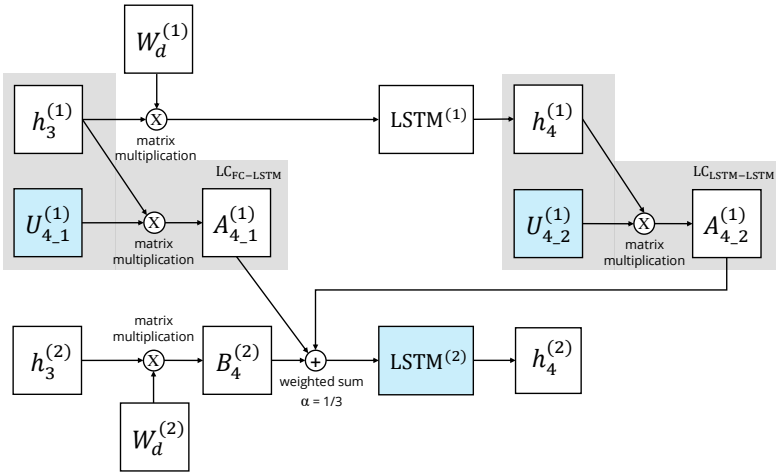


Figure 6.5. LSTM lateral connections. The shadowed areas correspond to the lateral connection that comes from the first-column FC layer and from the first-column LSTM layer. $U_{4\_1}^{(1)}$ and $U_{4\_2}^{(1)}$ are the learnable tensors from these lateral connections. After the weighted sum, a ReLU function f introduces a non-linearity. The blue blocks are learnable parameters.

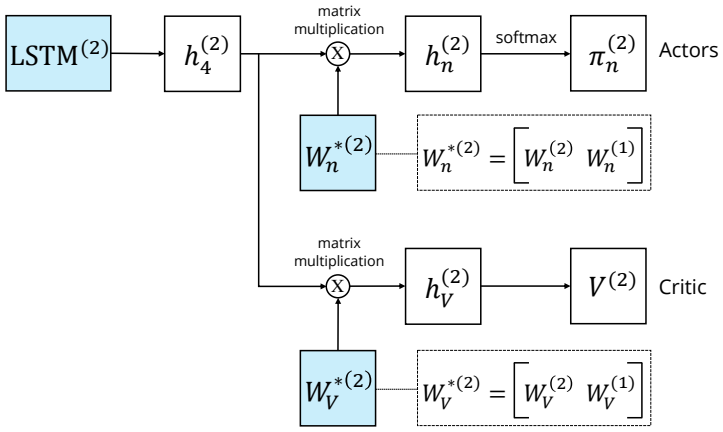


Figure 6.6. LSTM-FC lateral connection. This lateral connection is implemented as a concatenation of the weights from the first-column output layers. For the problem solved, $n \in \{1, 2, ..., 6\}$ as there is one policy per robot joint. The blue blocks are learnable parameters.

Finally, Table 6.1 presents the parameter breakdown per layer, including the lateral connections. For the CNN blocks, the expression corresponds to (number of filters $\times$ depth $\times$ width $\times$ height) + bias; for the FC blocks, it is (input size $\times$ output size) + bias; the LSTM has four gates, and their parameters are multiplied by two as the number of hidden and cell states is the same. Recall that during the second-column training, parameters from the first-column remain frozen.

Table 6.1. PNN trainable parameters per column and layer. Each row corresponds to one blue block from Figure 6.2 For the CNN layers, the information is given as (number of filters $\times$ depth $\times$ width $\times$ height) $+$ bias, whereas it follows (input size $\times$ output size) $+$ bias in the FC blocks. The LSTM has 4 gates.

	First column (Teacher)	**Second column (Student)**
$\text{FC}_V^{(2)}$	$(128 \times 1) + 1$	$(144 \times 1) + 1$
$\text{FC}_n^{(2)}$	$6 \times [(128 \times 7) + 7]$	$6 \times [(144 \times 7) + 7]$
$\text{LSTM}^{(2)}$	$2 \times (512 \times 128) + 2 \times 512$	$2 \times (64 \times 16) + 2 \times 64$
$\text{LC}_{LSTM-LSTM}$	-	16×128
$\text{LC}_{FC-LSTM}$	-	16×128
$\text{FC}^{(2)}$	$(128 \times 1152) + 128$	$(16 \times 288) + 16$
LC_{CNN-FC}	-	16×1152
$\text{CNN}_2^{(2)}$	$(32 \times 16 \times 5 \times 5) + 32$	$(8 \times 8 \times 5 \times 5) + 8$
$\text{LC}_{CNN-CNN}$	-	$(8 \times 16 \times 6 \times 6)$
$\text{CNN}_1^{(2)}$	$(16 \times 3 \times 8 \times 8) + 16$	$(8 \times 3 \times 8 \times 8) + 8$
Total	**301,147**	**43,323**

6.2. Objectives and contributions

PNNs are a promising technique to bridge the reality gap. Their approach through a few-shot transfer not only allows them to address the sim-to-real problem but also to leverage the experience of one or more agents that can be trained in related environments or tasks to improve the efficiency of the new agent's learning process. This is especially interesting because merging them with DR would allow applying high-level DR to a contained number of features and then combine their knowledge to prevent the performance drops encountered in Chapter 5 when randomizing all the features at the same time. try to randomized all features at the same time, the performance drops drastically.

After proving that, in terms of PNNs, a first-column agent performance can be enhanced through DR with the same training effort, in this chapter, we analyze how the agent's yield can be affected when the environment presents demanding conditions. The result will let us address our first objectives, which are understanding the PNNs learning mechanism, validating our PNN architecture implementation, and assessing the PNN agents forgetting after learning in the new setting. Thereafter, the following chapter's objective is to face our sim-to-real problem implementing the PNN architecture with a single teacher and student column.

The main contributions of this chapter are:

- The quantification of the robustness of the first-column agent when it is exposed to specific high-level changes in visual features.
- The evaluation of the PNN-agents learning dynamics when fine-tuned for environments that present different difficulty levels.
- The analysis and assessment of the forgetting of the second column agent.
- The implementation, analysis, and evaluation of a methodology based in the PNNs architecture to address the sim-to-real challenge under an industrial setting. Besides, the learning algorithm used when learning in the real environment is an original serialized A3C.
- The validation of using Augmented Reality (AR) to efficiently train and evaluate agents in real environments, preserving and even improving its performance and generalization ability when it faces real targets.

6.3. Sim-to-sim: Analyzing PNNs learning mechanisms

6.3.1. Method

Figure 6.7 illustrates the methodology followed for the understanding of PNNs and the evaluation of the second column's forgetting. We start with the design of a set of experiments to define which high-level visual changes in the environment involve a drop in the agent performance. In these tests, we experiment with changes in the background, the robot, and the target, which are the key players in the scene. After selecting three environments based on the agent's perceived difficulty (i.e., easy, intermediate, and complex), we train and evaluate three PNN-like agents to assess the knowledge transfer between the teacher and the student that develops the same task under a different setup. Lastly, we evaluate the forgetting of the best model of each PNN with respect to the baseline environment. These experiments are carried out with a sim-to-sim approach to control all the drivers that might lead to learning inefficiencies or noise in the insights obtained.

6.3.1.1. Procedure to determine the adversarial environments

To study the PNN limitations and evaluate their robustness and forgetting, we defined virtual adversarial environments that let us simulate the deployment of the acquired synthetic experience into a similar setup. We define adversarial environments as those that lead to a performance lower than 90 % in the BM post-training evaluation. With these outcomes, we can establish an evaluation benchmark to compare the results we will gather with the PNN-like agents.

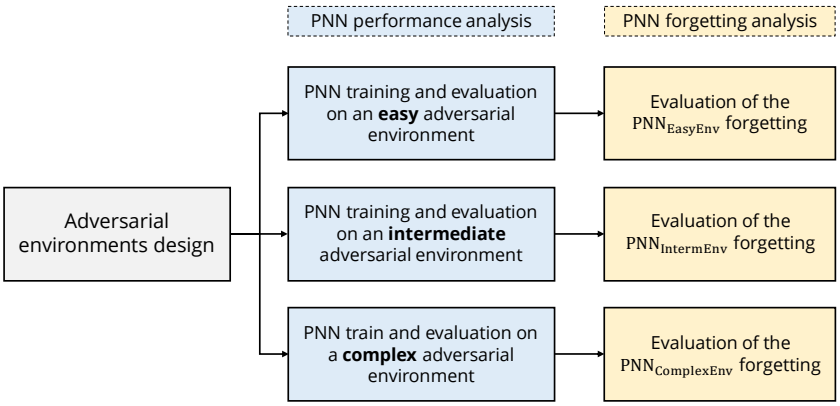


Figure 6.7. Sim-to-sim method diagram. Starting from the design of the adversarial environments, three PNN-like agents are trained and evaluated in terms of their performance, and their forgetting with respect to the teachers' task.

Figure 6.8 shows the procedure utilized to define the adversarial environment where the student agent will be trained. It begins with the BM post-training evaluation under scenarios where only one feature changes. Based on these results, we also evaluate the BM performance in scenarios where multiple characteristics vary. We conclude with the selection of three environments that the agent sees as easy, intermediate, or complex according to its success rate on each of them.

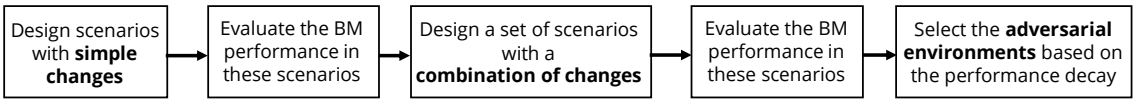


Figure 6.8. Method followed to design the adversarial environments. It is based on the performance analysis of the BM when visual changes are introduced in the environment.

We use the accuracy as the main metric to measure the performance of the agent, defined in (4.9) as the percentage of episodes in which the agent reached the goal. The rest of the post-training evaluation hyperparameters (i.e., evaluations across 1,000 episodes with random initial robot and target poses), and the metrics described in Section 4.2.1 are also used to complement the evaluation.

As one of the objectives is to understand the PNN learning mechanism, with a special focus on its state representation learning capabilities, we complement the previous quantitative analysis provided by the metrics with a qualitative perspective that attempts to explain the rationale followed by the agent in its decisions. This is made through the use of saliency maps [KB01]. Saliency methods are commonly used in image-based problems to highlight areas of the image where the convolutional layers' activation maps are more active. Basically, having a high saliency in a certain location means that variations in that part will have a large impact on the output. In our DRL setup, this means that we can distinguish those regions where the agent is paying more attention. We decided to use saliency maps because of their direct implementation and due to the fact that they pass the sanity checks proposed in [Ade+18]. Equation (6.3) defines how saliencies are calculated:

$$M_{ij} = \max_k \left| \frac{\partial \sum \text{score}(I)}{\partial I_{kij}} \right| \tag{6.3}$$

where M_{ij} is the saliency map for the pixel in row i, column j, and channel k of the input image I (the state observation), and score(I) output of the last CNN layer. The score that corresponds

to the output of the last CNN layer is aggregated, and then it is partially derived with respect to each pixel, identifying the pixels with higher gradient values per channel. Finally, the outcomes are normalized to reduce the variance.

6.3.1.2. PNN training and post-training evaluation procedure

The algorithm used to train the PNNs is the A3C algorithm already described in Section 4.1, with the difference that, since the second-column agent reuses experience from the first-column agent, the training process lasts 30 M steps instead of the 70 M required by the BM with torque control. The intermediate layers and lateral connections start with a random orthogonal initialization, while the output blocks and links are set to follow the first-column policy during the first episode, as proposed in [Rus+17].

The interim evaluation remains the same as defined in Section 4.2.1. Shortly, every 50 k steps we test the last shared model across 40 episodes with fixed initial robot and target poses and a 5 cm rewarding distance. After training, we perform a post-training evaluation of the best model where we assess it in 1,000 episodes with random initial configurations and a 10 cm rewarding distance.

6.3.1.3. PNN forgetting evaluation procedure

To evaluate the forgetting the student agent suffers with respect to the teacher's task, we assess its performance in the first-column environment with a post-training evaluation. Our goal with this experiment is to determine how much the student can remember about the teacher's task. The metrics employed during the forgetting assessment are the same as those employed in the post-training evaluation.

6.3.2. Results and discussion

6.3.2.1. Selection of the adversarial environments

Due to the visual nature of the task solved, it is key to analyze in detail the performance of the PNNs state representation layers, as they should be robust enough to overcome the discrepancies with the real setup. With this idea and the insights gained with the DR experiments, we start the selection of the easy, intermediate, and complex setups with the design of the adversarial environments. To do so, we performed color variations on 1) the background, which can be split into the ground and the sky; 2) the robot, which can be broken per joint; 3) the gripper, which is made of the fingers and a connector that joins the fingers with the last robot joint; and 4) the target. Table 6.2 and Table 6.3 gather the environment changes that define the scenarios where the BM is tested to define which ones can be classified as adversarial environments, and thus, can be a candidate to train the second-column agent. In these experiments, the BM is the agent with the camera pose randomized trained in Section 5.3.3.1 due to its great results.

Table 6.4 presents the post-training evaluation outcomes of the BM for Table 6.2 and Table 6.3 setups categorized as adversarial environments (i.e., the BM presents a post-training accuracy below 90 %). When the background is modified, either one or its two elements, the agent performance decreases between 30 % to 40 %. If changes are applied to the joints closer to the gripper or to the gripper itself, the reduction ranges from 20 % to 30 %, while when they apply to the robot color, the drop scales up to 45 %. Modifications in the target color result in a decrease of 20 % to 25 % if the cube is black or white, but when it is blue, the agent success rate plummets to 50 %.

Table 6.2. Environments designed and evaluated with single visual changes. Each row corresponds to an environment where the agent was evaluated. If no change is specified, the color remains as in the BM. The RGB code supplements the color name.

		Background		Gripper			
Environment	**Changes**	**Ground**	**Sky**	**Robot link n**	**Fingers**	**Connection part**	**Target**
BM	–	Green (0.19, 0.30, 0.23)	Gray (0.67, 0.71, 0.75)	$n \in [1,5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	Red (1.0, 0.0, 0.0)
Env_{SC1}	Ground	Black (0.0, 0.0, 0.0)	–	–	–	–	–
Env_{SC2}		White (1.0, 1.0, 1.0)	–	–	–	–	–
Env_{SC3}	Sky	–	Black (0.0, 0.0, 0.0)	–	–	–	–
Env_{SC4}		–	White (1.0, 1.0, 1.0)	–	–	–	–
Env_{SC5}	Robot	–	–	$n \in [1,5]$ Black (0.0, 0.0, 0.0)	–	–	–
Env_{SC6}		–	–	$n \in [1,5]$ White (1.0, 1.0, 1.0)	–	–	–
Env_{SC7}		–	–	$n \in [1,6]$ Black (0.0, 0.0, 0.0)	–	–	–
Env_{SC8}		–	–	$n \in [1,6]$ White (1.0, 1.0, 1.0)	–	–	–
Env_{SC9}		–	–	$n \in [1,6]$ Red (1.0, 0.0, 0.0)	–	–	–
Env_{SC10}		–	–	$n \in [1,6]$ Blue (0.0, 0.0, 1.0)	–	–	–
Env_{SC11}	Fingers changes	–	–	–	Black (0.0, 0.0, 0.0)	–	–
Env_{SC12}		–	–	–	Red (1.0, 0.0, 0.0)	–	–
Env_{SC13}	Target	–	–	–	–	–	Black (0.0, 0.0, 0.0)
Env_{SC14}		–	–	–	–	–	White (1.0, 1.0, 1.0)
Env_{SC15}		–	–	–	–	–	Blue (0.0, 0.0, 1.0)

Table 6.3. Environments designed and evaluated with multiple visual changes. Each row corresponds to an environment where the agent was evaluated. If no change is specified, the color remains as in the BM. The RGB code supplements the color name.

		Background		Gripper			
Environment	**Changes**	**Ground**	**Sky**	**Robot link n**	**Fingers**	**Connection part**	**Target**
BM	–	Green (0.19, 0.30, 0.23)	Gray (0.67, 0.71, 0.75)	$n \in [1, 5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	Red (1.0, 0.0, 0.0)
Env_{MC1}	Gripper	–	–	–	Green (0.19, 0.30, 0.23)	Green (0.19, 0.30, 0.23)	–
Env_{MC2}		–	–	–	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–
Env_{MC3}		–	–	–	White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	–
Env_{MC4}	Background	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–	–	–	–
Env_{MC5}		White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	–	–	–	–
Env_{MC6}		Blue (0.0, 0.0, 1.0)	Blue (0.0, 0.0, 1.0)	–	–	–	–
Env_{MC7}		Green (0.19, 0.30, 0.23)	Green (0.19, 0.30, 0.23)	–	–	–	–
Env_{MC8}		Gray (0.67, 0.71, 0.75)	Gray (0.67, 0.71, 0.75)	–	–	–	–
Env_{MC9}	Background & gripper	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–
Env_{MC10}		Blue (0.0, 0.0, 1.0)	Blue (0.0, 0.0, 1.0)	–	Blue (0.0, 0.0, 1.0)	Blue (0.0, 0.0, 1.0)	–
Env_{MC11}	Sky & gripper	–	Black (0.0, 0.0, 0.0)	–	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–
Env_{MC12}		–	White (1.0, 1.0, 1.0)	–	White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	–
Env_{MC13}	Robot	–	–	$n \in [3, 4]$ Black (0.0, 0.0, 0.0)	–	–	–
Env_{MC14}		–	–	$n \in [3, 4]$ White (1.0, 1.0, 1.0)	–	–	–
Env_{MC15}	Robot & Gripper	–	–	$n \in [4, 5]$ Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	Black (0.0, 0.0, 0.0)	–
Env_{MC16}		–	–	$n \in [4, 5]$ White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	White (1.0, 1.0, 1.0)	–

Table 6.4. Outcomes from the evaluation of the candidate adversarial environments. Only scenarios with an accuracy lower than 90 % are considered. The final selected adversarial environments are highlighted in bold and shadowed.

Environment	Changes	Mean return	std. dev. return	Mean episode length	std. dev. episode length	Mean failure distance (cm)	std. dev. failure distance	Max failure distance (cm)	Accuracy (%)
BM	–	55.65	0.79	25.73	0.64	0.00	0.00	0.00	**100.0**
Env_{SC2}	White ground	25.26	5.90	36.62	1.70	0.15	0.02	0.20	60.7
Env_{SC3}	Black sky	28.31	5.82	37.01	1.72	0.18	0.02	0.22	64.5
Env_{SC7}	Black robot	21.11	6.14	38.17	1.77	0.23	0.02	0.28	54.4
Env_{SC8}	White robot	19.59	5.24	38.91	1.52	0.23	0.02	0.27	54.0
Env_{SC10}	Blue robot	18.19	5.88	38.82	1.72	0.24	0.02	0.29	51.7
Env_{SC13}	Black target	35.21	4.76	34.95	1.57	0.18	0.02	0.22	72.9
Env_{SC14}	**White target**	39.20	4.78	32.85	1.30	0.20	0.03	0.26	**78.8**
Env_{SC15}	**Blue target**	17.43	6.18	39.67	1.63	0.20	0.01	0.22	**49.7**
Env_{MC4}	Black background	37.76	4.87	32.82	1.71	0.16	0.02	0.19	76.2
Env_{MC5}	**White background**	32.58	5.12	34.34	1.39	0.18	0.02	0.22	**69.8**
Env_{MC6}	Blue background	27.54	5.74	35.82	1.63	0.27	0.04	0.34	65.1
Env_{MC9}	Black background & gripper	37.18	5.63	33.15	1.71	0.16	0.03	0.27	75.8
Env_{MC10}	Blue background & gripper	29.36	5.29	35.20	1.64	0.27	0.03	0.32	67.8
Env_{MC11}	Black sky & gripper	24.80	5.04	38.08	1.39	0.19	0.01	0.22	60.0
Env_{MC13}	Black links 3 & 4	45.21	4.42	31.25	1.42	0.21	0.04	0.30	86.4
Env_{MC14}	White links 3 & 4	35.96	4.29	34.40	1.51	0.21	0.02	0.28	73.5
Env_{MC15}	Black links 4 & 5 & gripper	39.34	4.59	33.16	1.37	0.20	0.04	0.27	78.2

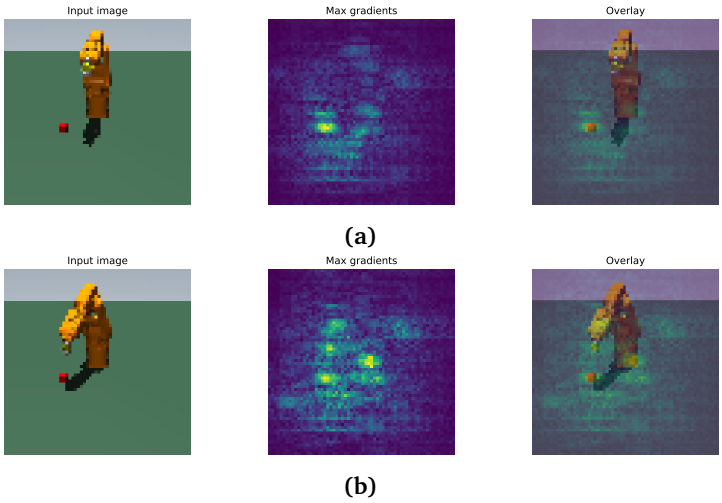


Figure 6.9. Environment observations and saliency maps in the episode beginning (a) and in an intermediate step (b) for the BM agent under its original scenario. he gradient values indicate that the agent's attention is focused on the target and the robot, with yellow representing high absolute values and blue indicating the lowest.

To investigate the reasons behind these results, Figure 6.9 presents the saliency map for the BM in a random initial position (a) and in an intermediate step (b). The scale of the saliency maps ranges from yellow for the highest absolute values to blue for the lowest, with green representing intermediate values. Based on the value of the gradients, we can state that at the

episode beginning the agent's attention is mainly on the target, the robot base, and the shadow, whereas in (b), it continues focusing on the target and the robot base, and also takes the last joints and the gripper into account. This suggests that the BM agent is able to identify the main environment elements that convey meaningful information to solve the problem. Nevertheless, some dispersely distributed ground pixels also seem to influence the policy.

Analyzing the BM accuracy drops, the decrease when the ground changes is larger than when the last joints or the gripper are modified could be due to its attention area being wider than the pixels that belong to the latter elements. However, when the success rate reduction is due to variations in the robot or target colors, the saliency maps should be interpreted based on the RGB channels. Since the robot pixels are orange (1.0, 0.5, 0.0), the effect of the blue channel on the CNN activation maps is insignificant. Hence, the lowest performance happens when the robot is entirely blue. The target follows the same rationale. As it is red (1.0, 0.0, 0.0), neither the green nor the blue channels influence the CNN activation maps, so any change in that area involves a partial loss of its perception capacity. Besides, we suggest that this result may be enhanced with the blue color because of the poor contrast between the cube, the dark green ground, and the robot shadow.

According to Table 6.4 and the analysis of the saliency maps, the three adversarial environments selected to put PNNs to the test were: 1) the easiest setup, which has the white target and where the BM has 78.8 % accuracy; 2) an intermediate setup which has a completely white background and where the BM succeeds 69.8 % of the times; and 3) the most challenging adversarial environment with the blue target, where the BM can only achieve an accuracy of 49.7 %. The intermediate scenario is especially interesting because it can be regarded as if a mask (i.e., the elimination of all the background keeping just the main scenario elements), had been applied to the environment. This is a technique that can be potentially used in the sim-to-real problem to avoid the interference of environmental noise and background variations. Table 6.5 summarizes the configuration of the adversarial environments selected.

Table 6.5. Adversarial environments selected. The RGB code supplements the color name. Features highlighted in bold are the changes with respect to the BM.

	Background		Gripper				
	Ground	Sky	Robot link n	Fingers	Connection part	Target	Accuracy (%)
Env_{Easy}	Green (0.19, 0.30, 0.23)	Gray (0.67, 0.71, 0.75)	$n \in [1,5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	**White (1.0, 1.0, 1.0)**	78.8
$\text{Env}_{\text{Interm}}$	**White (1.0, 1.0, 1.0)**	**White (1.0, 1.0, 1.0)**	$n \in [1,5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	Red (1.0, 0.0, 0.0)	69.6
$\text{Env}_{\text{Complex}}$	Green (0.19, 0.30, 0.23)	Gray (0.67, 0.71, 0.75)	$n \in [1,5]$ Orange (1.0, 0.5, 0) $n = 6$ Black (0.0, 0.0, 0.0)	White (1.0, 1.0, 1.0)	Yellow (1.0, 1.0, 0.0)	**Blue (0.0, 0.0, 1.0)**	49.7

Figure 6.10 presents the BM saliency maps for the adversarial environments in the same initial pose as Figure 6.9. The gradients in Figure 6.10 suggest that while in (b), the BM agent almost does not notice the target and it is only focused on part of the robot's shadow, in (c), it is slightly more focused on the robot's base and the complete shadow.

These analyses confirm the importance of the CNN blocks of the PNN architecture, which seem to embed the state representation. However, the CNN layers do not appear to be capable of efficiently dealing with visual modifications in the environment. Hence, for a successful sim-to-real process, the virtual and real environment must be visually similar, as stated in [Rus+17], or during the training phase of the teacher agent, the virtual setup should be sufficiently randomized. Otherwise, as Table 6.5 results evidence, the PNN student will require a large amount of real experience to fine-tune its knowledge.

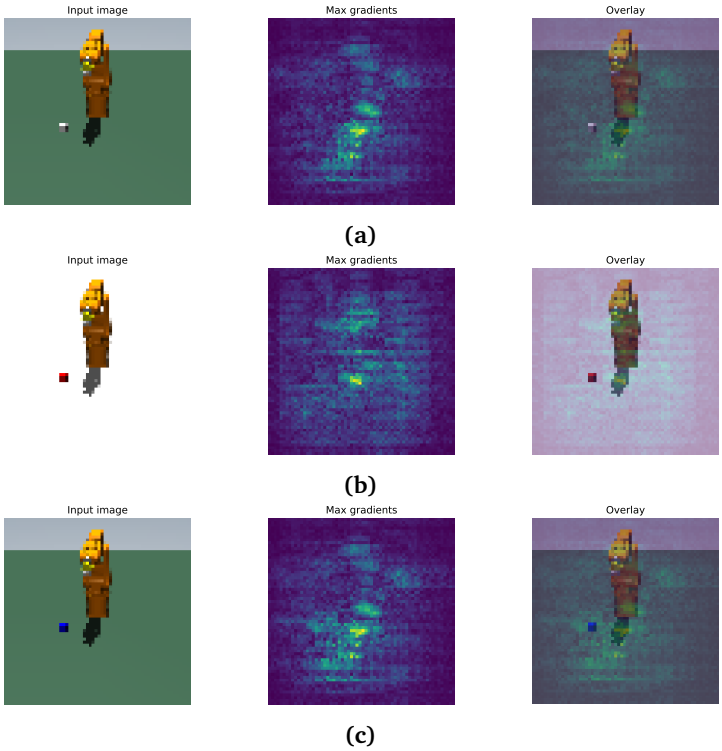


Figure 6.10. Environment observations and saliency maps from the same initial poses as in Figure 6.9 for BM in the easy (a), the intermediate (b), and the complex (c) adversarial environments. Note that the agent's attention is neither on the robot nor on the target; it is mainly in the robot's shadow.

6.3.2.2. PNN agents results in the adversarial environments

Figure 6.11 presents the training curves for the BM (orange) and the three student agents: the $\text{PNN}_{\text{EasyEnv}}$ (blue), where the target is white, the $\text{PNN}_{\text{IntermEnv}}$ (green), where the background is white, and the $\text{PNN}_{\text{ComplexEnv}}$ (purple), where the target is blue. Although the BM is the first column of the trained PNN agents, its training curve is also plotted to compare the processes and determine if the student agents are using the shared knowledge to enhance their training times and performance. To ease the comparison, we only present the first 30 M steps (i.e., the PNN agents training time) of the BM training. The three agents outperform the BM from the start, thanks to the architecture's initialization, which allows them to begin with the first-column policy. The $\text{PNN}_{\text{EasyEnv}}$ trendline hardly increases up to an average return of almost 50. This rapid reach of the steady state, about three times faster than the BM, may be attributed to the knowledge transfer from the BM, as well as the simplicity of the environment, which resulted

in only a 20 % performance degradation from the teacher. Conversely, the $PNN_{IntermEnv}$ agent reaches the steady state with the same average return as the $PNN_{EasyEnv}$ around 3 M steps after it. Lastly, the $PNN_{ComplexEnv}$ agent shows the most challenging training process. Although it initially surpasses the BM in performance, after 7 M steps, the agent plateaus at an average return of only 15, indicating that it struggles with this more visually distinct and difficult environment. The agent does not improve the performance of the BM later, suggesting that further architectural changes or deeper feature extraction are needed to handle such complex scenarios effectively.

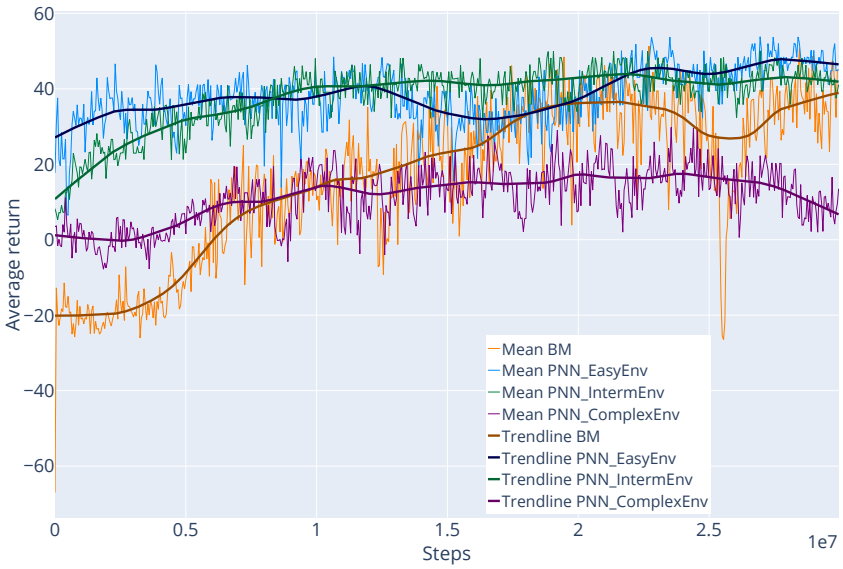


Figure 6.11. Average returns in the interim evaluation for $PNN_{EasyEnv}$ (blue), the $PNN_{IntermEnv}$ (green), the $PNN_{ComplexEnv}$ (purple), and the BM (orange) agents over 30 M steps (note the 1e7 scaling factor on the right-hand corner of the figure). The BM is plotted until its 30^{th} M step to compare models. While the $PNN_{EasyEnv}$ and $PNN_{IntermEnv}$ agents attain better results than the BM, the $PNN_{ComplexEnv}$ is not able to outperform it.

Table 6.6 summarizes the key outcomes of the evaluations for each agent. The $PNN_{EasyEnv}$ agent achieves 99.0 % accuracy at step 1.6 M, requiring minimal additional steps to improve the initial 78.8 %. This demonstrates that the agent quickly learns the task new environment with the transferred knowledge. Similarly, the $PNN_{IntermEnv}$ agent achieves 94.7 % accuracy after 5.65 M steps. Although this agent requires more training steps than $PNN_{EasyEnv}$ due to the increase in the environment complexity, it still significantly improves the BM performance in this setup, requiring little experience. Since the differences rely on visual features, the learning difficulties should be on the model's state representation blocks (i.e., the CNN blocks), which confirms the assumption made in Section 6.3.2.1 with the saliency map results. In contrast, the $PNN_{ComplexEnv}$ agent struggles more with its complex environment, achieving a maximum accuracy of 85.6 % at 10.65 M steps. After this point, the performance remains approximately the same. This is likely due to limitations in the state representation layers, which make it difficult to obtain a good state approximation when visual discrepancies lead to a significant deterioration in the teacher's performance.

Finally, Table 6.7 presents the forgetting assessment results for the three PNN agents. This evaluation measures the agents' ability to retain knowledge from the BM environment, even after learning in a different environment. The results indicate that the $PNN_{EasyEnv}$ and $PNN_{IntermEnv}$ agents suffer from partial forgetting when evaluated in the original BM environment. The $PNN_{EasyEnv}$ agent retains 83.7 % accuracy, while the $PNN_{IntermEnv}$ agent shows a similar trend,

Table 6.6. $PNN_{EasyEnv}$ agent, $PNN_{InterimEnv}$ agent, and $PNN_{ComplexEnv}$ agent results in the post-training evaluation. The evaluation interval is not exactly 50 k because a global step counter is shared among all the instantiated agents.

Agent	Evaluation step	Mean return	std. dev. return	Mean episode length	std. dev. episode length	Mean failure distance (cm)	std. dev. failure distance (cm)	Max failure distance (cm)	Accuracy (%)
$PNN_{EasyEnv}$	0	39.20	4.78	32.85	1.30	0.20	0.03	0.26	78.8
	1,600,031	54.32	9.22	27.60	5.46	0.27	0.10	0.41	99.0
$PNN_{InterimEnv}$	0	32.58	5.12	34.34	1.39	0.18	0.02	0.22	69.8
	5,650,232	50.85	18.08	29.49	7.32	0.17	0.04	0.30	94.7
$PNN_{ComplexEnv}$	0	17.43	6.18	39.67	1.63	0.20	0.01	0.22	49.7
	10,650,675	44.59	25.92	33.09	9.80	0.21	0.11	0.50	85.6

with an accuracy of 80.9 %. This suggests that while they learn the new tasks, some of the previously acquired knowledge from the BM environment is lost, likely due to *overfitting* on the second task. Interestingly, the $PNN_{ComplexEnv}$ agent retains the highest level of performance from the original BM environment, achieving 99.8 % accuracy when evaluated in the red target scenario. This suggests that, while the agent struggles to master the complex environment, it is still able to retain and utilize knowledge from the simpler BM environment, possibly due to the dominance of the lateral connections over the second-column learned weights.

Table 6.7. Forgetting assessment for the $PNN_{EasyEnv}$ agent, the $PNN_{InterimEnv}$ agent and the $PNN_{ComplexEnv}$ agent. The students are evaluated on the teacher's task.

Agent	Training environment	Evaluated environment	Mean return	std. dev. return	Mean episode length	std. dev. episode length	Mean failure distance (cm)	std. dev. failure distance	Max failure distance (cm)	Accuracy (%)
$PNN_{EasyEnv}$	White target	White target	54.32	9.22	27.60	5.46	0.27	0.10	0.41	99.0
	White target	Red target	44.19	26.57	34.84	9.46	0.18	0.10	0.55	83.7
$PNN_{InterimEnv}$	White ground and sky	White ground and sky	50.85	18.08	29.49	7.32	0.17	0.04	0.30	94.7
	White ground and sky	Green ground and gray sky	40.82	29.65	32.24	9.62	0.21	0.05	0.35	80.9
$PNN_{ComplexEnv}$	Blue target	Blue target	44.59	25.92	33.09	9.80	0.21	0.11	0.50	85.6
	Blue target	Red target	54.88	5.65	26.54	3.95	0.21	0.05	0.27	99,8

6.4. Sim-to-real: Bridging the reality gap with PNNs

6.4.1. Method

The methodology followed to solve the sim-to-real gap with our PNN implementation starts by deciding which model should be the teacher, selecting among the BM or the models trained with noisy observations in the virtual environment (please refer to Section 4.2 and Section 5.4 for more details on each model). Then we train the PNN-like agent in the real environment following a few-shot attempt. We conclude with two post-training evaluations, one with the ArUco markers and the other with real objects, to assess the agent's robustness and its generalization capability.

6.4.1.1. PNN training and post-training evaluation procedure

One of the major changes between the sim-to-sim and the sim-to-real approaches in terms of their deployment is that we had to modify the A3C implementation. In the real setup, we cannot instantiate several environments and agents to parallelize the learning process. Instead of using an Advantage Actor Critic (A2C) which, compared to the A3C, has a slower convergence rate and might present some instabilities, we developed a serialized A3C that should reduce the agent's variance. Even though it has only one process, our serialized A3C has two networks, one that represents the network instantiated by an A3C agent, the "behavior" network, and a "target" network that has the role of the A3C global network shared among the workers. The behavior network determines the policy in each step, and it is only updated by the target network after ten episodes. At the end of each episode, the behavior model gradients are obtained and transferred along its weights to the target model. Algorithm 6.1 details the process. The hyperparameters and training conditions were the same as in the sim-to-sim approach except for the number of training steps and the interval among interim evaluations. As a few-shot approach, the training steps are considerably reduced to 300 k steps, assuming that the agent will have reached a steady state by this time. Additionally, the interim evaluation is carried out every 2.5 k steps instead of 50 k like in the A3C.

The post-training evaluation procedure is more exhaustive than the assessments performed in the virtual environment, as we noticed that the accuracy depends heavily on the targets sampled. To avoid this bias in the results, we evaluated the trained agent five times per ArUco position using different initial robot poses. As a result, we can build a heat map with the accuracy percentages achieved in each target placement. We interpolated among its known nearest points to complete the heat map in positions where no ArUco tags were saved. Although the agent was trained within the [–0.2, 0.2] m interval for the y-axis, we evaluated it in the complete workspace area, which stretches within the [–0.3, 0.3] m interval. Additionally, we conducted a thorough analysis of the gripper trajectory to the target with the objective of understanding the agent's behavior depending on where the target is placed.

To conclude, after the post-training evaluation with the ArUco targets, we evaluate the agent performance with real objects, i.e., a red, a yellow, and a blue LEGO® cubes, and a red and white mug (Figure 3.6). We first assessed the red LEGO® cube as it was the most similar to the AR targets. We placed it in ten different positions that were not seen by the agent before. As we did previously, we obtained the accuracy by weighting five times the model performance in each goal position with random initial robot poses. According to the results with the red cube, we examine the agent's generalization capability with the other targets. However, this time we selected only those target positions where the agent achieved 100 % accuracy with the red LEGO® cube.

6.4.2. Description of the experiments

We carried out several training attempts (Table 6.8). Models M1 and M2 are trained with all the ArUco positions used during the virtual agent training. The first column of M1 corresponds to the virtual agent with noisy observations, and its second column is initialized according to Section 6.1.1. The M2 teacher is the BM agent, and its student weights are initially set with a pre-trained model exposed to noisy observations in the virtual environment. On the other hand, M3 and M4 were trained with a reduced ArUco area, as we observed in M1 and M2 that the agent struggled to reach the target positions close to the workspace limit. While the first

Algorithm 6.1. Pseudo-code for the serialized A3C used to train the PNN-like student with real experience. We use a target model, `model_t`, and behavior model, `model_b`, which are updated synchronously but following the A3C rationale to reduce the variance. `T` is the steps counter, and `T_max` is the maximum number of training steps. `interval_update` represents the number of episodes between the updates of the models, and `evaluation_interval` the steps between interim evaluations.

Input: `T_max`, `max_episode_length`, `interval_update`, `evaluation_interval`
Initialize `T`, `episode_counter`, `test_steps`;
Initialize `model_t` and `model_b`;
Initialize the RMSprop;
`model_b` $\leftarrow$ `model_t`;

while `T.value()` $\leq$ `args.T_max - 1` **do**
    if done **then**
        Reset `episode_length`;
        Obtain s_t;
        `done = False`;
    end
    while not done **do**
        $s_t = s_{t+1}$;
        π, V = `model_b`(s_t);
        Sample actions a_n from π using a greedy strategy;
        s_{t+1}, r_{t+1}, `episode_succeed` = `step`(a_n);
        `T += 1`;
        `test_steps += 1`;
        if `episode_succeed` or `episode_length` $\geq$ `max_episode_length` **then**
            `done = True`;
            `episode_counter += 1`;
            Break;
        end
    end
    Backpropagate and update `model_b`;
    `model_t gradients` $\leftarrow$ `model_b gradients`;
    if `episode_counter == interval_update` **then**
        `model_b` $\leftarrow$ `model_t`;
        `episode_counter = 0`;
    end
    if `test_steps` $\geq$ `evaluation_interval` **then**
        `test(model_b)`;
        `test_steps = 0`;
    end
end

column of M3 is the agent trained under noisy observations, in M4, it is the BM. The second columns are initialized for both models as explained in Section 6.1.1.

Table 6.8. PNNs-like agents designed to attempt the sim-to-real transfer.

Model name	Teacher column model	Target area (m)	Weight initialization
M1	Noisy observation	ArUco markers $x \sim (0.2, 0.4)$ $y \sim (-0.3, 0.3)$	Random
M2	Raw observation	ArUco markers $x \sim (0.2, 0.4)$ $y \sim (-0.3, 0.3)$	Student column virtually trained with noise
M3	Noisy observation	ArUco markers $x \sim (0.2, 0.4)$ $y \sim (-0.2, 0.2)$	Random
M4	Raw observation	ArUco markers $x \sim (0.2, 0.4)$ $y \sim (-0.2, 0.2)$	Random

6.4.3. Results and discussion

Due to the time required and the complexity of the post-training process, we made a preliminary selection of which setup gave the best result during the training process. The decision was made considering whether the agent was learning or not and based on the best accuracy percentage in the interim evaluation. Figure 6.12 presents the models' training curves and Table 6.9 gathers the accuracy results of the PNN models during their interim evaluation. M1 and M2 obtained 50 % and 55 % success rates on their best interim evaluations, which is above the results achieved by their respective first-column agents, 34.1 % and 15.8 % respectively. However, their training curves do not show a clear and rapid learning transitory characteristic of a few-shot approach, but on the contrary, they seem stuck. For this reason, we stop the training before the 300 k steps. M3 improves the previous outcomes reaching 72.5 % accuracy. However, as M1 and M2, the M3 learning process moves in a semi-steady state with no signs of improvement. Finally, M4 dramatically outperforms the previous results. The best interim evaluation reaches 95 % in less than 63 k steps, which is on a par with the state-of-the-art results presented in [Rus+17]. Besides, its training curve suggests that the agent is using the first column of the PNN that has around 15 % accuracy in the initial step.

Interestingly, the M4 agent with the BM as teacher has better results than M3, whose teacher is the agent that integrated low-level DR during its training with the noisy observations and that performed better the zero-shot transfer. We argue that this might be because, contrary to what we originally thought, if both setups are very similar the model's weights might not have a sufficient incentive to change correctly.

After selecting the best model, we perform the post-training evaluation in each of the ArUco positions saved. Figure 6.13 presents the heatmap with the accuracy percentages obtained. The agent accomplishes the task perfectly in most of the central area, exhibiting accuracies above 80 % in the worst cases. We noticed that in this zone, there is a decrease in the accuracy when the cube appears close to the robot's base. In the 2D image, the pixels occupied by the robot's base and the target overlap, which suggests that the filters of the CNN layers may be activating in the same area, hindering the agent's ability to perceive what the robot is and where the target is.

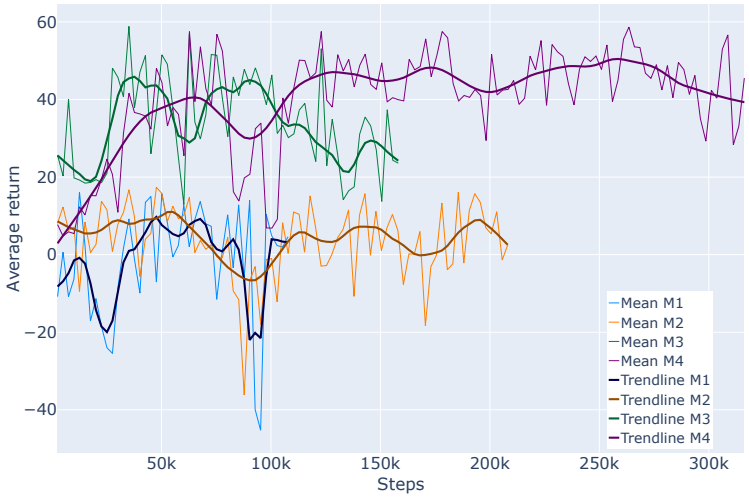


Figure 6.12. Average returns in the interim evaluation for PNN-like agents fine-tuned in the real environment. M1 (blue), M2 (orange), and M3 (green) do not show a learning transitory toward a better performance. Although M4 (purple) does not perform well initially, it is capable of training its student's network in a few-shot transfer using 60 k steps and reaching an average return of 60.

Table 6.9. PNN-like agents interim accuracy in the sim-to-real transfer.

Model name	Interim accuracy (%)	Training steps
M1	50	100 k (stopped improving after 35 k)
M2	55	280 k (stopped improving after 20 k)
M3	72.5	150 k (best model in 35 k)
M4	**95**	**300 k (best model in 60 k)**

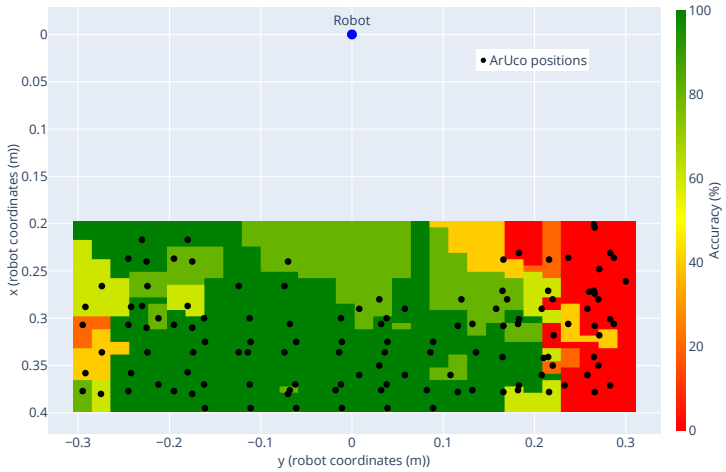


Figure 6.13. PNN-like agent accuracy heatmap for the AR targets in the IRB120 real environment. The x and y axes correspond to the robot's coordinates, while the color gradient reflects the accuracy percentage.

Analyzing the area with negative y coordinates, we also observe a very good performance, failing only in part of the farthest corner. It should be noted that, except in the previous small region, the agent can maintain its success rates despite the fact that these are positions not

contemplated during training, and therefore, it is able to generalize the knowledge acquired to a larger area. Nonetheless, shifting to the y positive values, the situation is completely different. Not only is the agent unable to generalize correctly beyond the training area, but its performance drops sharply to 0 % success. We argue that this happens due to the agent's learned behavior. Despite the robot and target's initial poses, most of the robot's movements tend to be downward and to its left side. Once it is in that area, it makes some shorter movements looking for the cube, but when it does not find it, it starts to move at the same height toward its right, moving the gripper closer or farther away with each step depending on where the target is. Seldom it can search in the left zone if it has not found the target in the first steps before starting its movement to the right. Figure 6.14 represents the gripper trajectory over six episodes where the target was on the area with y negative values (red and orange), on the central area (blue and green), and on the y positive coordinates (purple and brown). The square markers represent the initial points, while the crosses are the finish locations. Except for these last two, the robot reached the target in all four episodes. As stated, the robot seems to always perform the same movements regardless of the target location. The purple and brown trajectories show that if the targets are placed in the y positive region and they are not reached in the first steps, the robot motion keeps sweeping the workspace area until the episode ends.

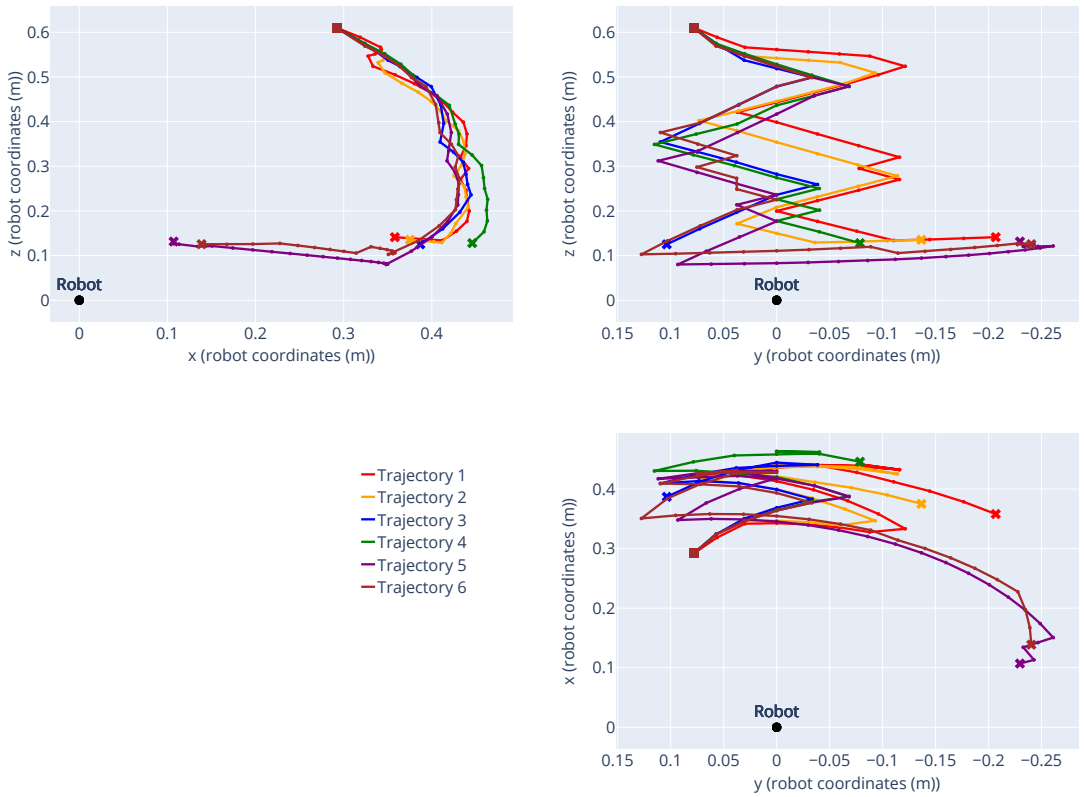


Figure 6.14. 3D robot trajectories in the IRB120 real environment projected onto a dihedral system comprising plan, front, and side views, obtained with the PNN-like agent. The red and orange trajectories correspond to targets placed on the region with y negative coordinates, the blue and green are central targets, and the purple and brown targets are on the area with y positive coordinates. The square markers represent the initial points, while the crosses are the finish locations. Except for the last two, the trajectories successfully reached the target.

To continue with the assessment of the PNN-like agent, Figure 6.15 shows the heatmap obtained when it was evaluated with a red LEGO® cube. These results confirm, on the one hand, that the solution designed using AR to speed up learning in the real environment is valid

since it allows the agent to maintain the results when a real object is used; and, on the other hand, that the previously observed pattern with the ArUco tags (Figure 6.13) in which the agent's performance was much lower on the y positive region is repeated.

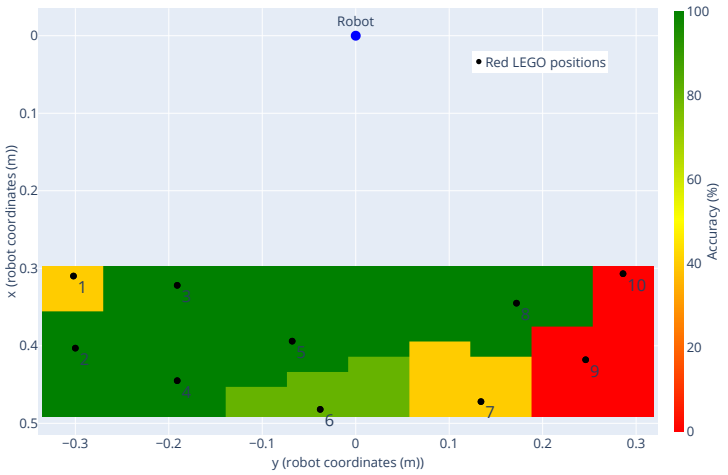


Figure 6.15. PNN-like agent accuracy heatmap for the red LEGO® cube in the IRB120 real environment. The x and y axes correspond to the robot's coordinates, while the color gradient reflects the accuracy percentage.

Once we have proved that the agent is able to perform correctly in unseen workspace zones and with real objects instead of AR targets, we keep evaluating its generalization capabilities by using a yellow and a blue LEGO® cubes, and a red and white mug. Table 6.10 presents the accuracy results obtained. Note that we only focused on four of the previous ten spots where the agent achieved 100 % success with the red LEGO® cube, positions 3 (0.322, –0.191) m, 4 (0.445, –0.191) m, 5 (0.394, –0.068) m, and 8 (0.345, 0.172) m in Figure 6.15. Overall, we can state that the agent generalizes very well the knowledge acquired with the AR red cube, being able to keep a high performance level even with real targets of different colors and shapes. It is remarkable that the agent's outcomes do not worsen with the blue target as in the virtual environment, suggesting that the learned state representation activations in the sim-to-real approach are more robust than those of the sim-to-sim implementation. For these three cases, the worst percentage is obtained in position 4, which is expected as it corresponds to the fa rthest point with respect to the robot's base.

Table 6.10. Accuracy percentages in the PNN transfer for the yellow, and blue LEGO® cubes, and the red and white mug at different position IDs in the IRB120 environment.

	Accuracy (%)		
Position ID	**Yellow LEGO® cube**	**Blue LEGO® cube**	**Red and white mug**
3	**100**	**100**	20
4	20	20	0
5	80	**100**	40
8	**100**	**100**	**100**

With these evaluations, we demonstrate that using AR allows rapid training in the real environment and does not negatively affect the agent's performance, on the contrary, it exhibits some interesting generalization capabilities that might result from the fact that the AR targets displayed have minor variations.

6.5. Conclusion

This chapter explored PNNs as a solution to bridge the reality gap in a few-shot approach. First, we analyzed the PNN learning dynamics and forgetting with respect to the teacher's task using a sim-to-sim strategy in which we trained three PNNs-like agents in environments with different complexity. The results show that the agents learn the task in the new scenarios re-using part of the knowledge from the PNN first column. With respect to the forgetting, we observed that for those setups where the agents attained almost a perfect performance, they present partial forgetting when they are evaluated on the first-column environment. However, if agents do not master the new task, they do not experience partial forgetting. We suggest that this behavior can be understood as an *overfitting* matter where agents that dominate the new task learn their weights completely molded to the new scenario. We leave as future work studying the influence of the implementation of lateral connections and if introducing episodes from the other columns' setups during the learning process or designing a voting mechanism that decides which column should determine the policy solves this partial forgetting issue.

In the resolution of the sim-to-real problem with the PNN architecture, the first-column agent was trained with raw observations from the virtual environment instead of the noisy observations that *a priori* performed better in the zero-shot try (this issue will be further explored). Regarding the second-column agent's accuracy in the workspace area, it attains fairly good results. Evaluating the agent in unseen target positions outside the training workspace area, we noticed that there is an asymmetrical behavior. While on the robot's right-hand side, it is able to generalize the learned knowledge, it cannot do the same on its left-hand side. We suggest that this might be because the learned policy always follows the same initial movements, regardless of where the cube is. We argue that, with the light NN architecture used as the student, the model is not able to properly capture all the required factors of variation in the target domain. Therefore, the agent needs to act with a poor representation of the environment and can only find a general robust moving policy. When using real objects as targets, we observed that the agent is able to generalize its knowledge and keep great performance levels. Besides, it also manages to reach the target when they have different colors and different shapes thanks to its general robust policy.

After these experiments, we conclude that using PNNs as a sim-to-real technique efficiently bridges the reality gap in a few-shot attempt. Overall, the good results achieved in the post-training evaluation and the agent's ability to generalize when exposed to unseen workspace areas, colors, or object types, demonstrate the PNN-like agent's robustness. However, the implementation of the method presents some weaknesses:

1. The fact that the initial movements of the robot are always the same, which involves a poor performance in a non-negligible region.
2. The difficulties in adapting and fine-tuning the hyperparameters of the learning algorithm in the real environment.
3. The time needed to get the agent to learn, since although 60 k samples are an excellent result according to the state-of-the-art [Rus+17], the transfer process will always be more inefficient than a zero-shot method.

7

Domain Adaptation

Imagination is the faculty of discovery.
Ada Lovelace (1815–1852)

This chapter focuses on the implementation of a methodology based on Domain Adaptation (DA) to reduce the visual discrepancies between virtual and real environments. It begins with an introduction to the fundamentals of CycleGANs as a DA technique used for image-to-image translation, followed by the introduction of an original CycleGAN variant, StyleID-CycleGAN (SICGAN). Then the objectives and contributions of the chapter are presented. The subsequent sections describe the methodology, the experiments conducted, and the results achieved, showcasing the effectiveness of SICGAN as an image-to-image translator and the effectiveness of the proposed pipeline to transfer experience to the real scenario in a zero-shot implementation. The results of this chapter are currently being reorganized in a paper named "SICGAN-Driven Sim2Real Transfer: Zero-Shot Deployment on Robotic Manipulators through Visual Deception," which will be submitted to a JCR journal.

7.1. Domain Adaptation fundamentals

Domain Adaptation (DA) [WD18] is a field that encompasses different techniques that address the sim-to-real problem through domain fusion, i.e., starting from a source domain and a target domain, the aim is to obtain a third domain that integrates characteristics of both. This common domain can be obtained by searching and designing a shared latent space that translates experience or, in the case of visual domain problems, through an image-to-image translation process. Due to the nature of our problem, we focus on this last path.

The idea of translating images in the sim-to-real context has two alternatives: if we translate from the source (i.e., the virtual environment) to the target domain (i.e., the real environment), an agent can be trained in a virtual environment with translated observations that look like a real image. If the translation is from the target to the source domain, when the reality gap is bridged, the real observations can be translated to resemble the simulated observations so the trained agent with virtual images can recognize the scenario. Therefore, in any of the two options, the

sim-to-real problem is solved with a zero-shot approach where the agent does not need real experience to fine-tune its weights as the adaptation is made as a previous "image preprocessing" step in the pipeline. One of the most well-known techniques to develop this image translation is the Cycle-Consistent Generative Adversarial Network (CycleGAN) [Zhu+17].

7.1.1. CycleGAN fundamentals

CycleGANs emerged as a solution to the limitations presented by GANs [Goo+14] concerning the requirements about paired data. At its core, CycleGAN has two neural network architectures: the Generators and the Discriminators. Each domain, S and T, has a generator that tries to translate images between domains and a discriminator that evaluates how similar the generated images are with respect to the real images from the target domain. Generator G maps images from the X domain to the Y domain, i.e., $G : X \to Y$, and, conversely, generator F maps from Y to X, i.e., $F : Y \to X$. Each generator has assigned a discriminator, thus, D_Y differentiates between real images from Y and the generated $G(X)$, and D_X discriminates between real images from X and the generated $F(Y)$. Figure 7.1 illustrates the relation between these elements.

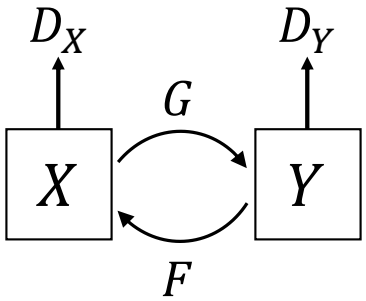


Figure 7.1. Relation between the X and Y domains, the generators G and F, and the discriminators D_X and D_Y in the CycleGAN.

Besides the image translation, the CycleGAN model ensures that the synthetic images preserve meaningful relationships from the original image in the source domain through two loss functions: one derived from the original GANs, and another uniquely designed for this model. The *adversarial loss* encourages generators to produce images indistinguishable from real images in the target domain with the purpose of "fooling" the discriminators. Equation (7.1) and (7.2) define the adversarial loss for G with D_Y and F with D_X, respectively. The goal of the discriminators is to minimize the sum of the losses, while the generators are trained to maximize it by generating realistic fake images.

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x)))] \tag{7.1}$$

$$\mathcal{L}_{\text{GAN}}(F, D_X, X, Y) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D_X(x)] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log(1 - D_X(F(y)))] \tag{7.2}$$

Another loss function (7.3) is required for the generator, the so-called *cycle consistency loss*, briefly introduced in Section 2.3.2. This loss guarantees that the mappings G and F are meaningful and maintain the input images' essential structure. Therefore, translating a synthetic image back to its source domain should resemble the original image (i.e., $F(G(x)) \approx x$ and $G(F(y)) \approx y$).

$$\begin{aligned} \mathcal{L}_{\text{cyc total}} &= \mathcal{L}_{\text{cyc}}(G, F) + \mathcal{L}_{\text{cyc}}(F, G) \\ &= \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1] \end{aligned} \tag{7.3}$$

As a result, the generators' objective function implemented in CycleGANs is:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, X, Y) + \lambda \mathcal{L}_{\text{cyc tot}} \tag{7.4}$$

where λ is a hyperparameter that weights the cycle consistency loss with respect to the adversarial loss.

Although our implementation is deeply based on the original CycleGAN architecture, we developed an original implementation that integrates features from StyleGANv2 and adds another loss. We refer to our original model as the StyleID-CycleGAN or SICGAN.

7.1.2. StyleID-CycleGAN implementation

The core architecture of the StyleID-CycleGAN (SICGAN) relies on the CycleGAN fundamentals. However, the vanilla CycleGAN implementation presents some issues related to the appearance of artifacts. To solve this issue, we implement three changes: on the one hand, based on the improvements made from StyleGAN [KLT18] to StyleGANv2 [Kar+19], we 1) remove batch normalization, and 2) we change the generators' CNN layers to demodulated convolutions, which transform weights to avoid sudden and sharp changes that might result in artifacts; on the other hand, 3) we include the *identity loss* to encourage generators to preserve the identity mapping (i.e., the original image features) when images from the target domain pass again through their generator. This loss was not implemented in the original vanilla CycleGAN implementation. Still, it was used later in just one of the applications as an additional constraint that might improve the synthetic image quality. It is defined as:

$$\begin{aligned} \mathcal{L}_{\text{id total}} &= \mathcal{L}_{\text{id}}(G, F) + \mathcal{L}_{\text{id}}(F, G) \\ &= \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(y) - y\|_1] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(x) - x\|_1] \end{aligned} \tag{7.5}$$

SICGAN uses ResNet-based generators [He+15] with several residual blocks that preserve important features during the translation and avoid the vanishing gradient. Figure 7.2 and Figure 7.3 depict the generators and residual blocks architectures, respectively. The input is a 224x224 pixel RGB image. As can be seen, the generators have the classical autoencoder structure. There is a first part with the demodulated convolutions, followed by nine residual blocks that also use demodulated convolutions. Afterward, the image is upsampled with the deconvolution or transposed convolution layers, restoring it to its original size while keeping the learned modifications in the downsampling and the residual stages. The activation function in the intermediate blocks is the ReLU, whereas in the final layer is the hyperbolic tangent (tanh) which generates pixel values in the range [–1, 1].

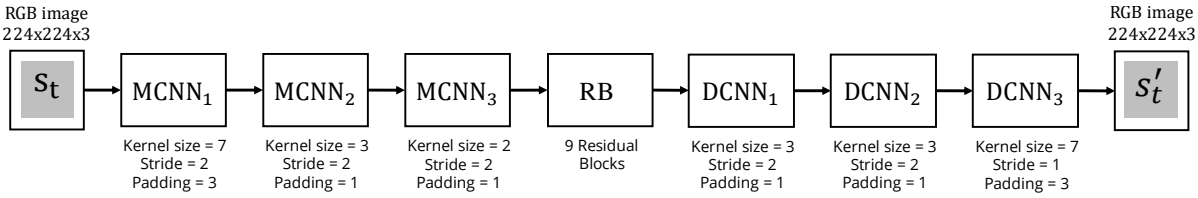


Figure 7.2. SICGAN generator architecture. The input is a 224x224 pixel RGB image, which goes through a set of modulated CNN layers (MCNN), nine residual blocks (RB) and ends with the demodulated CNN (DCNN) layers that outputs another 224x224 pixel RGB image.

Figure 7.4 presents the discriminator architecture. It takes the 224x224 pixel RGB image and passes it through the different CNN layers until the last block, a PatchGAN-based output [LW16;

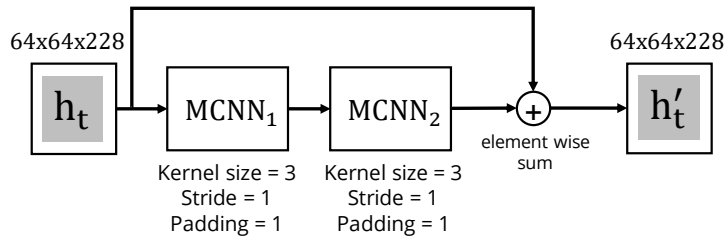


Figure 7.3. CycleGAN residual block architecture. It has two modulated CNN layers (MCNN) whose output is added in an element-wise sum to the input.

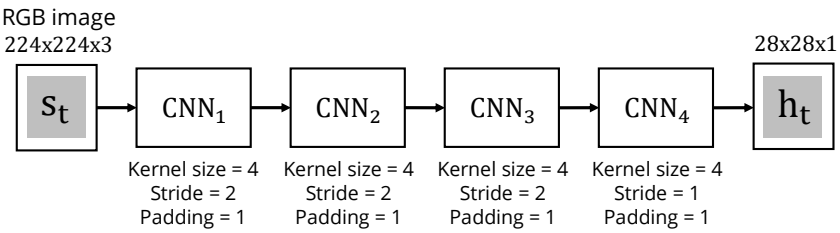


Figure 7.4. SICGAN discriminator architecture. The input is a 224x224 pixel RGB image, which goes through a set of CNN layers. The output is a 28x28 prediction matrix.

Iso+17] consisting of a 28x28 prediction matrix. PatchGAN is a type of discriminator used in GANs that classifies image patches, rather than the entire image, as real or fake. Each matrix element refers to an image region and determines whether it is part of the original image. As aforementioned, the objective is to train the generators so the discriminators cannot decide whether these image regions are synthetic. Instead of classifying the whole image as real or fake, the PatchGAN discriminator is a more efficient solution that focuses on local features, leading to better results. The activation function used in the intermediate blocks is the Leaky ReLu with a 0.2 negative slope. This function allows small negative values and prevents failures due to issues with the gradients.

Our final generators' loss function is defined as:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, X, Y) \\ + \lambda_{\text{cyc}} \left(\mathcal{L}_{\text{cyc}}(G, F) + \mathcal{L}_{\text{cyc}}(F, G)\right) \\ + \lambda_{\text{id}} \left(\mathcal{L}_{\text{id}}(G, F) + \mathcal{L}_{\text{id}}(F, G)\right)\end{aligned} \tag{7.6}$$

where $\lambda_{\text{cyc}} = 10$ and $\lambda_{\text{id}} = 0.1$.

7.2. Objective and contributions

The goal of using DA in our search for an efficient solution to the sim-to-real problem under an industrial setting is to achieve a zero-shot transfer by bridging the visual discrepancies between environments through domain translation. These visual mismatches make the virtually trained agent deployment in the real scenario challenging, as the only environment observations are images. Hence, if we modify the appearance of these images so that the agent is trained in the virtual scenario with observations similar to those of the real environment, its implementation in the real environment should be straightforward. This proposal can avoid the issue of choosing only a set of random features in DR, and the time and experience needed in the few-shot approach with PNNs.

The main contributions of this chapter are:

- The implementation and validation of a suitable methodology for addressing the sim-to-real issue in an industrial setting using DA in a previous stage from the DRL agent training.
- An original CycleGAN implementation, StyleID-CycleGAN (SICGAN), that integrates improvements from related architectures that minimize the weaknesses of the original vanilla CycleGAN proposal.
- The detailed analysis of the different types of observations that can be used to train the virtual agent seeking the zero-shot transfer.

7.3. Method

Figure 7.5 presents the methodology followed to achieve the zero-shot transfer with the DA approach using our SICGAN implementation. First, we train the SICGAN with a labeled dataset with virtual and real observations. As we seek a solution suitable for an industrial application and layout, we decided to address the sim-to-real step by translating images from the virtual to the real scenario. Our rationale was that since running the trained SICGAN in a GPU is highly recommended, we cannot expect to have this hardware available in an industrial setting. Hence, we prefer to train the model in the virtual environment with the translated images and convenient hardware resources, using standard equipment later in the real setup deployment. After training the virtual agent with the real-synthetic images, we directly deploy and evaluate it in the real environment using the raw observations from the camera as inputs. The evaluation uses ArUco Augmented Reality (AR) targets and real targets to test the agent's generalization capability.

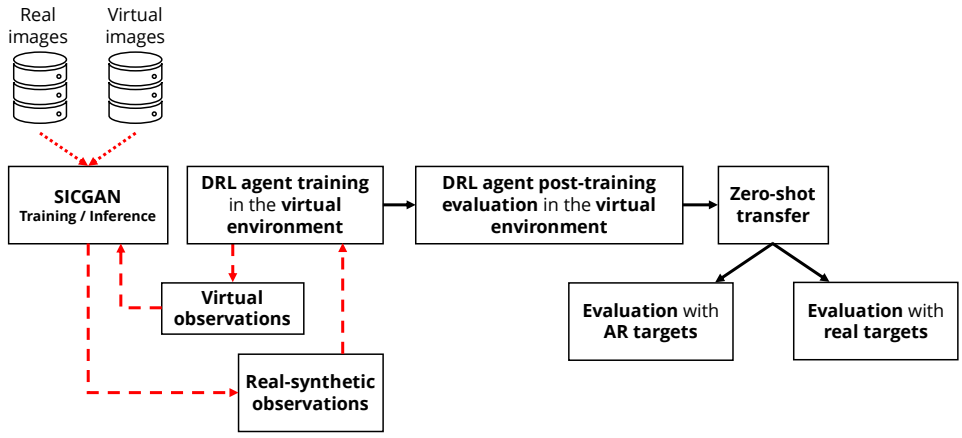


Figure 7.5. Zero-shot transfer methodology followed using SICGAN. The DRL agent is trained with real-synthetic observations generated by the SICGAN. After checking the correct performance in the virtual environment with a post-training evaluation, the DRL agent is deployed in a zero-shot transfer to the real environment, where it is assessed against AR and real targets.

Despite the previous decision, the SICGAN can also perform the opposite translation from the real to the virtual environment, a real-to-sim route for the images. The only modification that should be made to the methodology described is that the agent should be trained in the virtual environment with its raw observations. Once deployed in the real setup, the environment image should go through the SICGAN for translation to become the input to the trained agent.

7.3.1. StyleID-CycleGAN training procedure

The dataset prepared to train the SICGAN has 1,330 224x224 pixel RGB labeled images from the virtual and real environments with random robot poses, split into a 70/30 train-test ratio. Images only capture the robot and the scenario elements without the target. The adversarial loss was calculated following the Mean Squared Error (MSE) or squared L2 norm, while the cycle consistency loss and the identity loss were obtained using the Mean Absolute Error (MAE) or L1 norm. The optimizer used was Adam with $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The weights were initialized with a Normal distribution $w \sim N(0, 0.02^2)$. Table 7.1 gathers the most relevant hyperparameters.

Table 7.1. SICGAN model hyperparameters.

Hyperparameter	Value
Number of epochs	500 (max)
Batch size	1
Learning rate	5×10^{-4}
Generator optimizer	Adam ($\beta_1 = 0.5$ and $\beta_2 = 0.999$))
Discriminator optimizer	Adam ($\beta_1 = 0.5$ and $\beta_2 = 0.999$)
Loss functions	Squared L2 norm (adversarial) L1 loss (cycle consistency and identity)
Weight initialization	$w \sim N(0, 0.02^2)$
Image size	224×224×3

The best SICGAN model selection was made by analyzing those models with a better balance between the generator and discriminator train and validation losses and visually inspecting their performance when translating images both ways, from sim to real and vice versa.

7.3.2. DRL agent training and evaluation procedure

The DRL agent training procedure in the virtual environment does not fundamentally differ from the standard process explained in Section 4.2.1, but has some particularities related to the modification of the observations that feed the agent. The raw virtual environment observation is translated through the SICGAN to a real-like observation without considering the target. The target is plotted after the image has been translated. Its location between episodes is randomly chosen from the ArUco positions saved. As a result, the DRL agent receives a complete real-synthetic observation. Still, its actions are performed in the virtual environment, thus maintaining all the advantages of acting in a setup where the time and economic outlays derived from learning from scratch are minimal and where the efficiency of parallel learning in the A3C can still be exploited.

Regarding the DRL agent training, instead of waiting to reach the 30 M steps, we end the process in 25 M steps because the steady state is reached earlier. The interim evaluation and post-training evaluation procedure performed in the virtual environment remain as explained in Section 4.2.1.

Once the agent was evaluated in the virtual environment, we performed a zero-shot transfer to the real environment. As inSection 6.4, we evaluated each ArUco position five times with random initial robot poses. As a result, we built a heat map with the agent's accuracy percentage

in each tag, and to fulfill the complete workspace area, we interpolated with the nearest known neighbors. After the evaluation with the AR targets, we use real objects, e.g., a red, a yellow, and a blue LEGO® cubes, and a red and white mug (Figure 3.6), to assess the agent's generalization capability. Starting with the red LEGO® cube, which is the most similar to the AR target, we examine ten different positions the agent has not seen. Regarding these results, we select four positions where the agent masters the task and evaluate them with other objects of different colors or shapes.

7.4. Description of the experiments

After training the SICGAN, we designed two types of DRL agents. The first has the same MDP conditions as the BM (Section 4.2.2), without further restrictions than the ones imposed by the physics engine, while the second includes in the simulation the motion constraints of the physical robot (Section 3.2.1). Briefly, these constraints limit the robot's movement to a semicircular annulus, centered in the robot's base. The inner and outer circumferences have radii of 25 cm and 55 cm, respectively. Based on the results achieved in the virtual environment, we selected the best approach and model to perform the zero-shot transfer using the AR and real targets.

7.5. Results and discussion

7.5.1. StyleID-CycleGAN results

In the SICGAN, ideally the goal is to reach a balance where the generator and discriminator networks gradually improve. Over time, the generator loss should decrease stabilizing as the generator learns to fool the discriminator consistently, and the discriminator loss fluctuates around a midpoint where the discriminator is challenged but still effective, indicating that the generator is producing high-quality, realistic images. Figure 7.6 presents the SICGAN train and test losses for the generator and the discriminator. The generator loss curves are obtained by calculating for each of the generators the losses from Equation (7.6) dividing it by the number of train or test samples N ((7.7)). On the other hand, the discriminator loss is calculated as the adversarial losses mean ((7.1) and (7.2)).

$$\begin{aligned}\mathcal{L}_{\text{gen\_avg}}(G, F, D_X, D_Y) = \frac{1}{N}\sum_{i=1}^{N}\Big(&\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, X, Y) \\ &+ \lambda_{\text{cyc}} \cdot (\mathcal{L}_{\text{cyc}}(G, F) + \mathcal{L}_{\text{cyc}}(F, G)) \\ &+ \lambda_{\text{id}} \cdot (\mathcal{L}_{\text{id}}(G, F) + \mathcal{L}_{\text{id}}(F, G))\Big)\end{aligned} \tag{7.7}$$

Even though the generator's losses rapidly decrease, the increase in the discriminator's losses is almost unperceivable. Besides, instead of decreasing smoothly during the training process, from epoch 150 onwards, the loss starts to increase again until a sudden drop in epoch 300. We argue that around epoch 150 there might be a partial mode collapse, where the generator starts to produce similar outputs, limiting its diversity and, hence, its ability to fool the discriminator. The sudden loss drop around epoch 300 suggests a significant improvement in the generators' performance. Between these two epochs, the discriminator's loss slightly decreases and then increases, indicating that it is adapting to the generators' outputs. If we look at the synthetic

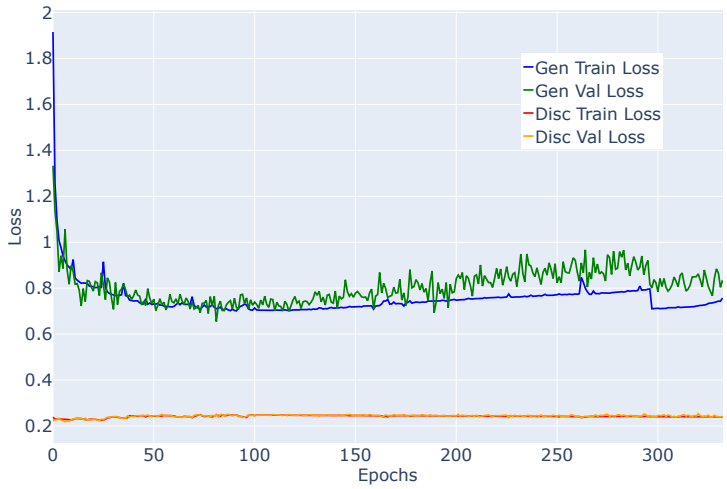
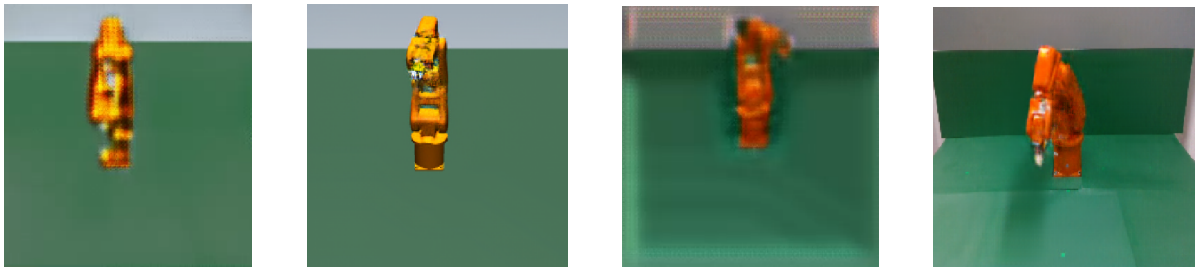


Figure 7.6. SICGAN losses curves in training and validation for the IRB120 environment. The y-axis represents the loss values, while the x-axis denotes the epochs. The model was trained for 300 epochs approximately. The generator loss curves (blue for training and green for validation) include the loss of both generators. The same occurs for the discriminator loss (red for training and orange for validation).

images produced in real-to-virtual or virtual-to-real translations, the results are quite good (Figure 7.7). The SICGAN model is learning useful representations to translate between the real and synthetic domains, even if the losses are not perfectly stable. The selected SICGAN model is that of epoch 114, has a training loss of 0.703 and 0.246 for the generator and the discriminator, respectively, and test losses of 0.719 and 0.246.



(a) **(b)** **(c)** **(d)**

Figure 7.7. Real-to-virtual observation translation in the initial SICGAN training step (a) and in the final selected model (b). Virtual-to-real observation translation in the initial SICGAN training step (c) and in the final selected model (d) for the IRB120 environment.

7.5.2. DRL agent results in the virtual environment

Figure 7.8 shows the training curves for the two agents trained with and without the motion constraints. Both agents reach the steady state, but the unconstrained DRL agent learns faster and achieves a final average return higher than the agent that incorporates the movement restrictions. We suggest that this happens since the beginning of the training process because the agent has to learn the limiting rules in addition to the task itself. As a result, it slows down and hinders the learning process. Moreover, with this approach, we guide the learning process excessively when one of the DRL maxims is not to tell the agent how to act but only what to do. Hence, we chose the DRL agent that learned without the limitations. Note that despite the fact that the agent does not have the physical motion limits explicitly defined in the MDP the MuJoCo environment avoids unfeasible solutions and, in the end, the agent learns to act appropriately.

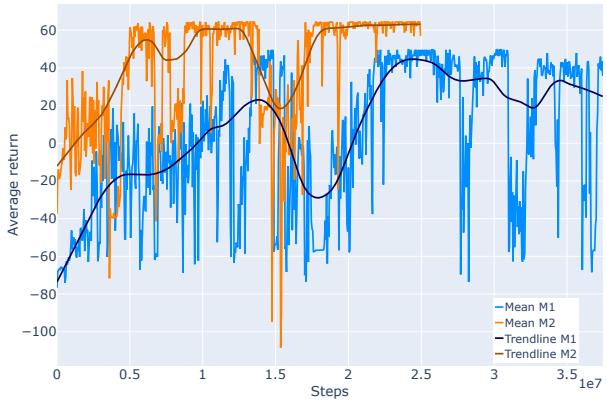


Figure 7.8. Average returns obtained in the interim evaluations of agents with the real-synthetic observations in the IRB120 environment for 35 M training steps (note the 1e7 scaling factor on the right-hand corner of the figure). The agent without movement constraints (orange) learns faster and more than the agent with the limitations (blue). The orange training was stopped earlier due to the high performance already achieved.

The best model without constraints was achieved at step 6.35 M, and its post-training evaluation in the virtual environment with the translated images resulted in a 100 % accuracy over 1,000 episodes with the random initial robot and target poses that we use to validate the model before performing the zero-shot transfer.

7.5.3. Zero-shot results

Considering the different strategies described in Chapter 5, we can use three types of observations to train an agent and perform the zero-shot transfer. Figure 7.9 analyzes the RGB histogram distributions, where (a) is the raw virtual, (b) is the noisy, (c) is the real-synthetic, and (d) the real observations. The raw virtual environment image presents very noticeable visual mismatches and a complete absence of noise, leading to a zero-shot transfer with only 15.8 % accuracy. If we introduce Gaussian noise, a low-level DR approach, to this raw virtual image, the histograms are more similar, but the zero-shot success rate is only 34.1 %. Finally, we introduced in this chapter the real-synthetic observations created by SICGAN, which presents a style almost identical to that of the real observations.

Regarding the zero-shot transfer for the DRL agent trained with the translated observations from SICGAN, Figure 7.10 displays the heatmap obtained with its accuracy results when we evaluate each ArUco pose five times. Conversely to the PNN-like agent heatmap (Figure 6.13), Figure 7.10 presents a much more balanced success rate distribution. Although the area with y positive coordinates is still the region with the worst results, its size has been considerably reduced, and the lowest accuracies occur in remote positions that have not been seen during training. The same happens in the farthest corner of the robot's right-hand side. In general, the agent performance is clearly above 80 % for most of the tags, surpassing the results achieved with the PNN few-shot transfer.

In addition, we analyze the trajectory of the robot's gripper for six different episodes (Figure 7.11), two on the region with y negative coordinates (red and orange), two on the central area (blue and green), and the area with y positive coordinates (purple and brown). The square markers represent the initial points, while the crosses are the finish locations. Except for the fifth trajectory (purple), the robot reached the target in the other episodes. Indeed, the agent failed that target, but the finish position was nearby. Since the beginning, the robot's movement is consistent with where the target is placed, conversely to the trajectories observed

with the PNN-like agent where it always performed the same type of approach regardless of the goal pose (Figure 6.14). This suggests that the agent trained with the SICGAN is able to obtain a better state representation than the PNN-like agent as it is able to adapt its decisions depending on where the target is and search it within the proper region.

Analyzing now the zero-shot results attained with the red LEGO® cube, Figure 7.12 depicts the accuracy heatmap for the same ten positions evaluated with the PNN-like agent. In this case, we also validate the use of AR targets for speeding up the training and performing an efficient and detailed examination of the agent performance in the real setup. The distribution of the success rates in the workspace area is similar to Figure 7.10. In the ten positions, the agent can reach the cube at least once.

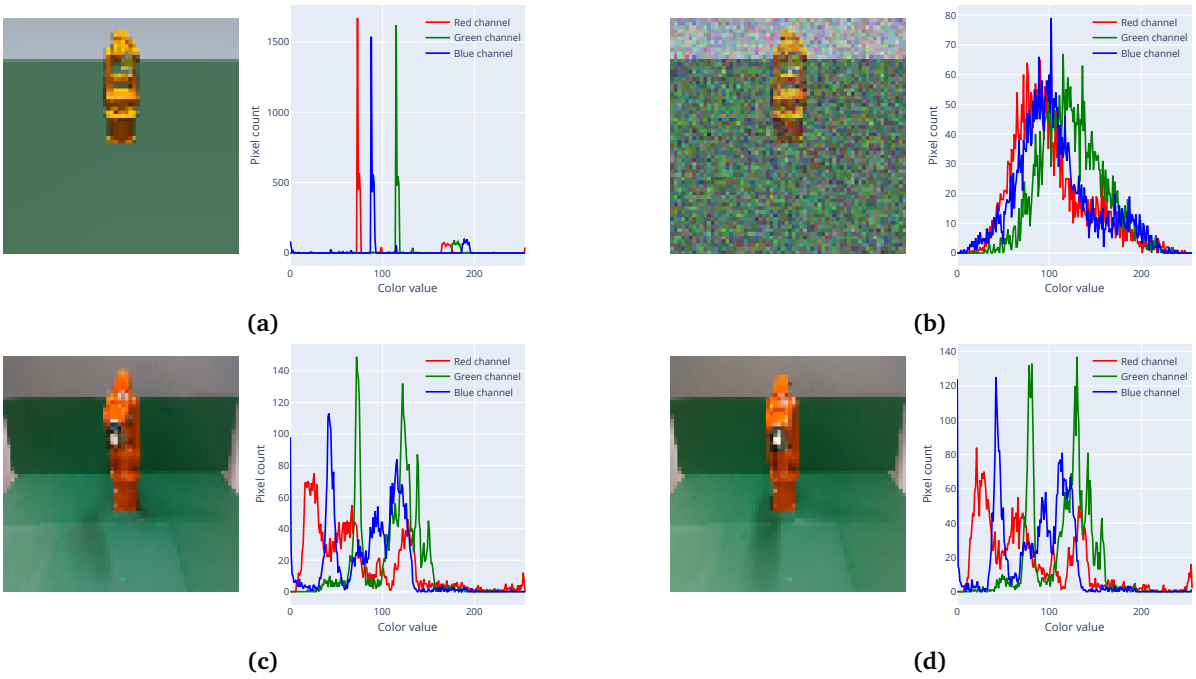


Figure 7.9. RGB histograms for the raw virtual (a), noisy virtual (b), real-synthetic (c), and real (d) observations. In the sequence presented, each observation is a step closer to the real observation distribution, and hence, there is more chance of success in the sim-to-real transfer.

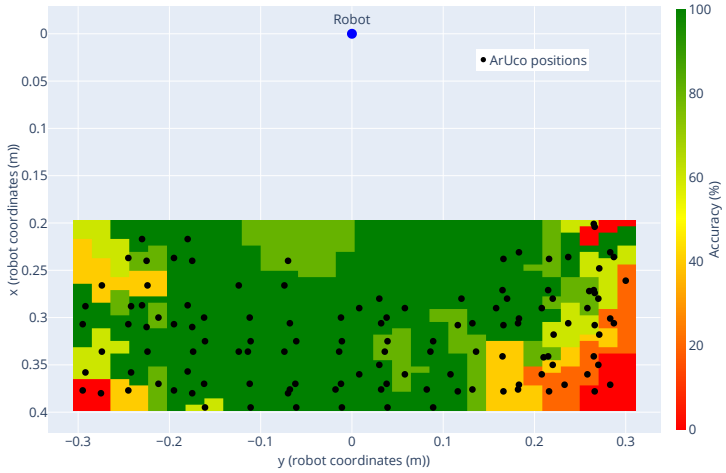


Figure 7.10. DRL agent trained with real-synthetic observations accuracy heatmap for the AR targets in the IRB120 real environment. The x and y axes correspond to the robot's coordinates, while the color gradient reflects the accuracy percentage.

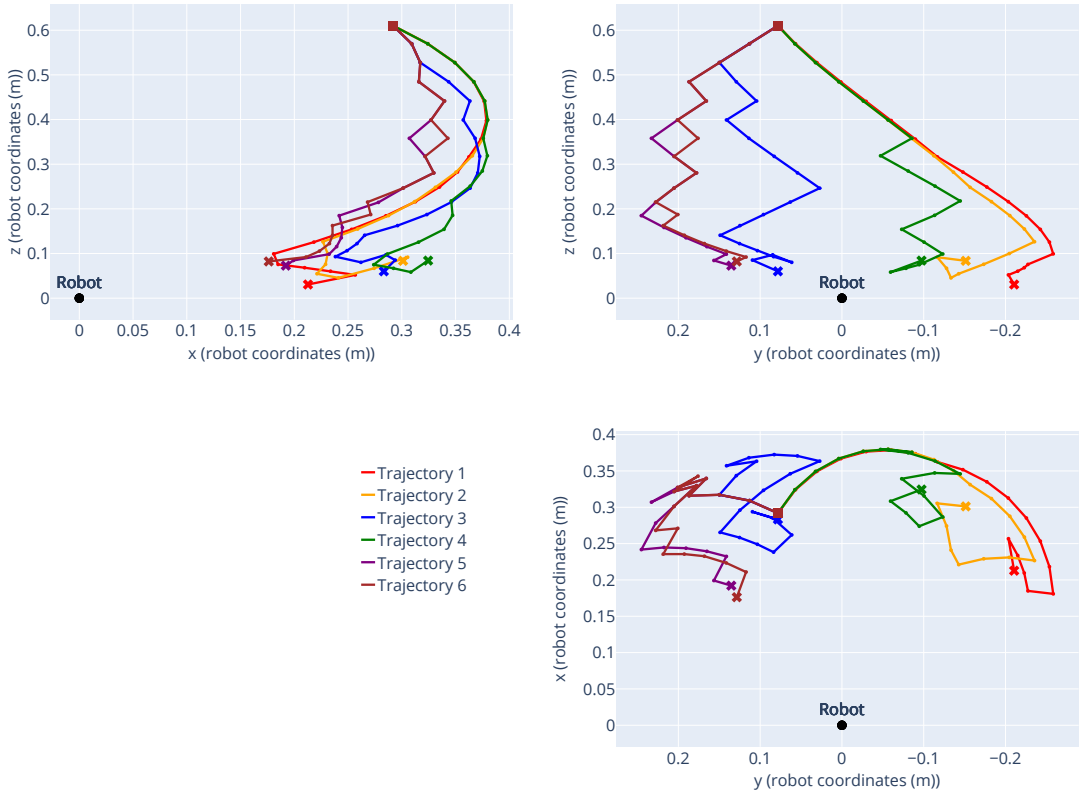


Figure 7.11. 3D robot trajectories in the IRB120 real environment projected onto a dihedral system comprising plan, front, and side views, obtained with the zero-shot. The red and orange trajectories correspond to targets placed on the region with y negative coordinates, the blue and green are central targets, and the purple and brown targets are on the area with y positive coordinates. Except for the fifth, the trajectories successfully reached the target. The square markers represent the initial points, while the crosses are the finish locations.

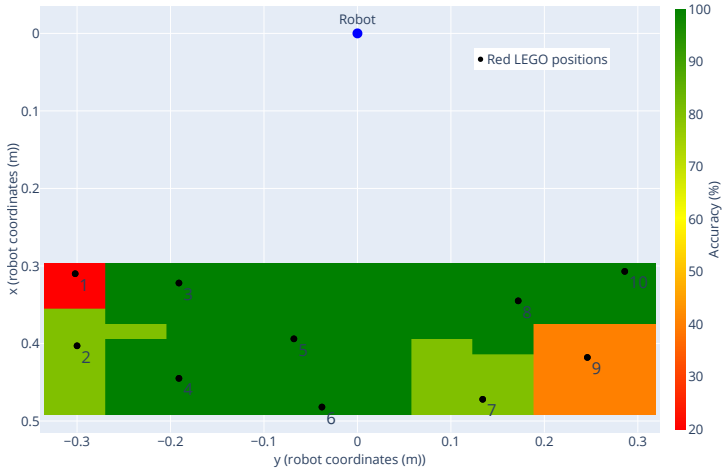


Figure 7.12. DRL agent trained with real-synthetic observations accuracy heatmap for the red LEGO® cube in the real environment. The x and y axes correspond to the robot's coordinates, while the color gradient reflects the accuracy percentage.

To complete the evaluation of the agent, we assess its generalization capability when facing the yellow and blue LEGO® cubes and the red and white mug. The results of Table 7.2 focus on the same four locations as in the PNN evaluation, position 3 (0.322, –0.191) m, position 4 (0.445, –0.191) m, position 5 (0.394, –0.068) m, and position 8 (0.345, 0.172) m, where this agent also achieved 100 % success. We observe that for the yellow cube, the agent still has a perfect performance. With the red and white mug, except for the position 4 where it attains

40 %, which was also the most struggling target for the PNN-like agent, the agent outperforms the results achieved in the few-shot approach. Finally, for the blue cube, we noticed a worsened performance compared to the PNN-like agent, but that is aligned with the results obtained when we assessed the agent performance in the adversarial environments (see Section 6.3.2.1, where the success of the BM agent decay almost 50 % when it faced a blue target). We understand that the good results achieved with the PNN respond to the robust general policy learned, in the zero-shot transfer, the agent's behavior suggests it is relying on the learned state representation. This does not happen when the target is yellow or red and white, as their RGB code still allows the correct activation of the filters learned with the red target used during training. In light of

Table 7.2. Accuracy percentages in the few-shot and the zero-shot transfers for the yellow and blue LEGO® cubes, and the red and white mug at different position IDs in the IRB120 environment.

		Accuracy (%)					
		Yellow LEGO® cube		Blue LEGO® cube		Red and white mug	
Position ID	(x, y) coordinates (m)	Few-shot transfer	Zero-shot transfer	Few-shot transfer	Zero-shot transfer	Few-shot transfer	Zero-shot transfer
3	(0.322, –0.191)	**100**	**100**	**100**	0	20	**100**
4	(0.445, –0.191)	20	**100**	20	0	0	40
5	(0.394, –0.068)	80	**100**	**100**	20	40	**100**
8	(0.345, 0.172)	**100**	**100**	**100**	**100**	**100**	**100**

these results, we can state that zero-shot transfer is successful and that the agent exhibits really good generalization capabilities in terms of the workspace area that it can cover and the color and type of the targets it can handle despite not using DR during its training.

7.6. Conclusion

During this chapter, we have proposed a zero-shot method based on an original CycleGAN implementation named SICGAN, which leverages a DA image-to-image translation approach to train a virtual agent with real-synthetic images and deploy it directly on the real environment using its observations. After training the SICGAN and validating the model, we trained the virtual agent translating the raw virtual environment observations into real-synthetic images. With the methodology we propose, the hardware in the real environment is not a limiting factor. However, it could be interesting to analyze in future research the agent performance when the observations are translated from the real to the virtual environment and compare its efficiency with respect to the current proposal.

The deployment of the virtually trained agent with the translated observations in the real setup with a zero-shot approach exhibits great results in the whole workspace area except for a few positions from the farthest right and left corners that were unseen during the training phase. Besides, the agent's behavior shows from the episode beginning that it can sense correctly where the target is and tries to reach it using the most efficient trajectory towards it, searching in the nearest zones if it does not hit it in the first attempt. On the other hand, the evaluation with the real targets proves that the agent can generalize the knowledge learned with the AR red cube, achieving the same performance level for all the targets except for some positions with the blue LEGO® cube where the performance decays considerably. We leave for future work the use of high-level DR during the training of the virtual agent with the real-synthetic images to overcome possible state representation limitations. Besides, it could be interesting to research on the integration of PNNs with this zero-shot proposal with the aim of analyzing the impact of

training several agents in a virtual environment performing different tasks, i.e., virtually train different columns of the same PNN, and then performing a zero-shot transfer to the real setup using our image-to-image translation approach.

To conclude, these results demonstrate the good performance of our proposed zero-shot methodology to efficiently solve the sim-to-real problem in an industrial application. Compared to the methodology based on the few-shot approach using the PNN architecture, with the current pipeline we have to train one more model, the SICGAN, and the virtual training phase with the translated observations is slower due to the SICGAN inference time, which indeed is not relevant as it is an off-process step. Conversely, we eliminate the training time in the real setup and the uncertainties regarding the hyperparameters and learning process of the serialized A3C algorithm. Hence, we consider the zero-shot transfer with the SICGAN more efficient than the few-shot with the PNN.

8

Sim-to-real methodology validation and semantic knowledge integration

One must do what is possible in order
to achieve the impossible.
Simone Weil (1909–1943)

This chapter presents the results of the proposed sim-to-real methodology validation. The primary goal is to demonstrate the robustness of this methodology by applying it to a different industrial asset, the UR3e collaborative robotic arm, following the same approach used with the IRB120 robot. Additionally, the chapter outlines the systematic process of adapting and validating the methodology across diverse tasks, ensuring its general applicability. It also explores the integration of semantic knowledge into DRL, highlighting how this additional contextual information improves both the learning efficiency and final performance of DRL agents. The results of this investigation have been submitted to the 39th Annual AAAI Conference on Artificial Intelligence in a paper called "Boosting Deep Reinforcement Learning Efficiency with Knowledge Graph Embeddings for Robotic Applications", and it's currently under review.

8.1. Objective and contributions

The main objective of this chapter is to validate the proposed sim-to-real methodology developed with the IRB120 robot in another industrial asset, the UR3e collaborative robotic arm (Section 3.3). To prove our proposed methodology's robustness and general applicability to bridge the reality gap in an industrial problem, we have gathered the most relevant insights from the experiments developed and explained throughout the dissertation chapters, followed that pipeline to solve the same task but with another robotic arm, and defined the general stages that have to be fulfilled to apply it to another industrial application that might not necessarily involve the same assets.

Additionally, we present our latest work in collaboration with the RoCoCo research group from the Sapienza University of Rome, which further improves the sample efficiency issue in DRL agents. To do so, we research in a sim-to-sim approach how introducing semantic information about the environment during the agent's learning process enhances its performance regarding the number of samples needed and final accuracy achieved with respect to a Baseline Model (BM) without contextual information. With these experiments, we build the foundations of the following steps that should be taken to keep improving the efficiency of our proposed methodology.

The main contributions of this chapter are:

- The validation of the proposed sim-to-real methodology on an industrial asset different from the one used during its development.
- An original DRL architecture that successfully integrates Knowledge Graph Embeddings (KGEs) with the environment semantic information during the agent's learning process.
- The quantification of the DRL agents' performance improvement, in terms of time and accuracy, when the agent has no semantic information, when it has partial knowledge, and when it has the complete embedding.
- A quantitative and qualitative assessment of the embedding's impact across diverse environments and two robotic manipulators.
- An analysis of the potential improvement in the sample efficiency of agents by evaluating the distribution of the joint orientations across the assessed episodes.

8.2. Sim-to-real methodology validation

8.2.1. Method

Figure 8.1 recalls the proposed methodology that efficiently solves the sim-to-real problem in an industrial application that was presented in the Chapter 7 and has been selected as the most suitable solution due to its great results and ease of implementation. This approach is the result of having researched, implemented, and tested three transfer learning techniques, Domain Randomization (DR) (Chapter 5), Progressive Neural Networks (PNNs) (Chapter 6), and Domain Adaptation (DA) (Chapter 7), concluding that the most efficient and robust transfer learning strategy was the zero-shot transfer achieved through DA, and more specifically, by the image-to-image translation through an original CycleGAN implementation called the StyleID-CycleGAN (SICGAN). Therefore, Figure 8.1 begins with the dataset collection to train the SICGAN. After training the SICGAN and validating the best model achieved, we train the DRL agent using the translated observations resulting from passing the raw virtual environment images through the trained SICGAN, which produces the real-synthetic pictures. Once the agent is trained and evaluated in the virtual environment, we perform a zero-shot transfer to the real setup where the agent is deployed. We assess its performance in this environment using first the Augmented Reality (AR) targets gathered with the ArUco tags to speed up the process, but then we validate these results using four real objects, a red, yellow, and blue LEGO® cubes, and a red and white mug.

Since our goal is to validate the methodology, we did not make substantial changes in the training procedures of the models. Hence, the complete details of the SICGAN and the DRL

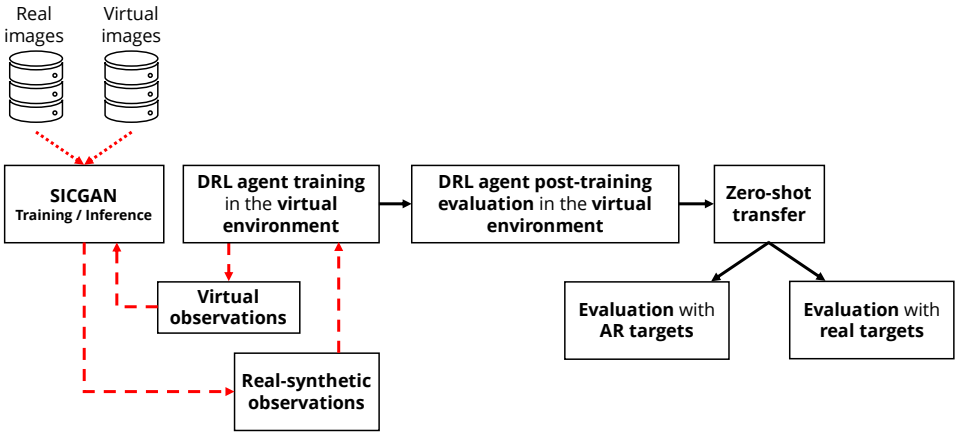


Figure 8.1. Methodology of the zero-shot transfer achieved with DA through the SICGAN. Relying on a trained model of the SICGAN with virtual and real images, the DRL agent is trained with real-synthetic observation generated by the SICGAN. After checking the correct performance in the virtual environment with a post-training evaluation, the DRL agent is deployed in a zero-shot transfer to the real environment, where it is assessed against AR and real targets.

agent training process can be found in Section 7.3. Briefly, the UR3e dataset has 1,600 224x224 pixel resolution labeled images from each environment split in a 70/30 train-test ratio. The DRL agent was trained without movement constraints for 70 M steps using the A3C algorithm. The zero-shot post-training evaluation procedure assesses first the agent's performance five times per ArUco position and then evaluates its success when facing real targets.

8.2.2. Results and discussion

Figure 8.2 presents the SICGAN training and test losses. The generator's losses decrease rapidly, although this drop is slower than in the IRB120. In this period, the discriminator's losses slightly descend, meaning that despite the quick generator improvement, it cannot yet fool the discriminator. Thereafter, the losses of the generators and discriminators stabilize, reaching a point where the generators can produce good-enough synthetic images, and the discriminators can still differentiate some images. This consistent performance suggests a good balance between generator and discriminator. After epoch 150, the generator's test loss rises gradually, indicating that its generalization capabilities are struggling with the unseen data. Besides, the fluctuations with spikes might imply training instabilities or a potential mode collapse, as in the IRB120, where the generator is learning to produce limited variations.

Taking a look into the images produced by the generator when translating from the virtual scenario to the real one, which is the direction we are interested in, Figure 8.3 shows that the visual correspondence is achieved with enough detail for our application. The selected model was obtained in epoch 249 and had a training loss of 0.741 and 0.237 for the generator and discriminator and test losses of 0.829 and 0.251.

Figure 8.4 exhibits the DRL agent training process in a virtual environment with real-synthetic images. Despite the dispersion in the average return distribution and the performance decrease around the 35 M steps, the agent reaches the steady state within the 70 M steps, obtaining the best model at the 55^{th} M step with a 90 % accuracy in the 1,000 episodes virtual environment post-training evaluation. Note that the training process lasts longer than in the IRB120 because the UR3e morphology is more challenging to understand than the IRB120 due to the offsets between joints.

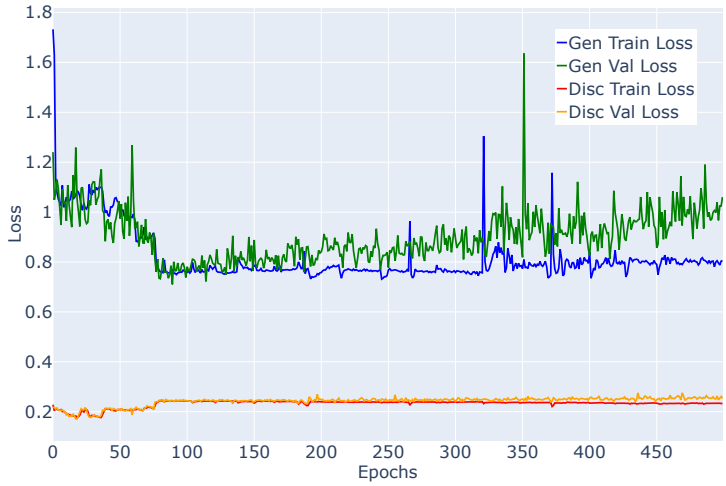
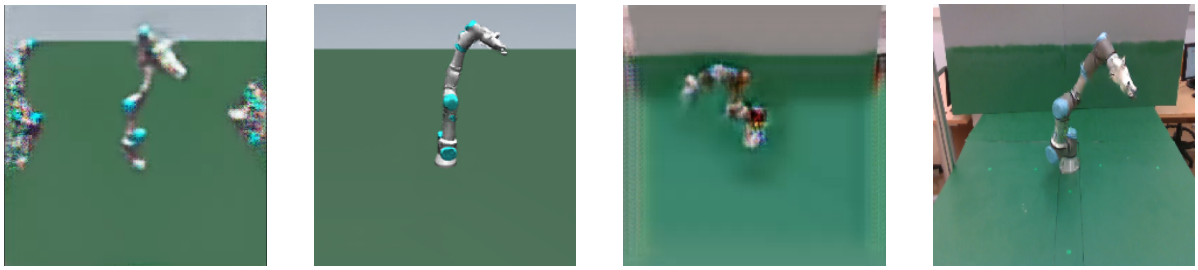


Figure 8.2. SICGAN loss curves in training and validation for the UR3e environment. The y-axis represents the loss values, while the x-axis denotes the epochs. The model was trained for approximately 500 epochs. The generator loss curves (blue for training and green for validation) include the loss of both generators. The same occurs for the discriminator loss (red for training and orange for validation).



(a) **(b)** **(c)** **(d)**

Figure 8.3. Real-to-virtual observation translation in the initial SICGAN training step (a) and in the final selected model (b). Virtual-to-real observation translation in the initial SICGAN training step (c) and in the final selected model (d) for the UR3e environment.

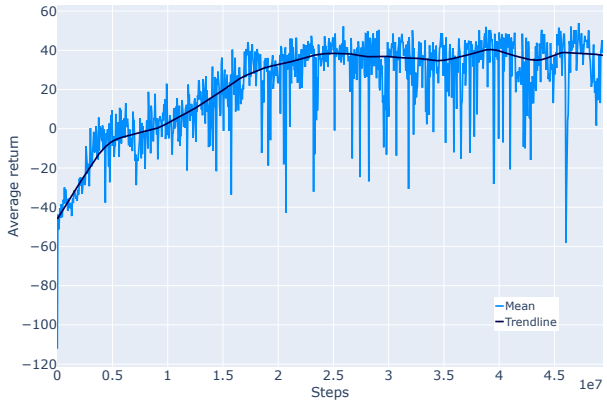


Figure 8.4. Average returns in the DRL agent interim evaluations with the real-synthetic observations in the UR3e environment. The agent has no movement constraints. It reaches the steady regime in 25 M steps.

Figure 8.5 presents the heatmap with the accuracy results of the virtually trained DRL agent when we performed the zero-shot transfer. The accuracy distribution shows a general trend of higher precision near the area with positive x coordinates and central positions, with some variations as the target positions move further away, reaching the space limit. The slight drop in accuracy might be due to the complexity introduced by larger displacements or changes in the target orientation, as already seen in the PNN agents and the zero-shot transfer results in

the IRB120 (Section 6.4.3 and Section 7.5.3, respectively). Overall, the results indicate that the robot consistently performs well within a relatively wide range. Therefore, they validate the proposed methodology and pipeline to efficiently solve the sim-to-real problem using a robotic arm that has to approach a target that randomly moves in the workspace. Besides, the outcomes suggest that the robot's geometry may influence the difficulty of reaching certain areas of the workspace when using visual feedback from a fixed-mounted camera. If this influence were evaluated across the entire working area, it could allow for better distribution of target positions in virtual training to cover more challenging regions. We leave this for future work.

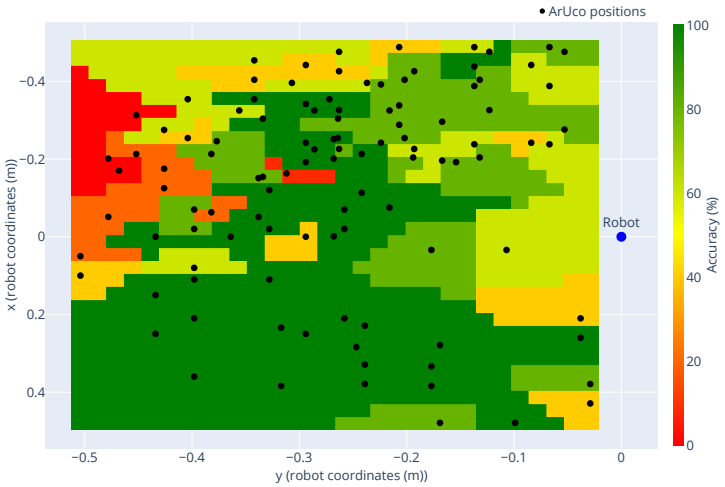


Figure 8.5. DRL agent trained with real-synthetic observations accuracy heatmap for the AR targets in the UR3e real environment. The x and y axes correspond to the robot's coordinates, while the color gradient reflects the accuracy percentage.

To conclude with the methodology validation, some experiments were done using real targets. We evaluate the agent's performance in five positions unseen during training. The targets used were the same as in the IRB120 experiments in the real scenario, the red, yellow, and blue LEGO® cubes, and the red and white mug. Table 8.1 presents the mean accuracies obtained per position and target. These results empirically show the ability of a DRL agent trained with our methodology to generalize. The worst success percentages belong to the blue cube, which has already been noticed in the IRB120 experiments, and we argue it has to do with the learned filters in the state representation layers and the activations for the RGB channels.

Table 8.1. Accuracy percentages for the red, yellow, and blue LEGO® cubes, and the red and white mug at positions not seeing during training in the UR3e environment.

	Accuracy (%)			
(x, y) coordinates (m)	**Red LEGO® cube**	**Yellow LEGO® cube**	**Blue LEGO® cube**	**Red and white mug**
(–0.385, –0.228)	**100**	**100**	60	**100**
(–0.285, –0.378)	**100**	**100**	80	**100**
(0.011, –0.437)	**100**	**100**	60	80
(0.319, –0.298)	**100**	**100**	40	**100**
(0.403, –0.102)	**100**	**100**	20	80

8.2.3. Sim-to-real methodology for industrial applications

As the final step toward this thesis's main objective, we propose the generalization of the designed methodology to address a different industrial application than the case study analyzed and solved through the dissertation.

Figure 8.6 illustrates the proposed pipeline. The **set-up stage 1** involves, on the one hand, virtualizing the environment where the agent will operate, emphasizing that high photorealism is not required for this step, which lightens the setup process, and on the other, defining the most suitable MDP to model the specific task to be solved, taking into account that the agent's observations will be RGB images. This type of observation reduces the feature engineering complexity only requiring that the visual inputs are representative of real-world conditions. The virtualization and MDP definition should be validated by ensuring that the agent can effectively learn from this model representation of the problem. After this is verified, in **set-up stage 2**, a dataset is built consisting of both virtual and real labeled images of the operating environments to train the SICGAN. Once the SICGAN is trained and validated, the **agent is trained** in the **virtual environment**, but this time, it learns using **real-synthetic translated images**, helping its adaptability to real-world scenarios while still in the simulation phase off-process.

Subsequently, a post-training evaluation is conducted to check the performance results and ensure that the agent's performance in the virtual environment using translated images meets the desired accuracy and behavior. Upon successful evaluation, the agent is then prepared for a **zero-shot transfer** to the real environment, allowing it to apply its learned policies on-process in real-world tasks without additional fine-tuning. This approach streamlines the sim-to-real transfer process, enhancing efficiency in deploying DRL agents in industrial applications while minimizing the need for extensive real-world data collection and training.

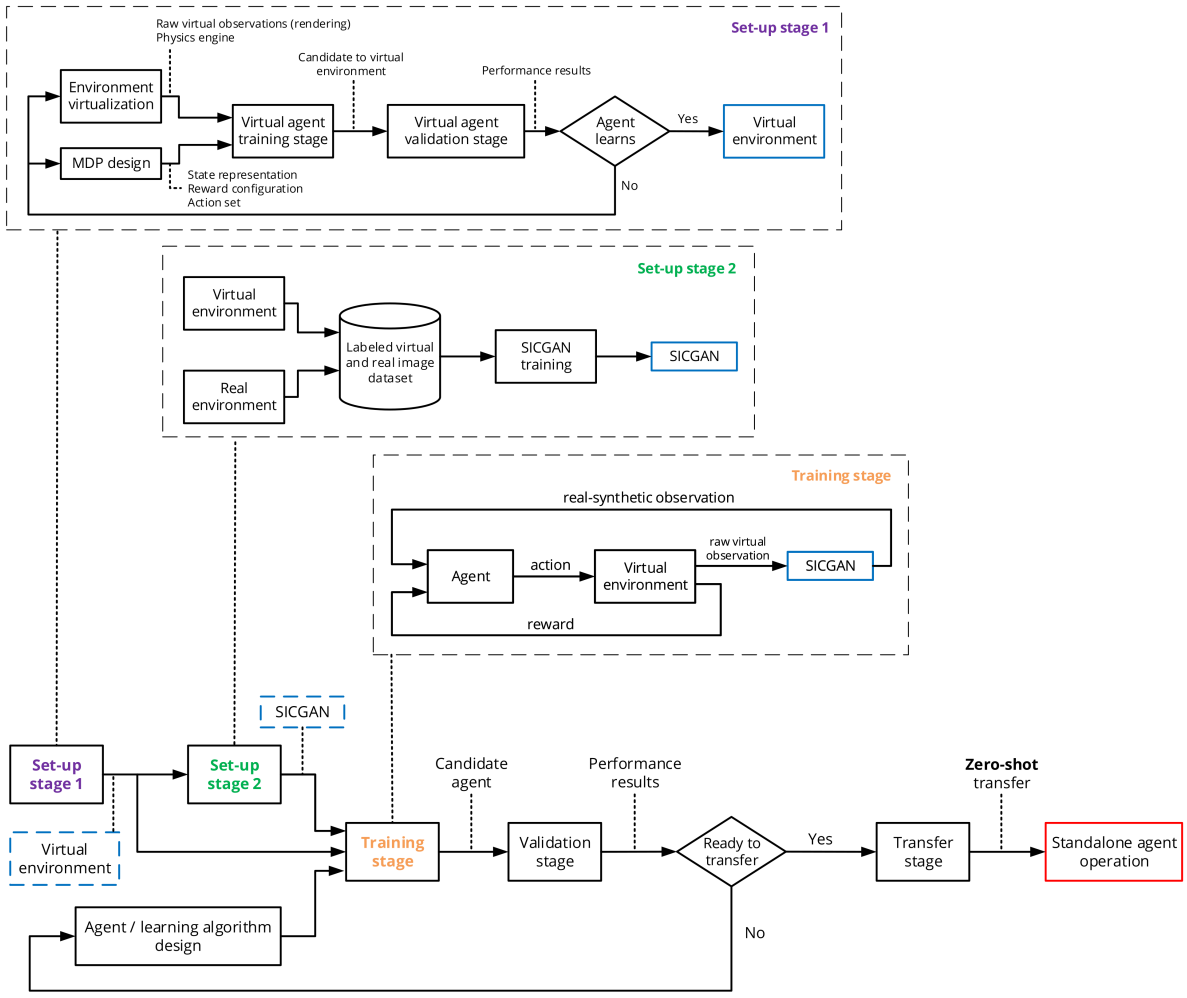


Figure 8.6. General sim-to-real zero-shot transfer methodology for industrial applications. The final pipeline begins with the set-up stage 1 in which the environment is virtualized and the MDP defined. After validating the virtual environment, set-up stage 2 defines the steps needed to train the SICGAN. Once this is accomplished, the DRL agent can be trained using real-synthetic observations generated by the SICGAN from the raw visual observations. After it is trained and evaluated in the virtual environment, the DRL agent can be deployed in the real environment using a zero-shot transfer.

8.3. Semantic knowledge integration

Framed in our search for improving the sample efficiency issue in DRL agents, we have investigated the impact of adding contextual information about the environment on the agent's performance. Our key hypothesis is that any learning process, particularly in DRL setups, should improve when the agent can access information about the environment that complements its raw observations, as demonstrated in [DL20]. We suggest that this approach provides the agent with a way to better structure the experiences it encounters during learning. Besides, this knowledge can be used almost for free as there is only a slight increase in the computational burden.

8.3.1. Method

The integration of contextual information in the learning process is based on knowledge graphs. A knowledge graph $\mathcal{G}$ is a structured representation of knowledge, consisting of entities $\mathcal{E}$ (nodes or vertices) and relations $\mathcal{R}$ (edges) that establish connections between entities. Each entity represents a distinct concept, object, or property relevant to the domain, while relations define

how these entities interact or relate to one another. For instance, $\mathcal{G}$ consists of triples (h, r, t), where $h, t \in \mathcal{E}$ are the head and tail entities, and $r \in \mathcal{R}$ is the relation (e.g., *(mug, hasColor, red)*). These triples illustrate how entities are interconnected, capturing meaningful relationships and properties. The task then involves representing the graph $\mathcal{G}$ in a continuous vector space, where entities are encoded as vectors $v_{\mathcal{E}} \in \mathbb{R}^{|\mathcal{E}| \times d_{\mathcal{E}}}$ and relations as matrices $W_{\mathcal{R}} \in \mathbb{R}^{|\mathcal{R}| \times d_{\mathcal{R}}}$, with $d_{\mathcal{E}}$ and $d_{\mathcal{R}}$ as the dimensions of the entity and relation embeddings, respectively. The goal is to learn these embeddings by minimizing a loss $\mathcal{L}$ over a scoring function $f(h, r, t)$, applied to both true triples and a set of negative triples generated via negative sampling. Figure 8.7 presents an example of the general knowledge graph.

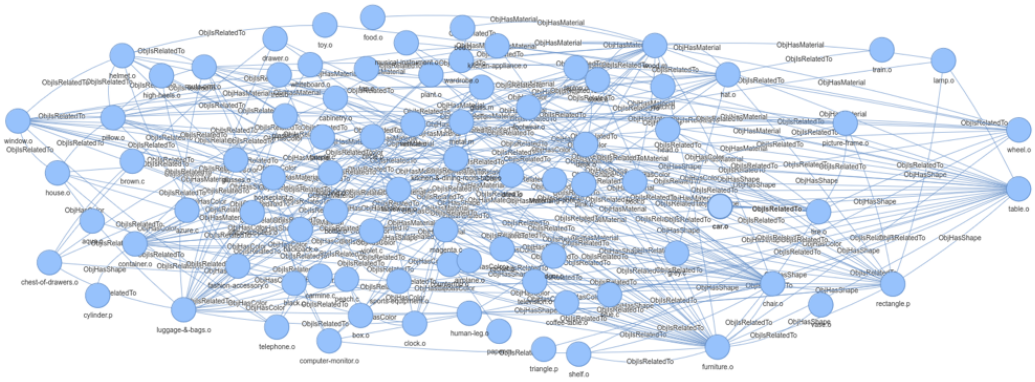


Figure 8.7. Complete knowledge graph diagram.

To tailor $\mathcal{G}$ to a specific environment, we can select a subgraph $\mathcal{S}$, where $\mathcal{S} \subseteq \mathcal{G}$. $\mathcal{S}$ contains the relevant entities and relations for the particular scene that the agent is interacting with. For example, in a scene where the objects are a mug, a bottle, and a cereal box, $\mathcal{S}$ might include nodes representing these objects and edges representing their properties or relationships (e.g., *(mug, hasAttribute, handle)*, *(bottle, hasShape, cylindrical)*, *(cereal box, hasColor, brown)*) as shown in Figure 8.8. In our approach, $\mathcal{G}$ is pre-built, and $\mathcal{S}$ is manually constructed based on the objects present in the environment, but future work could involve automating this process with a subgraph selector informed by a semantic segmentation model that can automatically generate $\mathcal{S}$ based on visual scene analysis. Once $\mathcal{S}$ has been defined, we employ GloVe [PSM14], a pre-trained unsupervised learning model, to translate $\mathcal{S}$ into a continuous vector space. GloVe generates word embeddings by leveraging global co-occurrence statistics from a large corpus, capturing semantic relationships through the likelihood of words appearing together within given contexts. Although GloVe was originally trained on general language corpora, it provides meaningful embeddings for our domain-specific text representations by associating words based on semantic similarity.

In our approach, labels from entities and relations in $\mathcal{S}$ are concatenated into a textual format, such as "item mug, hasColor red, hasAttribute handle." These text-based representations are then processed by GloVe to generate embeddings for each entity and relationship. By using GloVe as a pre-trained model off-process, we obtain embeddings that encapsulate underlying semantic relationships in $\mathcal{S}$ without requiring additional in-domain training. This embedding process results in vector representations, known as KGEs, that preserve the relational structure of our subgraph and can be seamlessly integrated into the agent's learning pipeline. Specifically, the embeddings are concatenated with the activations from the visual observation layers before the policy approximation block. In the architecture, this occurs after the first FC layer, allowing the agent to incorporate both visual and semantic information, enhancing its decision-making

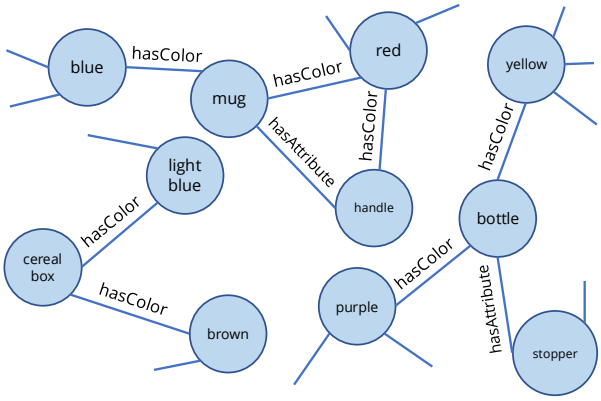


Figure 8.8. Knowledge subgraph diagram.

capabilities in the environment. We placed it there and not before the FC layer because it gives a higher abstraction level to the activations coming from the CNNs.

Figure 8.9 presents the overall view of the methodology and our proposal that can be summarized as follows:

1. Select a subgraph $\mathcal{S}$ from the overall knowledge graph $\mathcal{G}$, which represents the current environment observation perceived by the agent.

2. Transform $\mathcal{S}$ into a textual format by concatenating the node labels and embedding this representation using the GloVe model into KGEs. Note that these KGEs are invariant during the agent's training process.

3. Incorporate the resulting embeddings into the DRL agent's architecture by concatenating them before the policy approximation block.

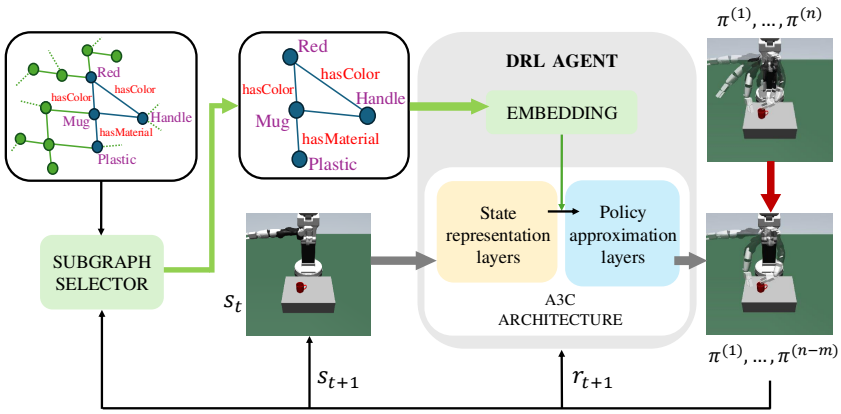


Figure 8.9. Proposed framework diagram. The process begins with a complete graph representing all entities and relationships within the environment. A subgraph selector is then employed to identify and extract the most relevant entities and relationships. This subgraph is embedded and integrated into the DRL agent's architecture by concatenating it with the layer preceding the policy approximation blocks. This integration leads to significant improvements in both learning speed and accuracy, while also reducing the amount of experience needed.

We argue that this implementation allows the DRL agent to process both the raw visual data and the contextual information provided by the KGEs, improving the agent's understanding of the environment. Figure 8.10 depicts the architecture with this modification.

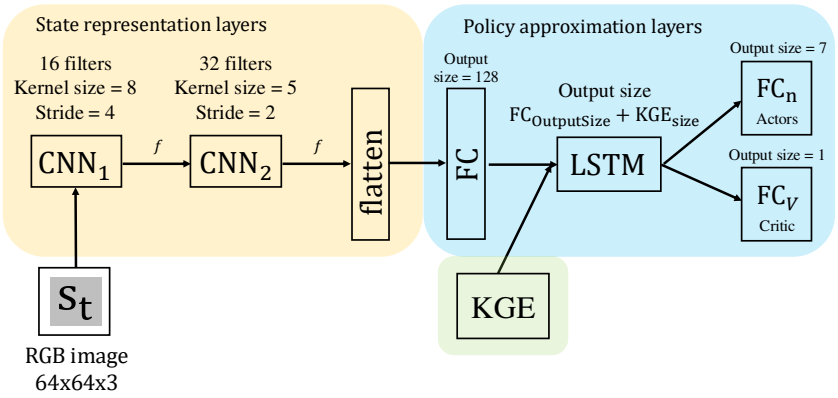


Figure 8.10. Proposed A3C architecture to integrate the KGEs in the learning process. The size of the KGE is determined by the number of encoded features. The output is distributed into seven actors for the TIAGo robot and six actors for the IRB120, in addition to the critic value.

8.3.2. Training and evaluation procedure

All experiments remain in a sim-to-sim approach to better control the external drivers that might hinder the agents' learning process and can potentially difficult the understanding of the KGE effect in the agents' performance. For this reason, we follow the same training and post-training evaluation procedures described in Section 4.2.1. Commanding the joints position, we train for 70 M steps, doing interim evaluation every 50 k steps. The post-training assessment is performed across 1,000 episodes with random initial robot and target poses.

To analyze the KGEs effect on sample efficiency issue, we examine the joints' angle distribution for those joints whose distribution presents more meaningful changes in either their standard deviation or mean. The data gathered correspond to the joints' values during the interim evaluation of the best model.

8.3.3. Description of the experiments

To exploit the KGEs potential and be able to draw meaningful conclusions about their use in DRL, we increased the environment's and MDP's complexity. First, we analyze the results in two robots, the IRB120 and the TIAGo from PAL Robotics [PMF16]. The TIAGo is a 7 DoF mobile manipulator with a two-finger gripper designed for human-robot interaction. As it has one more DoF than IRB120, the A3C output will have one additional policy estimator. Second, instead of a red cube, the reachable targets are a mug, a bottle, and a cereal box. Besides, it is not enough to hit them at a certain distance but to reach the affordance point with a specific orientation so that the gripper is ready to pick up the object. Figure 8.11 shows the graspable spots for each target as a black sphere that is only drawn for a better understanding, as in the observation captured during the training, it will not appear. Besides, Figure 8.11 illustrates the orientations the gripper must align with, where the x coordinate is the red axis, the y coordinate is green, and the z coordinate is blue. In the mug, the site is at the handle and has an orientation of 20° with respect to the ground plane; in the bottle, it is at the stopper, and the gripper has to approach it perpendicular to the ground plane; while in the box, it is at the rim above half the height and the orientation must be parallel to the ground plane. At the beginning of the episode, a target is randomly selected and placed in the workspace, while the others are not shown in the scene.

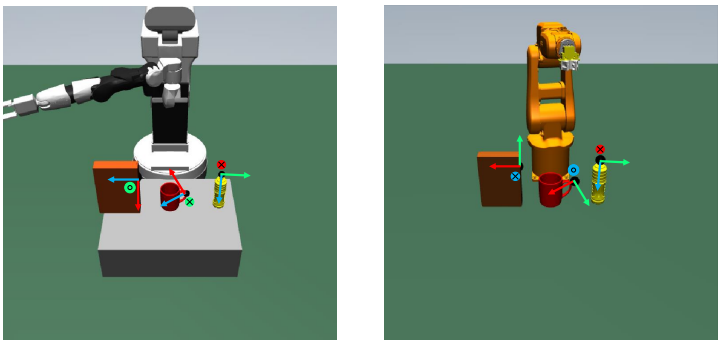


Figure 8.11. Environment observations with the three targets for both the TIAGo and IRB120 scenarios. For clarity, a black sphere has been placed at each grasping point. However, during training, the observation is reduced to a 64x64 pixel resolution, and the spheres are omitted. The site x-axis is red, the y-axis is green, and the z-axis is blue.

MDP Definition	
Ep. Length	50 steps maximum
Reward constraints	$\text{rel}_{\text{dist}} \leq 5$ cm (training) $\text{rel}_{\text{dist}} \leq 10$ cm (post-training) $\text{rel}_{\text{deg}} \leq 15°$ (training) $\text{rel}_{\text{deg}} \leq 20°$ (post-training)

Reward configuration	$-2 * \text{rel}^2_{\text{dist}} - \text{rel}_{\text{deg}}/70$
	if ($\text{rel}_{\text{dist}} > 5$ cm or $\text{rel}_{\text{deg}} > 15°$)
	100
	if ($\text{rel}_{\text{dist}} \leq 5$ cm and $\text{rel}_{\text{deg}} \leq 15°$)

Action set	0 ±MPI ±MPI/10 ±MPI/100
Initial robot config	$U(15\%\text{WRLL}, -15\%\text{WRUL})$ for joints 1 and 2
Initial target config	x $\sim U(x_{min}, x_{max})$ y $\sim U(y_{min}, y_{max})$

Table 8.2. MDP definition for the problem solved with semantic knowledge. rel_{dist} and rel_{deg} represent the relative distance and degrees between the gripper and the grasping point. MPI is the Maximum Position Increment. WRLL and WRUL are the Working Range Lower Limit and the Working Range Upper Limit of the robots. The target position is selected according to a uniform distribution.

Regarding the MDP definition, Table 8.2 gathers the new conditions. We kept the same episode length, action set, and initial robot and target configurations but introduced the new orientation conditions into the reward configuration and constraints.

To evaluate the impact of the integration of semantic knowledge on the DRL agent learning process, we carry out two sets of experiments. In the first one, the targets have only one possible color. The mug is red (1.0, 0.0, 0.0), the bottle is yellow (1.0, 1.0, 0.0), and the box is brown (0.55, 0.27, 0.07). In the second one, we applied DR to the targets' color so that the mug can be red or blue (0.0, 0.0, 1.0), the bottle can be yellow or purple (0.4, 0.0, 0.9), and the box can be brown or light-blue (0.5, 0.7, 0.9). In both sets, we train three agents using different contextual information. There is a BM agent with no KGE concatenated, a partial KGE agent where the information in the embedding gathers only the target types, and a full KGE agent that

integrates knowledge about the type and color or possible colors of the targets depending on the experiment set. Recalling Figure 8.10, depending on the information embedded, the KGE dimension varies from a vector with size 150, when the embeddings contain the object type or the object type plus just one color, to 300, when the information encoded has the object type plus the two possible colors when DR is applied. By examining the joints' angles distribution, we complement the experiments with a quantitative and qualitative analysis of the KGE in the agents' efficiency to reach with fewer samples an optimal policy.

8.3.4. Results and discussion

8.3.4.1. Experiments without DR

Figure 8.12 displays the training curves for the TIAGo under the setup where no DR is applied. It can be seen that in the TIAGo, the full KGE agent learns slightly faster and has a higher average return than the partial KGE agent that has a slower transitory phase. The full KGE agent attains 72 % success rate, similar to the 70 % accuracy obtained by the partial KGE agent. These differences regarding the learning time and accuracy are higher with the BM, as this agent only attains 60 % success rate, remaining its average return below the full KGE curve during all the process. Therefore, in the TIAGo setup with invariant features, the improvement derived from the use of KGEs is 12 % in the accuracy and 25 % in the learning time between the full KGE agent and the BM. These results are summarized in the first row of Table 8.3.

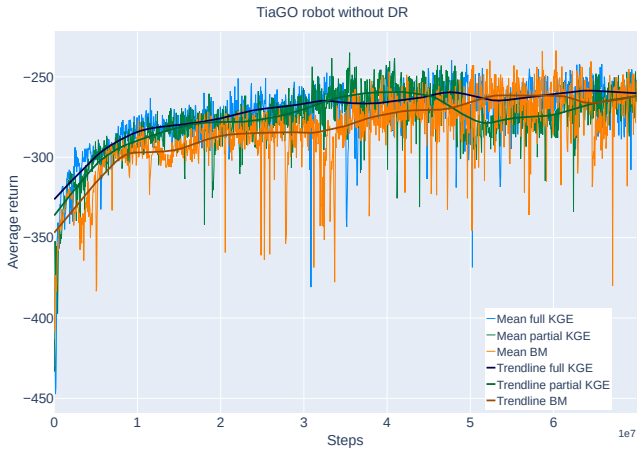


Figure 8.12. Average returns in the interim evaluations of the full KGE model (blue), the partial KGE model (green), and the BM (orange) trained with the TIAGo and fixed targets' colors over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure).

Figure 8.13 shows the training process with the IRB120 when the targets' color still invariant. Even though the gap between agents seems to be larger than in the TIAGo, especially in the middle of the training, the transitory phase of the full KGE and partial KGE agents are very similar. Their best models obtained almost the same accuracy, 92 % and 91 %, respectively. Comparing these results with the BM, its best model reaches 80 % success, resulting also in 12 % accuracy improvement when using the full KGE to complement the raw visual observations. Regarding the improvement in the learning time, the full KGE agent and BM achieve the best models at the same time, although the full KGE trendline is always above the BM, and its transitory is much faster. These results are summarized in the second row of Table 8.3.

We argue that TIAGo agents tend to achieve lower accuracy compared to the IRB120 agents due to the increased difficulty of their environment. We suggest that factors such as the TIAGo's

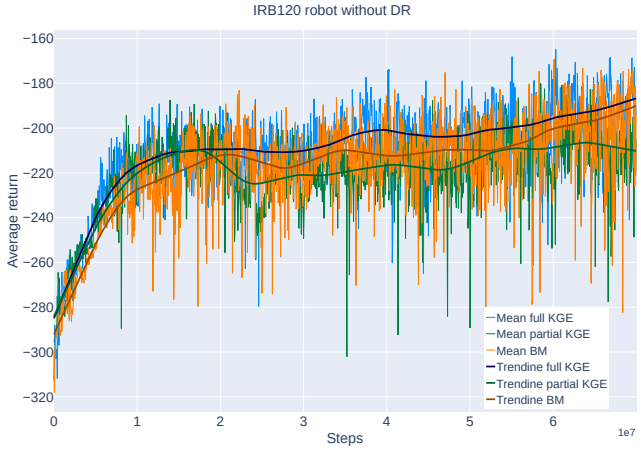


Figure 8.13. Average returns in the interim evaluations of the full KGE model (blue), the partial KGE model (green), and the BM (orange) trained with the TIAGo and fixed targets' colors over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure).

arm morphology, the additional degree of freedom compared to the IRB120, its reach, and its color contribute to the heightened complexity of the task, even though the task remains the same. Despite this, we consistently observe that there is an improvement in the transitory regime and a decrease in the learning process time. Comparing the performance of the full KGE and partial KGE agents, we see that there is almost no difference. This might demonstrate that incorporating in the embeddings the color specification does not significantly impact the agents' performance when color does not vary.

After analyzing the learning process and accuracy outcomes, we examine the distribution of the joint values during the interim evaluation of the best model. The first column of Figure 8.14 represents the joint angle distributions for the TIAGo, while the second column are the distributions for the IRB120. In both cases, we focus on the comparison between the full KGE (blue) and the BM (orange). In the TIAGo, joints 3 and 4 present significant changes, while in the IRB120 are joints 2 and 3. TIAGo's joint 3 presents a 0.01 reduction in its std. dev. plus a large mean variation. Conversely, joint 4 has a large std. dev. decrease of 0.21 points plus a mean shift. Besides, the distribution shape changes to a bimodal shape which might imply a policy variation. These outcomes may indicate that under these conditions with no DR, the KGE agent needs less experience than the BM agent and, hence, learns faster. On the other hand, IRB120's joints 2 and 3 have a std. dev. drop of 0.02 points, plus a relevant mean shift in joint 3. We argue that the absence of meaningful changes in the IRB120 standard deviations may be coherent with the fact that the BM is already close to mastering the task, so the difference in the amount of experience needed is not that relevant.

Figure 8.14. Joint angle distributions for the best full KGE agent (blue), and the best BM agent (orange) without DR. The first column corresponds to the TIAGo, and the second to the IRB120. The selected joints for TIAGo are joints 3 and 4, and for the IRB120, they are joints 2 and 3.

8.3.4.2. Experiments with DR

Figure 8.15 shows the training curve for the TIAGo when DR is applied to the targets' color. The full KGE agent achieves 79 % accuracy, higher than the 62 % obtained with the partial KGE agent, and the 63 % of the BM. Concerning the learning times, the full KGE agent achieves a reduction in the learning time needed by almost 60 % with respect to the BM. Besides, as in the experiments without DR, the transitory phase of the full KGE agent is consistently better. In this case, where the problem complexity has notably increased due to the DR implementation, there is a more relevant performance improvement when the full KGE, with the target type and its colors, is integrated. These results are summarized in the third row of Table 8.3.

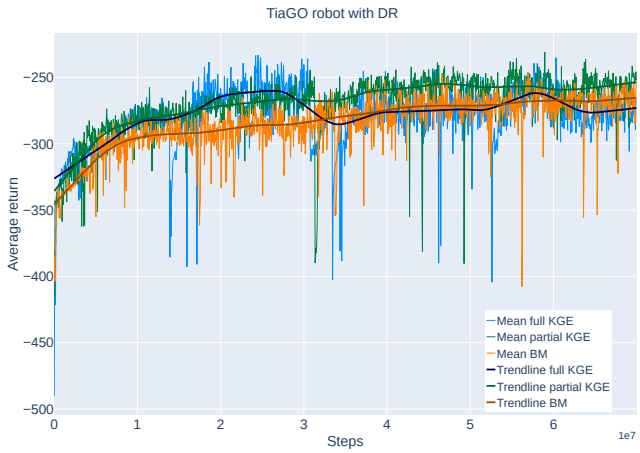


Figure 8.15. Average returns in the interim evaluations of the full KGE model (blue), the partial KGE model (green), and the BM (orange) trained with the TIAGo and DR applied over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure).

Experiment set	Robots	Agent type	Accuracy (%)	Best model step
Without DR	TIAGo	BM	60	60 M
		Partial KGE	70	**36 M**
		Full KGE	**72**	48 M
	IRB120	BM	80	60 M
		Partial KGE	91	**59 M**
		Full KGE	**92**	60 M
With DR	TIAGo	BM	63	57 M
		Partial KGE	62	59 M
		Full KGE	**79**	**24 M**
	IRB120	BM	76	56 M
		Partial KGE	**87**	43 M
		Full KGE	86	**24 M**

Table 8.3. Experiments made to analyze the KGE influence in the DRL agent learning process, using TIAGo and IRB120 robotic arms. There are two sets: without DR and with DR. Each set includes three types of agents: BM, Partial KGE agent, and Full KGE agent.

Analyzing the IRB120 full KGE agent and the BM training process in Figure 8.16, we also observe a faster learning process in the first agent. Whereas the full KGE model achieves a 86 %, the BM cannot surpass 76 %. However, with this robotic arm, the partial KGE agent obtains 87 % success rate. We suggest that the color information encoded in the KGE does not increase the accuracy due to the specific colors present in this scenario. Since the robot is orange, the RGB filters for the green and, particularly, the blue channels are less relevant than those for the red channel. This imbalance in the learned visual features may result in the partial exclusion of color information embedded in the KGE. In contrast, this issue does not occur with the TIAGo, as its white arm allows it to utilize the filters for all three RGB channels more evenly. These results are summarized in the fourth row of Table 8.3.

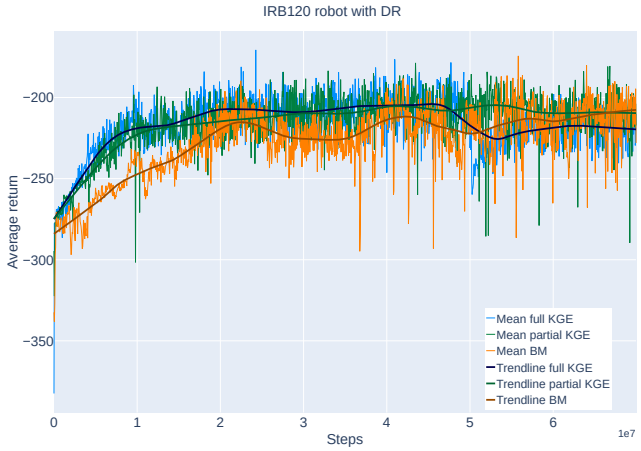


Figure 8.16. Average returns in the interim evaluations of the full KGE model (blue), the partial KGE model (green), and the BM (orange) trained with the IRB120 and DR applied over 70 M steps (note the 1e7 scaling factor on the right-hand corner of the figure).

Note that the full KGE agent presents a performance decrease in both setups, although the distance with the other models is later recovered. We suggest this happens because of the internal learning dynamics or a high learning rate, as we have already observed throughout this thesis dissertation in the training process of other agents. Table 8.3 summarizes all the accuracy results of the trained agents with and without DR.

Examining the joint angle distributions, the first column of Figure 8.17 illustrates the TIAGo's joint angle distributions, and the second column the IRB120's distributions, where the full KGE agent corresponds to the blue curve and the BM to the orange. In this experiment set, for both the TIAGo and the IRB120, the most relevant change is a considerable drop in the distribution std. dev.. Whereas in the former this happens in joints 2, 4, and 5; in the latter, it occurs in joints 1, 2, and 4. As in the previous experiments, these results suggest that the information encoded in the embeddings may involve an improvement in the sample efficiency, requiring less exploration and quickly exploiting the correct policies. This aligns with the fact that the full KGE agents learn faster than the BM agents. Besides, the std. dev. decrease is more pronounced and affects a greater number of joints compared to scenarios without DR, likely due to the increased complexity of the problem.

Figure 8.17. Joint angle distributions for the best full KGE agent (blue), and the best BM agent (orange) with DR. The first column corresponds to the TIAGo and the second to the IRB120. The selected joints for TIAGo are joints2, 4 and 5, and for the IRB120, they are joints 1, 2, and 3.

8.4. Conclusion

In this chapter, we have validated the sim-to-real methodology designed with the IRB120 in another industrial asset, the UR3e collaborative robotic arm. The proposed pipeline consists of the use of DA to train an original CycleGAN implementation, the SICGAN, for image-to-image translation. This way, we can define a pre-processing stage where the raw virtual observation is translated to a real-synthetic image that feeds the agent in the virtual environment. After training the agent with these modified observations, we can perform a successful zero-shot in the real setup. The results obtained with the UR3e demonstrate that despite the fact that the robot's morphology and its colors make the problem more complex, the different models, architectures, and, in general, the methodology are robust enough to efficiently address the sim-to-real transfer.

In addition, we defined the general procedure that has to be followed if the industrial application to be tackled with DRL is different from the control and manipulation with robotic arms. Briefly, we have to virtualize the environment, making it relatively similar to the real scenario, to later define the MDP and test it by training the agent in the virtual setup. Once we check that our agent is able to learn, we should train a SICGAN to perform the image-to-image translation between domains. With the DA model trained, we can re-train the virtual agent but with the real-synthetic observations resulting from passing the raw virtual environment images through the SICGAN. Finally, we can perform the zero-shot transfer in the real setup. Although it would be presumptuous to say that this methodology opens the door for solving with DRL any industrial problem, we believe that it is widely applicable and robust enough to deal with most of the sequential decision-making problems that can be monitored using a visual input.

We have also presented our last work in collaboration with the RoCoCo research group from Sapienza University of Rome, where we investigate the impact of introducing semantic knowledge about the environment into the agent's learning process to complement its visual scene understanding providing some notions of commonsense. We have demonstrated that adding contextual information after the state representation learning layers substantially improves the agents' learning time and accuracy. Additionally, we have observed how the use of KGE has a positive effect on the sample efficiency, leading to a shrink of the empirical distribution of the joint angles and, thus, a more efficient learning. We leave as future research lines, analyzing the impact of concatenating the embedding in other architecture layers, how we can extract the subgraph relying on an image segmentation model, and the evaluation of the KGEs influence when transferring the agent to the real scenario.

9

Conclusions, Contributions and Future Work

One never notices what has been done;
one can only see what remains to be done.
Marie Curie (1867–1934)

This final chapter provides an overview of the progress made throughout this thesis. It presents the key conclusions drawn from the experiments conducted and highlights the most novel contributions. Additionally, it addresses the unresolved issues not covered in the thesis and explores potential directions for future research.

9.1. Summary and conclusions

This thesis has proposed a methodology suitable for industrial applications to solve the sim-to-real problem by transferring efficiently the DRL experience. The proposal is the result of having investigated various transfer learning techniques. According to the existing work on how to bridge the reality gap, we focused on those techniques more appropriate for an industrial setting. As a result, we selected Domain Randomization (DR), Progressive Neural Networks (PNNs), and Domain Adaptation (DA) as the most adequate transfer learning approaches for our setting. Additionally, we explored semantic knowledge to enhance the agent's performance by integrating contextual information about the environment.

We address the approach to a target that is randomly placed in the workspace, as the first stage of a pick-and-place industrial operation, using first the IRB120 industrial robotic arm and then the UR3e collaborative robotic arm to validate the designed methodology. The only input given to the agent is the environment's 64x64 RGB image.

Regarding the use of DR to introduce variability into the agents' learning process, so the target domain samples are perceived as coming from a known distribution seen during the training phase, we believe it is really interesting to enhance the agents' robustness and generalization capabilities, but in relatively complex environments, it should be used as a complement to other

techniques as the number of features that can be randomized is unaffordable. Guided by this rationale, we investigated first the use of DR in a high-level approach, where attributes like the camera pose, scene colors, and the target size and shape vary during the training, and second, a low-level (i.e., pixel level) DR implementation where the variability is introduced as Gaussian noise in the image with the aim of resembling the virtual and real observations RGB histograms. The high-level DR experiments proved that within the same effort, i.e., training time, the agent improve its robustness and generalization capabilities and, on the other hand, that randomizing more than a certain number of attributes leads to an increase in the environment's complexity that the agent cannot handle properly. One of the concluding remarks from the low-level DR trials is that there is an optimal noise level at which the agent reaches its best performance when, after being virtually trained, it is transferred directly to reality in a zero-shot transfer. Before and after this noise threshold, its performance in the real environment is inferior. Besides, the addition of Gaussian noise to the raw virtual observations results in an accuracy improvement of almost 20 % in the zero-shot transfer compared to the performance when the agent was trained with the noiseless virtual images. However, the final accuracy of 34.1 % was far from being considered a successful sim-to-real transfer.

After studying DR as a complementary technique, PNNs were examined as a few-shot transfer learning technique that allows a rapid transition from the virtual to the real environment but requires the collection of some real experience to fine-tune the agent's policy. Before the sim-to-real stage, we carried out sim-to-sim experiments to assess the PNNs' perception ability, due to the visual nature of the problem solved, and their forgetting with respect to the teacher's task when the student network is fine-tuned to solve a different task. The most remarkable insights were that there is a clear performance decay up to 50 % when the agent is exposed to visual changes in the scene, like the color of some elements, and, concerning the forgetting, student agents tend to partially forget the knowledge already acquired and frozen in the teacher column when they master the new task, being the worst performance decay of 10 %. With respect to the use of PNNs as a sim-to-real technique, we present four strategies to bridge the reality gap that differs on the teacher agents used. The first interesting outcome was that the agent with the teacher trained with the raw virtual observations presented a better sim-to-real transfer than the agent with the teacher trained with the noisy observations from the low-level DR integration, despite the fact that individually, when we performed a zero-shot transfer of the agents, the second one achieved better results. We suggest that this may happen because the PNN architecture did not promote large changes settling for values far from good behavior. The successful sim-to-real transfer was achieved after gathering 60 k samples, which is on-pair with the state-of-the-art results. The post-training evaluation showed that although the agent struggled when the goal was placed in a certain region of the workspace, it was able to reach accuracies above 80 % in most of the zones, and generalize the learned knowledge with the AR targets when we place real objects like LEGO® cubes with different colors or a mug.

Looking for a zero-shot transfer that might be more efficient than a few-shot solution as PNNs, we explored the image-to-image translation with an original implementation of the Cycle-Consistent Generative Adversarial Network (CycleGAN), the SICGAN, which transforms raw virtual observations to real-synthetic images, and raw real observations to virtual-synthetic images. As we are focusing on industrial applications, we decided not to rely heavily on the hardware resources available in the industrial settings, so the developed methodology advocates for training in the virtual environment with real-synthetic observations, which are obtained by passing the virtual images through the previously trained SICGAN, and then performing zero-shot transfer of the agent using directly the real observations. The post-training

evaluation demonstrated that, overall, the agent's performance was better than the PNN-like agent, achieving also a more balanced accuracy distribution over the workspace. As in the few-shot approach, we validated the robustness and generalization capabilities of the deployed agent when evaluated with the AR and real targets, improving also the PNN-like agent results.

As a result, our final sim-to-real methodology and the one validated with the UR3e in Chapter 8 was based on DA and the image-to-image translation between domains. However, this does not mean that the PNN approach was invalidated. Indeed, we believe that there is a really promising research field in the integration of both techniques plus DR to enhance the agent's ability to learn several tasks in the virtual environment and perform zero-shot transfers in the real setup.

We finally tested the use of semantic knowledge to improve the agent's learning process in terms of time and return obtained. The experiments were performed under a sim-to-sim approach defining two sets, one without DR and another with DR applied to the targets' colors. The results demonstrate that in both cases, the addition of contextual information as an input in the form of a Knowledge Graph Embedding (KGE) positively impacts the agent success rate, leading to improvements up to 60 % in the learning time and 12 % in the accuracy.

9.2. Original contributions

The main objective of this thesis was to define, implement, and validate a methodology that efficiently transfers experience between DRL agents to bridge the reality gap. Hence, the main original contributions of this dissertation are:

- A general methodology to efficiently transfer experience between DRL agents trained in different domains. The proposed pipeline was developed and validated using two different robotic arms and, owing to the use of visual inputs, the procedure can be extrapolated to most of the industrial settings.
- The quantification of the robustness improvement in a teacher PNN-like agent when DR is applied in a high-level approach to several features, either alone or combining them, and in a low-level strategy in the form of Gaussian noise to the environment observations to boost the sim-to-real transfer. For these evaluations, we used an original benchmark that can be used with virtual, real experience, or both.
- The quantification and assessment in a sim-to-sim approach of 1) a teacher PNN-like agent robustness when it faces high-level modifications in visual attributes, 2) PNN-like agents learning dynamics when they are fine-tuned in environments of varying difficulty, and 3) the forgetting PNN-like agents suffer when trained on a new task.
- The implementation, analysis, and validation of a few-shot sim-to-real methodology based on the PNNs architecture and a zero-shot sim-to-real methodology based on DA, both suitable for industrial applications.
- An original CycleGAN implementation named StyleID-CycleGAN (SICGAN), which solves the appearance of artifacts by integrating the improvements of StyleGANv2 and the Identity Loss in the generators' loss function.
- The validation of using AR to efficiently train and evaluate agents in real environments, which potentially leads to an improvement of the agents' generalization capability when facing real targets.

- An original DRL architecture that integrates contextual information from the environment in the form of KGEs to enhance the agents' learning process in terms of time and accuracy, as well as the quantification of the DRL agents' performance improvement when total or partial semantic information is included as an input to the agents, as well as its effect on the agents' exploration and exploitation abilities.

The result from this thesis that has already been published in a journal article is:

- L. Güitta-López, J. Boal, and Á. J. López-López, "Learning more with the same effort: How randomization improves the robustness of a robotic deep reinforcement learning agent", Applied Intelligence, 2022, ISSN: 1573-7497. DOI: 10.1007/s10489-022-04227-3.

Additionally, at the time of writing, there are the following journal articles under review and working papers:

- L. Güitta-López, J. Boal, and Á. J. López-López, "Evaluating the perception, understanding, and forgetting of Progressive Neural Networks: a quantitative and qualitative analysis", under review at the moment of writing.
- L. Güitta-López, V. Suriani, J. Boal, Á. J. López-López, D. Nardi, "Boosting Deep Reinforcement Learning Efficiency with Knowledge Graph Embeddings for Robotic Applications", 39th Annual AAAI Conference on Artificial Intelligence, submitted, pending on decision.
- L. Güitta-López, L. Güitta-López, J. Boal, and Á. J. López-López, "SICGAN-Driven Sim2Real Transfer: Zero-Shot Deployment on Robotic Manipulators through Visual Deception", working paper.
- L. Güitta-López, J. Boal, and Á. J. López-López, "Optimizing few-shot Sim2Real transfer: exploring Progressive Neural Networks and the role of the teacher column for robotic arm tasks", working paper.

9.3. Future work

The investigation presented throughout this dissertation is just the beginning of a promising path toward the development of a methodology that efficiently transfers experience between DRL agents appropriate for industrial settings. As a result of the work done, new questions and doors have opened that might be interesting to address in future research:

- Integration of DR and PNNs: once demonstrated that high-level DR enhances the agents' robustness and their generalization capability, the next step will be to investigate how this affects the sim-to-real transfer if the PNN teacher column is an agent trained with that diversity, or if multiple teachers are trained with high-level DR separately and then a student reuses that knowledge.
- PNNs implementation: the PNN architecture implemented is deeply based on [Rus+17], but it has some details that were not specified in the original work. For instance, the way lateral connections are added to the next-column hidden activations is not completely solved. Hence, there is still room for analyzing the PNN-like agent performance under different integrations of the lateral connections addition. On the other hand, the partial forgetting discovered on PNN-like agents with respect to the teacher's task should be addressed either with a voting system that, depending on a variable, decides which

column defines the policy, or introducing episodes from the teacher's task during the training of the student.

- Few-shot sim-to-real transfer using PNNs: as presented in the results, although the zero-shot transfer with the agent that was trained with the noisy virtual observation was much better than the zero-shot transfer with the agent trained under raw virtual observations, it is interesting that the most successful sim-to-real transfer using PNNs occurs when the teacher is the agent with the raw visual inputs. Although we have made a plausible assumption of the reasons that might motivate this behavior, a thorough analysis to compare the learning processes will be convenient. Besides, due to the behavior of the PNN-like agent in the post-training evaluation in the real setup, it will also be recommendable to examine why it performs the same steps at the episode beginning regardless of where the target is.

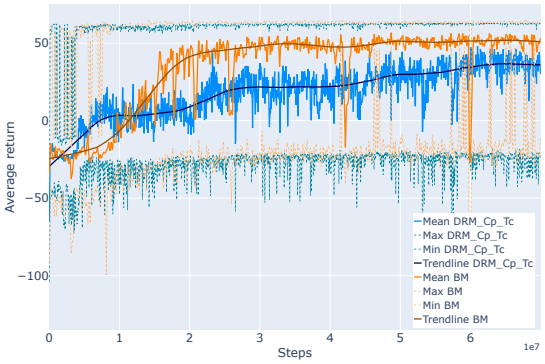
- Zero-shot sim-to-real transfer using DA: regarding the results obtained with the real targets, we observed that the agent struggled to approach the target when the object was blue. As this behavior was already observed in the adversarial environments definition and considering the results obtained when high-level DR was applied in the virtual scenario, introducing this variability during the virtual training of the agent will enhance its performance in the real setup. On the other hand, as we have stated, the fact that we have opted for one technique as the most efficient does not mean that PNNs should be completely discarded. Indeed, there is a promising research line that will consist of training several agents from the same PNN architecture to develop different tasks in the virtual environment and then perform a zero-shot transfer in the real setup using DA. This way we can think of a single agent able to solve different problems. Finally, although we believe that the translation from the raw virtual observation to the real domain is the most efficient strategy for an industrial application, the analysis and evaluation of the agent's performance when this path is visited the other way around might also result in relevant insights.

- Integration of semantic knowledge: based on some design decisions that, in the end, led to good results, there is still an analysis to be done regarding the possible locations of the embedding in the architecture. With respect to the subgraph selector, currently, it is done manually, but with the vast amount of pre-trained image segmentation models, we can think of using one of them to determine the important environment entities and extract the subgraph automatically. Finally, as this research remained in a sim-to-sim approach, the outcomes and conclusions should be validated with experiments after a sim-to-real transfer.

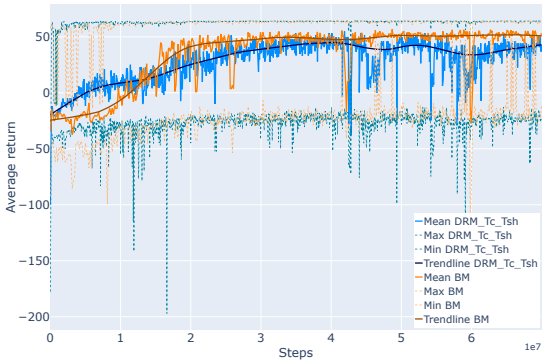
A

Domain Randomization applied to a combination of features

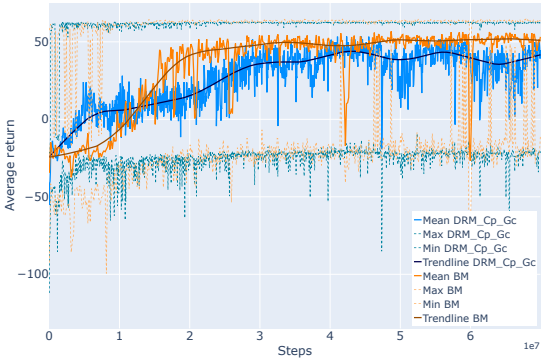
This appendix presents the figures with the average returns obtained in the interim evaluation of the DR agents trained with several features randomized.



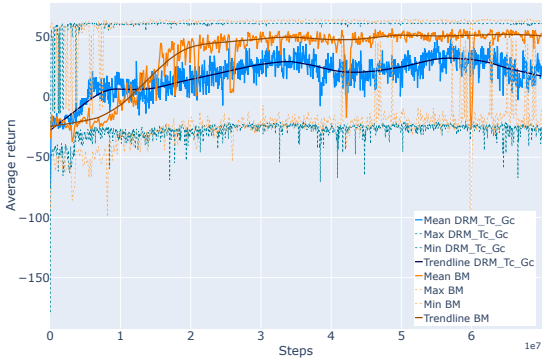
(a) Average returns in the interim evaluation for the BM (orange) and DRM_{CpTc} (blue) over 70 M steps.



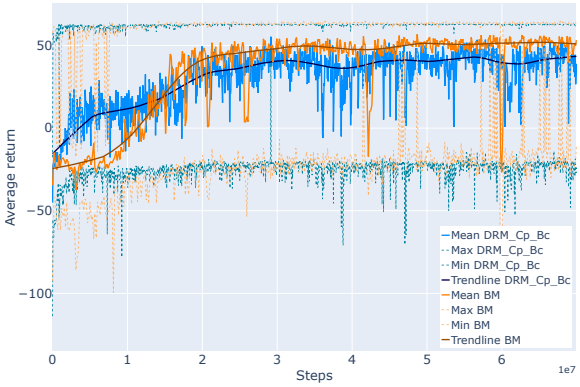
(b) Average returns in the interim evaluation for the BM (orange) and $\text{DRM}_{\text{TcTsh}}$ (blue) over 70 M steps.



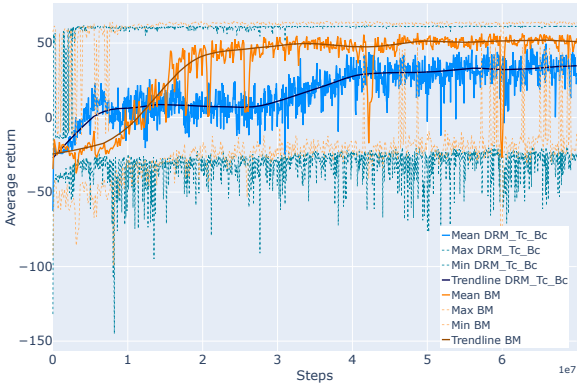
(c) Average returns in the interim evaluation for the BM (orange) and DRM_{CpGc} (blue) over 70 M steps.



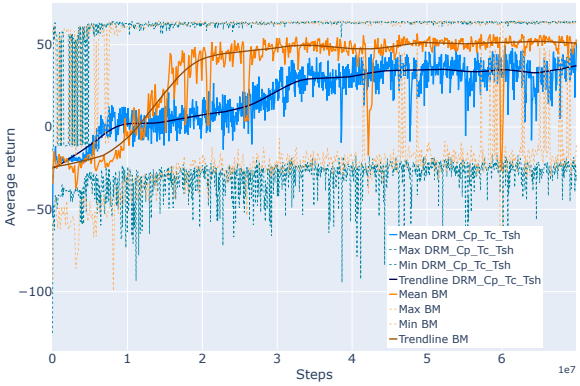
(d) Average returns in the interim evaluation for the BM (orange) and DRM_{TcGc} (blue) over 70 M steps.



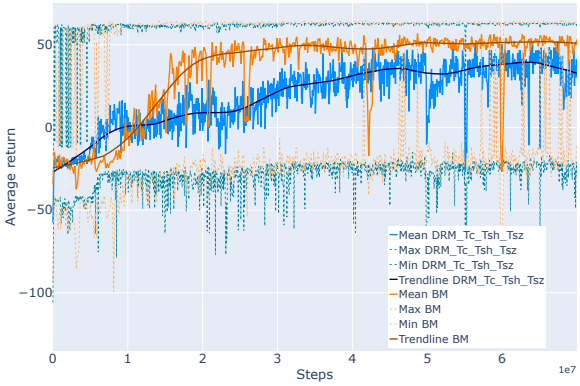
(e) Average returns in the interim evaluation for the BM (orange) and DRM_{CpBc} (blue) over 70 M steps.



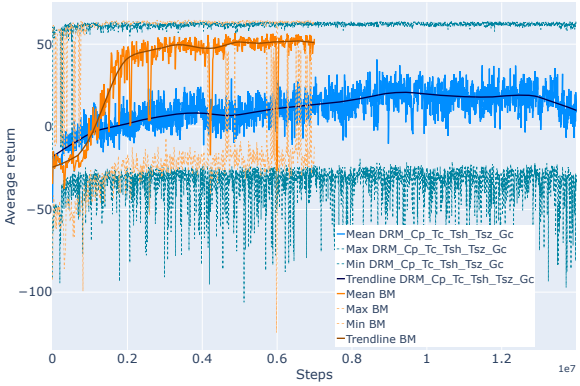
(f) Average returns in the interim evaluation for the BM (orange) and DRM_{TcBc} (blue) over 70 M steps.



(g) Average returns in the interim evaluation for the BM (orange) and $\text{DRM}_{\text{CpTcTsh}}$ (blue) over 70 M steps.



(h) Average returns in the interim evaluation for the BM (orange) and $\text{DRM}_{\text{TcTshTsz}}$ (blue) over 70 M steps.



(i) Average returns in the interim evaluation for the BM (orange) and $\text{DRM}_{\text{CpTcTshTszGc}}$ (blue) over 70 M steps for the former and 100 M steps for the later.

Figure A.1. Average returns in the interim evaluation when DR is applied to a combination of features.

B

Detailed results of the trained MDP models

This appendix presents the table with the detailed results of all the trained MDP models in the post-training evaluation.

Table B.1. Post-training evaluation results for all the MDP models. The model selected is highlighted in bold.

Model	Rewarding distance (cm)	Mean return	std. dev. return	Mean episode length	std. dev. episode length	Average accuracy (%)
M1	5	46.23	3.58	33.12	1.20	86.6
	10	55.24	1.14	26.14	0.71	99.4
M2	5	52.72	2.36	30.72	0.92	**95.7**
	10	55.80	0.62	25.40	0.57	**100.0**
M3	5	60.99	3.14	30.58	0.84	96.9
	10	74.22	1.34	25.71	0.61	99.7
M4	5	53.62	6.13	32.35	1.12	90.1
	10	81.25	3.11	25.87	0.64	99.8
M5	5	47.79	3.90	32.45	1.27	89.0
	10	55.39	1.03	25.88	0.73	99.6
M6	5	51.00	2.90	34.57	1.01	93.6
	10	55.01	1.05	28.38	0.72	99.6
M7	5	56.58	4.14	32.73	1.08	91.2
	10	73.35	1.50	26.17	0.61	99.9

References

[ABB19] ABB Robotics, "ABB IRB 120 Robot Datasheet", ABB, Tech. Rep., 2019.

[AD21] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress", *Artificial Intelligence*, vol. 297, Aug. 2021. DOI: 10.1016/j.artint.2021.103500.

[Ade+18] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity Checks for Saliency Maps", in *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, Montreal, Canada, 2018, pp. 9525–9536. DOI: 10.5555/3327546.3327621.

[AJ21] R. Alghonaim and E. Johns, "Benchmarking Domain Randomisation for Visual Sim-To-Real Transfer", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Xian, China, 2021, pp. 12 802–12 808. DOI: 10.1109/ICRA48506.2021.9561134.

[Aka+21] E. Akanksha, Jyoti, N. Sharma, and K. Gulati, "Review on Reinforcement Learning, Research Evolution and Scope of Application", in *Proceedings of the 5th International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India, Apr. 2021, pp. 1416–1423. DOI: 10.1109/ICCMC51019.2021.9418283.

[And+20] M. Andrychowicz *et al.*, "Learning dexterous in-hand manipulation", *International Journal of Robotics Research*, vol. 39, pp. 3–20, 1 2020. DOI: 10.1177/0278364919887447.

[Arn+20] K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki, "Meta Reinforcement Learning for Sim-to-real Domain Adaptation", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, 2020, pp. 2725–2731. DOI: 10.1109/ICRA40945.2020.9196540.

[Aru+17] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey", *IEEE Signal Processing Magazine*, vol. 34, pp. 26–38, nº6 2017. DOI: 10.1109/MSP.2017.2743240.

[Bab+17] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU", in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, Toulon, France, Nov. 2017. DOI: 10.48550/arXiv.1611.06256.

[BDM17] M. G. Bellemare, W. Dabney, and R. Munos, "A Distributional Perspective on Reinforcement Learning", in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, ser. ICML'17, vol. 70, Sydney, Australia, 2017, pp. 449–458. DOI: 10.5555/3305381.3305428.

[Bel57] R. Bellman, "A Markovian Decision Process", *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[Bis06] C. M. Bishop, *Pattern Recognition and Machine Learning*. 2006, ISBN: 978-0-387-31073-2.

[Bou+18] K. Bousmalis *et al.*, "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, 2018, pp. 4243–4250. DOI: 10.1109/ICRA.2018.8460875.

[Bro+16] G. Brockman *et al.*, "OpenaI Gym", *Computing Research Repository (CoRR)*, 2016. DOI: 10.48550/arXiv.1606.01540.

[Cha+15] A. X. Chang *et al.*, "ShapeNet: An Information-Rich 3D Model Repository", Stanford University, Princeton University, and Toyota Technological Institute at Chicago, Tech. Rep., 2015. DOI: 10.48550/arXiv.1512.03012.

[Che+19] Y. Chebotar *et al.*, "Closing the sim-to-real Loop: Adapting simulation randomization with real world experience", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2019-May, Montreal, Canada, 2019, pp. 8973–8979. DOI: 10.1109/ICRA.2019.8793789.

[Che+21] X.-H. Chen, S. Jiang, F. Xu, Z. Zhang, and Y. Yu, "Cross-Modal Domain Adaptation for Cost-Efficient Visual Reinforcement Learning", in *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS)*, Online, 2021, pp. 12 520–12 532.

[Che+22] Y. Chen, X. Li, S. Guo, X. Y. Ng, and M. Ang, "Real2Sim or Sim2Real: Robotics Visual Insertion using Deep Reinforcement Learning and Real2Sim Policy Adaptation", in *Proceedings of the 17th International Conference on Intelligent Autonomous Systems (IAS)*, Zagreb, Croatia, Jun. 2022. DOI: 10.1007/978-3-031-22216-0_41.

[CL18] Z. Chen and B. Liu, *Lifelong Machine Learning*, 2nd ed. Springer International Publishing, 2018, pp. 1–187, ISBN: 978-3-031-00453-7. DOI: 10.1007/978-3-031-01581-6.

[Cza+19] W. M. Czarnecki *et al.*, "Distilling Policy Distillation", in *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, Naha, Japan, 2019. DOI: 10.48550/arXiv.1902.02186.

[Dev+18] C. Devine, P. A. Abbeel, T. Darrell, and S. L. Levine, "Deep Object-Centric Representations for Generalizable Robot Learning", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, 2018, pp. 7111–7118. DOI: 10.1109/ICRA.2018.8461196.

[Dik+18] H. U. Dike, Y. Zhou, K. K. Deveerasetty, and Q. Wu, "Unsupervised Learning Based On Artificial Neural Network: A Review", in *Proceedings of the 2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, Shenzhen, China, 2018, pp. 322–327. DOI: 10.1109/CBS.2018.8612259.

[DL20] G. Davidson and B. M. Lake, "Investigating Simple Object Representations in Model-Free Deep Reinforcement Learning", in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 42, Online, 2020, pp. 2023–2029. DOI: 10.48550/arXiv.2002.06703.

[Du+21] Y. Du, O. Watkins, T. Darrell, P. Abbeel, and D. Pathak, "Auto-Tuned Sim-to-Real Transfer", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2021-May, Xian, China, 2021, pp. 1290–1296. DOI: 10.1109/ICRA48506.2021.9562091.

[Du+23] Y. Du *et al.*, "Guiding pretraining in reinforcement learning with large language models", in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, vol. 202, Honolulu, HI, USA, 2023, pp. 8657–8677. DOI: 10.48550/arXiv.2302.06692.

[FA18] J. J. Faraway and N. H. Augustin, "When small data beats big data", *Statistics and Probability Letters*, vol. 136, pp. 142–145, May 2018. DOI: 10.1016/j.spl.2018.02.031.

[For+17] M. Fortunato *et al.*, "Noisy Networks for Exploration", *Computing Research Repository (CoRR)*, 2017. DOI: 10.48550/arXiv.1706.10295.

[Fra+18] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning", *Foundations and Trends in Machine Learning*, vol. 11, pp. 219–354, 3–4 Dec. 2018. DOI: 10.1561/2200000071.

[FvHM18] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods", in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Stockholm, Sweden, 2018. DOI: 10.48550/arXiv.1802.09477.

[Gar+14] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion", *Pattern Recognition*, vol. 47, pp. 2280–2292, 6 Jun. 2014. DOI: 10.1016/j.patcog.2014.01.005.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, ISBN: 9780262035613.

[GBL22] L. Güitta-López, J. Boal, and Á. J. López-López, "Learning more with the same effort: how randomization improves the robustness of a robotic deep reinforcement learning agent", *Applied Intelligence*, 2022. DOI: 10.1007/s10489-022-04227-3.

[Gha04] Z. Ghahramani, "Unsupervised Learning", in Springer Berlin Heidelberg, 2004, pp. 72–112, ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_5.

[Goo+14] I. J. Goodfellow *et al.*, "Generative Adversarial Nets", in *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS)*, vol. 27, Long Beach, CA, USA, 2014, pp. 2672–2680. DOI: 10.5555/2969033.2969125.

[Gro+12] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuška, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients", *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 42, pp. 1291–1307, 6 2012. DOI: 10.1109/TSMCC.2012.2218595.

[Haa14] J. K. Haas, "A History of the Unity Game Engine", 2014.

[He+15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015, pp. 770–778. DOI: 10.1109/cvpr.2016.90.

[HE16] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning", in *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS)*, Barcelona, Spain, 2016, pp. 4572–4580. DOI: 10.5555/3157382.3157608.

[Hes+18] M. Hessel *et al.*, "Rainbow: combining improvements in deep reinforcement learning", in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence and 30th Innovative Applications of Artificial Intelligence Conference and 8th AAAI Symposium on Educational Advances in Artificial Intelligence*, New Orleans, LA, USA, 2018. DOI: 10.5555/3504035.3504428.

[Hig+17] I. Higgins *et al.*, "DARLA: Improving Zero-Shot Transfer in Reinforcement Learning", in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, vol. 70, Sydney, Australia, 2017, pp. 1480–1490. DOI: 10.5555/3305381.3305534.

[Ho+21] D. Ho, K. Rao, Z. Xu, E. Jang, M. Khansari, and Y. Bai, "RetinaGAN: An Object-aware Approach to Sim-to-Real Transfer", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Xian, China, 2021, pp. 10 920–10 926. DOI: 10.48550/arXiv.2310.04517.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised Learning", in Springer New York, 2009, pp. 485–585, ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7_14.

[Hub+23] J. Huber, F. Hélénon, H. Watrelot, F. B. Amar, and S. Doncieux, "Domain Randomization for Sim2real Transfer of Automatically Generated Grasping Datasets", *Computing Research Repository (CoRR)*, Oct. 2023.

[Hus+17] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation Learning: A Survey of Learning Methods", *Association for Computing Machinery.*, vol. 50, no. 2, 2017. DOI: 10.1145/3054912.

[Iso+17] P. Isola, J.-Y. Zhu, T. Zhou, A. A. Efros, and B. A. Research, "Image-to-Image Translation with Conditional Adversarial Networks", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5967–5976. DOI: 10.1109/CVPR.2017.632.

[Jam+19] S. James *et al.*, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 12 619–12 629. DOI: 10.1109/CVPR.2019.01291.

[Jan+24] Y. Jang, J. Baek, S. Jeon, and S. Han, "Bridging the simulation-to-real gap of depth images for deep reinforcement learning", *Expert Systems with Applications*, vol. 253, Nov. 2024. DOI: 10.1016/j.eswa.2024.124310.

[Jeo+20] R. Jeong *et al.*, "Self-Supervised Sim-to-Real Adaptation for Visual Robotic Manipulation", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, 2020, pp. 2718–2724. DOI: 10.1109/ICRA40945.2020.9197326.

[Jia+21] Y. Jiang *et al.*, "SimGAN: Hybrid Simulator Identification for Domain Adaptation via Adversarial Reinforcement Learning", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Xian, China, 2021, pp. 2884–2890. DOI: 10.1109/ICRA48506.2021.9561731.

[Kad+23] Y. Kadokawa, L. Zhu, Y. Tsurumine, and T. Matsubara, "Cyclic policy distillation: Sample-efficient sim-to-real reinforcement learning with domain randomization", *Robotics and Autonomous Systems*, vol. 165, Jul. 2023. DOI: 10.1016/j.robot.2023.104425.

[Kan+17] K. Kansky *et al.*, "Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics", in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, vol. 70, Sydney, Australia, 2017, pp. 1809–1818. DOI: 10.5555/3305381.3305568.

[Kar+19] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and Improving the Image Quality of StyleGAN", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA, 2019, pp. 8107–8116. DOI: 10.1109/cvpr42600.2020.00813.

[KB01] T. Kadir and M. Brady, "Saliency, Scale and Image Description", *International Journal of Computer Vision*, vol. 45, pp. 83–105, 2 2001. DOI: 10.1023/A:1012460413855.

[KH04] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator", in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, Sendai, Japan, 2004, pp. 2149–2154. DOI: 10.1109/IROS.2004.1389727.

[Kir+23] R. Kirk, A. Zhang, M. A. Research, and T. Rocktäschel, "A Survey of Zero-shot Generalisation in Deep Reinforcement Learning", *Journal of Artificial Intelligence Research*, vol. 76, pp. 201–264, 2023. DOI: 10.1613/jair.1.14174.

[KL51] S. Kullback and R. A. Leibler, "On information and sufficiency", *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[KLT18] T. Karras, S. Laine, and A. Timo, "A Style-Based Generator Architecture for Generative Adversarial Networks", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2018, pp. 4396–4405. DOI: 10.1109/CVPR.2019.00453.

[KSA21] T. Kegyes, Z. Süle, and J. Abonyi, "The Applicability of Reinforcement Learning Methods in the Development of Industry 4.0 Applications", *Complexity*, vol. 2021, 2021. DOI: 10.1155/2021/7179374.

[KSH20] A. Kanervisto, C. Scheller, and V. Hautamäki, "Action Space Shaping in Deep Reinforcement Learning", in *Proceedings of the IEEE Conference on Games (CoG)*, Osaka, Japan, 2020, pp. 479–486. DOI: 10.1109/CoG47356.2020.9231687.

[LBH15] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, pp. 436–444, 7553 May 2015. DOI: 10.1038/nature14539.

[Lee+21] A. X. Lee *et al.*, "Beyond Pick-and-Place: Tackling Robotic Stacking of Diverse Shapes", in *Proceedings of the 5th Conference on Robot Learning (CoRL)*, London, United Kingdom, Oct. 2021. DOI: 10.48550/arXiv.2110.06192.

[Li+22a] D. Li, L. Meng, J. Li, K. Lu, and Y. Yang, "Domain adaptive state representation alignment for reinforcement learning", *Information Sciences*, vol. 609, pp. 1353–1368, Sep. 2022. DOI: 10.1016/j.ins.2022.07.156.

[Li+22b] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, pp. 6999–7019, 12 Dec. 2022. DOI: 10.1109/TNNLS.2021.3084827.

[Li+23] C. Li, P. Zheng, Y. Yin, B. Wang, and L. Wang, "Deep reinforcement learning in smart manufacturing: A review and prospects", *CIRP Journal of Manufacturing Science and Technology*, vol. 40, pp. 75–101, Feb. 2023. DOI: 10.1016/j.cirpj.2022.11.003.

[Li+24] K. Li, J. Wang, L. Yang, C. Lu, and B. Dai, "SemGrasp: Semantic grasp generation via language aligned discretization", *Computing Research Repository (CoRR)*, 2024. DOI: 10.48550/arXiv.2404.03590.

[Lil+16] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning", in *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, Y. Bengio and Y. LeCun, Eds., San Juan, Puerto Rico, 2016. DOI: 10.48550/arXiv.1509.02971.

[Loc+20] F. Locatello *et al.*, "Object-Centric Learning with Slot Attention", in *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS)*, Online, 2020, pp. 11 525–11 538. DOI: 10.5555/3495724.3496691.

[Luo+24] Y. Luo, W. Li, P. Wang, H. Duan, W. Wei, and J. Sun, "Progressive Transfer Learning for Dexterous In-Hand Manipulation with Multi-Fingered Anthropomorphic Hand", *IEEE Transactions on Cognitive and Developmental Systems*, pp. 1–12, 2024. DOI: 10.1109/TCDS.2024.3406730.

[LW16] C. Li and M. Wand, "Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks", in *Proceedings of the Computer Vision – ECCV 14th European Conference*, vol. 9907, Amsterdam, The Netherlands, 2016, pp. 702–716. DOI: 10.1007/978-3-319-46487-9.

[Meh+20] B. Mehta, M. D. Mila, F. G. Mila, C. J. P. Mila, P. Montréal, and C. L. Paull, "Active Domain Randomization", in *Proceedings of the 4th Conference on Robot Learning (CoRL)*, Cambridge, MA, USA, 2020, pp. 1162–1176. DOI: 10.48550/arXiv.1904.04762.

[Mia+23] R. Miao, Q. Jia, F. Sun, G. Chen, H. Huang, and S. Miao, "Semantic Representation of Robot Manipulation with Knowledge Graph", *Entropy*, vol. 25, no. 4, p. 657, 2023. DOI: 10.3390/e25040657.

[MJD18] J. Matas, S. James, and A. J. Davison, "Sim-to-Real Reinforcement Learning for Deformable Object Manipulation", in *Proceedings of the 2nd Conference on Robot Learning (CoRL)*, Munich, Germany, 2018. DOI: 10.48550/arXiv.1806.07851.

[Mni+15] V. Mnih *et al.*, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, pp. 529–533, 7540 2015. DOI: 10.1038/nature14236.

[Mni+16] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning", in *Proceedings of the 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York City, NY, USA, 2016, pp. 1928–1937.

[Moz+20] M. Mozifian, R. Alghonaim, A. Zhang, J. Pineau, and D. Meger, "Intervention Design for Effective Sim2Real Transfer", *Computing Research Repository (CoRR)*, Dec. 2020. DOI: 10.48550/arXiv.2012.02055.

[MRP20] L. R. Martínez, R. A. O. Rios, and M. D. Prieto, *New Trends in the Use of Artificial Intelligence for the Industry 4.0*, R. A. O. Rios and L. R. Martínez, Eds. InTechOpen, 2020, vol. 105980, ISBN: 9781838804664. DOI: http://dx.doi.org/10.5772/intechopen.86015.

[MSK21] M. Motamedi, N. Sakharnykh, and T. Kaldewey, "A Data-Centric Approach for Training Deep Neural Networks with Less Data", in *35th Conference on Neural Information Processing Systems (NeurIPS)*, Online, 2021.

[Mur+18] Z. Murez, S. Kolouri, D. Kriegman, R. Ramamoorthi, and K. Kim, "Image to Image Translation for Domain Adaptation", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 4500–4509. DOI: 10.1109/CVPR.2018.00473.

[Mur+21] F. Muratore, C. Eilers, M. Gienger, and J. Peters, "Data-Efficient Domain Randomization with Bayesian Optimization", *IEEE Robotics and Automation Letters*, vol. 6, pp. 911–918, 2 Apr. 2021. DOI: 10.1109/LRA.2021.3052391.

[MY15] I. Muhammad and Z. Yan, "Supervised Machine Learning Approaches: a survey", *ICTACT Journal on Soft Computing*, vol. 05, pp. 946–952, 03 Apr. 2015. DOI: 10.21917/ijsc.2015.0133.

[Nai+18] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming Exploration in Reinforcement Learning with Demonstrations", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, Sep. 2018, pp. 6292–6299. DOI: 10.1109/ICRA.2018.8463162.

[NR00] A. Y. Ng and S. Russell, "Algorithms for Inverse Reinforcement Learning", in *Proceedings of the 17th International Conference on Machine Learning (ICML)*, Stanford, CA, USA, 2000, pp. 663–670. DOI: 10.5555/645529.657801.

[Ort20] J. H. Ortiz, *Industry 4.0 Current Status and Future Trends*. IntechOpen, 2020, ISBN: 978-1-83880-094-9. DOI: 10.5772/intechopen.86000.

[Pan+22a] F. Pan, T. Zhang, L. Luo, J. He, and S. Liu, "Learn Continuously, Act Discretely: Hybrid Action-Space Reinforcement Learning For Optimal Execution", in *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI)*, Vienna, Austria, 2022, pp. 3912–3918. DOI: 10.24963/ijcai.2022/543.

[Pan+22b] Y. Pang, J. Lin, T. Qin, and Z. Chen, "Image-to-Image Translation: Methods and Applications", *IEEE Transactions on Multimedia*, vol. 24, pp. 3859–3881, 2022. DOI: 10.1109/TMM.2021.3109419.

[Par+19] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review", *Neural Networks*, vol. 113, pp. 54–71, May 2019. DOI: 10.1016/j.neunet.2019.01.012.

[PCD08] C. Pádraig, M. Cord, and S. J. Delany, "Supervised Learning", in P. C. Matthieu and Cunningham, Eds. Springer Berlin Heidelberg, 2008, pp. 21–49, ISBN: 978-3-540-75171-7. DOI: 10.1007/978-3-540-75171-7_2.

[Pen+18a] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-guided deep reinforcement learning of physics-based character skills", *Association for Computing Machinery Transactions on Graphics*, vol. 37, 4 2018. DOI: 10.1145/3197517.3201311.

[Pen+18b] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, Sep. 2018, pp. 3803–3810. DOI: 10.1109/ICRA.2018.8460528.

[PMF16] J. Pagès, L. Marchionni, and F. Ferro, "TIAGo: the modular robot that adapts to different research needs", https://api.semanticscholar.org/CorpusID:218478582, 2016.

[PR17] A. C. Pereira and F. Romero, "A review of the meanings and the implications of the Industry 4.0 concept", *Procedia Manufacturing*, vol. 13, pp. 1206–1214, 2017. DOI: 10.1016/j.promfg.2017.09.032.

[Pra+19] A. Prakash *et al.*, "Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, 2019, pp. 7249–7255, ISBN: 9781538660270. DOI: 10.1109/ICRA.2019.8794443.

[PSM14] J. Pennington, R. Socher, and C. Manning, "GloVe: Global Vectors for Word Representation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[Pte+24] M. Pternea *et al.*, "The RL/LLM Taxonomy Tree: Reviewing Synergies Between Reinforcement Learning and Large Language Models", *Journal of Artificial Intelligence Research*, vol. 80, 2024. DOI: 10.1613/jair.1.15960.

[PZ21] N. Polyzotis and M. Zaharia, "What can Data-Centric AI Learn from Data and ML Engineering?", in *35th Conference on Neural Information Processing Systems (NeurIPS)*, Online, Dec. 2021. DOI: 10.48550/arXiv.2112.06439.

[QSK23] B. Quartey, A. Shah, and G. Konidaris, "Exploiting Contextual Structure to Generate Useful Auxiliary Tasks", in *Workshop on Generalization in Planning on Neural Information Processing Systems (NeurIPS)*, New Orleans, USA, 2023. DOI: 10.48550/arXiv.2303.05038.

[Rao+20] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari, "RL-CycleGAN: Reinforcement Learning Aware Simulation-To-Real", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2020, pp. 11 157–11 166. DOI: 10.1109/CVPR42600.2020.01117.

[Ren+19] X. Ren *et al.*, "Domain Randomization for Active Pose Estimation", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, 2019, pp. 7228–7234. DOI: 10.1109/ICRA.2019.87941.

[RPF19] F. Ramos, R. C. Possas, and D. Fox, "BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators", *Robotics: Science and Systems (RSS)*, Jun. 2019. DOI: 10.48550/arXiv.1906.01728.

[RSF13] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework", in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013, pp. 1321–1326. DOI: 10.1109/IROS.2013.6696520.

[Rus+16a] A. A. Rusu *et al.*, "Policy Distillation", in *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, Nov. 2016. DOI: 10.48550/arXiv.1511.06295.

[Rus+16b] A. A. Rusu *et al.*, "Progressive Neural Networks", *Computing Research Repository (CoRR)*, Jun. 2016. DOI: 10.48550/arXiv.1606.04671.

[Rus+17] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-Real Robot Learning from Pixels with Progressive Nets", in *Proceedings of the 1st Conference on Robot Learning (CoRL)*, Mountain View, CA, United States, 2017. DOI: 10.48550/arXiv.1610.04286.

[RYG22] M. Reid, Y. Yamada, and S. S. Gu, "Can Wikipedia help offline reinforcement learning?", *Computing Research Repository (CoRR)*, 2022. DOI: 10.48550/arXiv.2201.12122.

[SA20] F. Shoeleh and M. Asadpour, "Skill based transfer learning with domain adaptation for continuous reinforcement learning domains", *Applied Intelligence*, vol. 50, 2 Feb. 2020. DOI: 10.1007/s10489-019-01527-z.

[SAV20] E. Stevens, L. Antiga, and T. Viehmann, *Deep Learning with PyTorch*. 2020, ISBN: 9781617295263.

[SB18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 2018, ISBN: 9780262039246.

[Sch+15a] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay", in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015. DOI: 10.48550/arXiv.1511.05952.

[Sch+15b] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization", in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, vol. 37, Lille, France, 2015, pp. 1889–1897. DOI: 10.5555/3045118.3045319.

[Sch+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms", *Computing Research Repository (CoRR)*, 2017. DOI: 10.48550/arXiv.1707.06347.

[Sch16] K. Schwab, *The Fourth Industrial Revolution*. 2016, ISBN: 9781944835019.

[Shi+23] Y. Shi *et al.*, "A Sim-to-Real Learning-Based Framework for Contact-Rich Assembly by Utilizing CycleGAN and Force Control", *IEEE Transactions on Cognitive and Developmental Systems*, vol. 15, pp. 2144–2155, 4 2023. DOI: 10.1109/TCDS.2023.3237734.

[Sil+14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms", in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, vol. 32, Beijing, China, 2014, I–387–I–395. DOI: 10.5555/3044805.3044850.

[TA20] Y. Tang and S. Agrawal, "Discretizing Continuous Action Space for On-Policy Optimization", in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, vol. 34, New York, NY, USA, Apr. 2020, pp. 5981–5988. DOI: 10.1609/aaai.v34i04.6059.

[TAK23] G. Tiboni, K. Arndt, and V. Kyrki, "DROPO: Sim-to-real transfer with offline domain randomization", *Robotics and Autonomous Systems*, vol. 166, Aug. 2023. DOI: 10.1016/j.robot.2023.104432.

[TET12] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control", in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[Tob+17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World", in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, Canada, Mar. 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.

[Tow+23] M. Towers *et al.*, *Gymnasium*, 2023. DOI: 10.5281/zenodo.8127026.

[TPL20] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2020, pp. 10 778–10 787. DOI: 10.1109/CVPR42600.2020.01079.

[Tra+19] R. Traoré, H. Caselles-Dupré, T. Lesort, T. Sun, N. Díaz-Rodríguez, and D. Filliat, "Continual Reinforcement Learning deployed in Real-life using Policy Distillation and Sim2Real Transfer", in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, Long Beach, CA, USA, Jun. 2019.

[Uni20] Universal Robots, "Univeral Robots e-Series UR3e User Manual", Tech. Rep., version 5.7, 2020.

[vHGS15] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning", in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, Austin, TX, USA, 2015. DOI: 10.1609/aaai.v30i1.10295.

[Wan+16a] J. X. Wang *et al.*, "Learning to reinforcement learn", *Computing Research Repository (CoRR)*, Nov. 2016. DOI: 10.48550/arXiv.1611.05763.

[Wan+16b] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning", in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, New York City, NY, USA, 2016, pp. 1995–2003. DOI: 10.5555/3045390.3045601.

[WD18] M. Wang and W. Deng, "Deep Visual Domain Adaptation: A Survey", *Neurocomputing*, vol. 312, pp. 135–153, Feb. 2018. DOI: 10.1016/j.neucom.2018.05.083.

[Wil92] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning", *Machine Learning*, vol. 8, no. 3–4, pp. 229–256, 1992. DOI: 10.1007/BF00992696.

[Xin+21] J. Xing, T. Nagata, K. Chen, X. Zou, E. Neftci, and J. L. Krichmar, "Domain Adaptation In Reinforcement Learning Via Latent Unified State Representation", in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, Online, 2021, pp. 10 452–10 459. DOI: 10.1609/aaai.v35i12.17251.

[Xu+21] X. Xu, Y. Lu, B. Vogel-Heuser, and L. Wang, "Industry 4.0 and Industry 5.0—Inception, conception and perception", *Journal of Manufacturing Systems*, vol. 61, pp. 530–535, Oct. 2021. DOI: 10.1016/j.jmsy.2021.10.006.

[Yua+22] C. Yuan *et al.*, "Sim-to-Real Transfer of Robotic Assembly with Visual Inputs Using CycleGAN and Force Control", in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Xishuangbanna, China, 2022, pp. 1426–1432. DOI: 10.1109/ROBIO55434.2022.10011878.

[Yue+19] X. Yue, Y. Zhang, S. Zhao, A. Sangiovanni-Vincentelli, K. Keutzer, and B. Gong, "Domain Randomization and Pyramid Consistency: Simulation-to-Real Generalization without Accessing Target Domain Data", in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, South Korea, 2019, pp. 2100–2110. DOI: 10.1109/ICCV.2019.00219.

[Zha+19] J. Zhang *et al.*, "VR-Goggles for Robots: Real-to-Sim Domain Adaptation for Visual Control", *IEEE Robotics and Automation Letters*, vol. 4, pp. 1148–1155, 2 Apr. 2019. DOI: 10.1109/LRA.2019.2894216.

[Zhu+17] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks", in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, vol. 2017-October, Venice, Italy, Dec. 2017, pp. 2242–2251. DOI: 10.1109/ICCV.2017.244.

[Zhu+18] Y. Zhu *et al.*, "Reinforcement and Imitation Learning for Diverse Visuomotor Skills", in *Proceedings of Robotics: Science and Systems*, Pittsburgh, PA, USA, Feb. 2018. DOI: 10.15607/RSS.2018.XIV.009.

[ZLZ20] Z. Zhu, K. Lin, and J. Zhou, "Transfer Learning in Deep Reinforcement Learning: A Survey", *Computing Research Repository (CoRR)*, Sep. 2020. DOI: 10.48550/arXiv.2009.07888.