Université de Montréal

On quantifying the value of simulation for training and evaluating robotic agents

par

Anthony Courchesne

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Intelligence Artificielle et Robotique

April 2021

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

On quantifying the value of simulation for training and evaluating robotic agents

présenté par

Anthony Courchesne

a été évalué par un jury composé des personnes suivantes :

Philippe Langlais
(président-rapporteur)

Liam Paull
(directeur de recherche)

Guillaume Rabusseau
(membre du jury)

Résumé

Un problème récurrent dans le domaine de la robotique est la difficulté à reproduire les résultats et valider les affirmations faites par les scientifiques. Les expériences conduites en laboratoire donnent fréquemment des résultats propres à l'environnement dans lequel elles ont été effectuées, rendant la tâche de les reproduire et de les valider ardues et coûteuses. Pour cette raison, il est difficile de comparer la performance et la robustesse de différents contrôleurs robotiques. Les environnements substituts à faibles coûts sont populaires, mais introduisent une réduction de performance lorsque l'environnement cible est enfin utilisé. Ce mémoire présente nos travaux sur l'amélioration des références et de la comparaison d'algorithmes ("Benchmarking") en robotique, notamment dans le domaine de la conduite autonome.

Nous présentons une nouvelle platforme, les Autolabs Duckietown, qui permet aux chercheurs d'évaluer des algorithmes de conduite autonome sur des tâches, du matériel et un environnement standardisé à faible coût. La plateforme offre également un environnement virtuel afin d'avoir facilement accès à une quantité illimitée de données annotées. Nous utilisons la plateforme pour analyser les différences entre la simulation et la réalité en ce qui concerne la prédictivité de la simulation ainsi que la qualité des images générées. Nous fournissons deux métriques pour quantifier l'utilité d'une simulation et nous démontrons de quelles façons elles peuvent être utilisées afin d'optimiser un environnement proxy.

Mots Clés

Conduite Autonome, Robotique, Références et Comparaisons, Apprentissage Automatique, Sim-to-Real, Simulation, Science Reproductible, Apprentissage par Imitation, Reality Gap

Abstract

A common problem in robotics is reproducing results and claims made by researchers. The experiments done in robotics laboratories typically yield results that are specific to a complex setup and difficult or costly to reproduce and validate in other contexts. For this reason, it is arduous to compare the performance and robustness of various robotic controllers. Low-cost reproductions of physical environments are popular but induce a performance reduction when transferred to the target domain. This thesis present the results of our work toward improving benchmarking in robotics, specifically for autonomous driving.

We build a new platform, the Duckietown Autolabs, which allow researchers to evaluate autonomous driving algorithms in a standardized framework on low-cost hardware. The platform offers a simulated environment for easy access to annotated data and parallel evaluation of driving solutions in customizable environments. We use the platform to analyze the discrepancy between simulation and reality in the case of predictivity and quality of data generated. We supply two metrics to quantify the usefulness of a simulation and demonstrate how they can be used to optimize the value of a proxy environment.

Keywords

Autonomous Driving, Robotics, Benchmarking, Machine Learning, Sim-to-Real, Simulation, Reproducible Science, Imitation Learning, Reality Gap

Contents

List of Tables

List of Figures

List of Acronyms and Abbreviations

AIDO	AI Driving Olympics
BC	Behavior Cloning
DTA	Duckietown Autolab
DUCKIENet	Decentralized Urban Collaborative Benchmarking Network
CNN	Convolutional Neural Network
CTE	Cross-Track Error
EKF	Extended Kalman Filter
KF	Kalman Filter
IID	Independant and Identically Distributed
IL	Imitation Learning
IMU	Inertial Measurement Unit

LF	Lane-following
LIDAR	LIght Detection And Ranging (Sensor unit)
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-Layer Perceptron
MOD	Mean Orientation Deviation
MPD	Mean Position Deviation
PID	Proportional-Integral-Derivative
PLV	Proxy Learning Value
POD	Proxy Observation Discrepancy
POMDP	Partially-Observable Markov Decision Process
PPV	Proxy Predictivity Value
PRPV	Proxy Relative Predictivity Value

RL	Reinforcement Learning
SRCC	Sim-vs-Real Correlation Coefficient

Acknowledgements

First, I would like to thank my supervisor Liam for believing in me and generously sharing his immense knowledge. He has been raising the bar constantly and helped me understand how to properly conduct scientific experiments. I am grateful for everything he did for me, as well as the overall Robotics and Embodied AI Lab during the pandemic. Even if they did not last long, Wednesday hacking sessions were a great idea that I was always looking forward to.

I thank the Faculty of Art and Science (FAS) and Département d'Informatique et de Recherche Opérationelle de l'Université de Montréal (DIRO) for financially supporting my research during the pandemic.

I would like to thank my parents for sharing the emotions that I have been going through. Moreover, I am grateful for the expertise and advice that my mother has constantly provided me, and my father, who shared his tools and personal workshop to help me get through my studies. I am extremely grateful to my brother, who has been kind enough to host and support me during this peculiar year. I cannot imagine how any of this would have been possible without his generosity.

I would like to acknowledge the help that I received from people at The Duckietown Foundation, Mila, and the Montreal Robotics group, without which this work would not have been possible.

It is important to mention how helpful it was to have frequent online discussions with my friend Louis-Alexandre in times were physical distancing usually ends up in social distancing.

Last but not least, I want to thank my girlfriend Marie-Christine for supporting, loving, and understanding me from the beginning to the end of this adventure.

Introduction

Problem Definition

With the advance of deep learning techniques to control robots, simulators have seen an increase in popularity as agents trained entirely in a simulated environment were successful in performing difficult tasks in the real world [**1**], [**52**]. Multiple scientific works has since made use of simulators, raising concerns about the value of a simulated environment. It was said that "Simulators are doomed to succeed", referring to the fact that simulators can often be adjusted to prove a hypothesis without much scientific value, in which case an agent overfits a simulated environment and fails to have any value in a real-world environment. The discrepancy between the real and simulated environment is called the "Reality Gap", but it has been used without metric or clear definition. It is relevant to re-think the purpose of the simulator and evaluate its usefulness in different setups. The lack of metrics to evaluate a simulation to reality gap has been hindering the progress of science since papers have been claiming different ways of crossing the gap without a precise way to benchmark and compare to other state-of-the-art methods. The main contributions of this thesis are trying to address this problem.

The Usefulness of Benchmarks

Unified benchmarks are crucial to be able to reproduce and compare research. Computer vision has been able to make significant progress following the arrival of datasets such as KITTI [**33**], where claims of performance were being validated on a standard dataset, confirming the value of new approaches. In robotics, when a claim is made, it is not easy to reproduce the same environment. Robots are expensive and minor discrepancies in the reproduced environment might yield major performance change variations. This is especially true in sim-to-real transfer when there are multiple environments. One way to compare efficiently different algorithms is through robotic challenges (duckietown [**64**], darpa [**14**], iGibson [**97**]) where all techniques are evaluated on standardized tasks, environments and hardware.

Autonomous Driving

The specific robotic task that we are interested in for our work is autonomous driving. This consists of various tasks, the easiest being Lane-Following (LF), where an agent has to drive on the right side of the yellow dashed lane for as long as possible. There are multiple scenarios but each usually includes straight paths and 90° left and right turns. The goal is to reduce the cross-track error (CTE) by driving as close as possible to the center of the lane. The agent also needs to maximize its survival time and the distance it has moved in the right direction, and it gets penalized if it goes completely out of the lane. Depending on a task specification, these scoring components can be used or weighted differently. Additionally, autonomous driving often includes related tasks such as driving through intersections, avoiding pedestrians and obstacles, and properly adapting driving when sharing the road with other moving vehicles.

Duckietown, a Proxy to Autonomous Driving

In our work, we make extensive use of the Duckietown platform, where duckiebots (fig. 0.1) are used as proxies for cars. The tasks mentioned previously are available as the Lane-Following, Lane-Following-Vehicles, Lane-Following-Pedestrians, and Lane-Following-Intersections challenges. The environment is a *duckietown*, the roads are modulable foam tiles and civilians (including drivers and pedestrians) are rubber duckies. The same tasks and environment are available in a simulation.

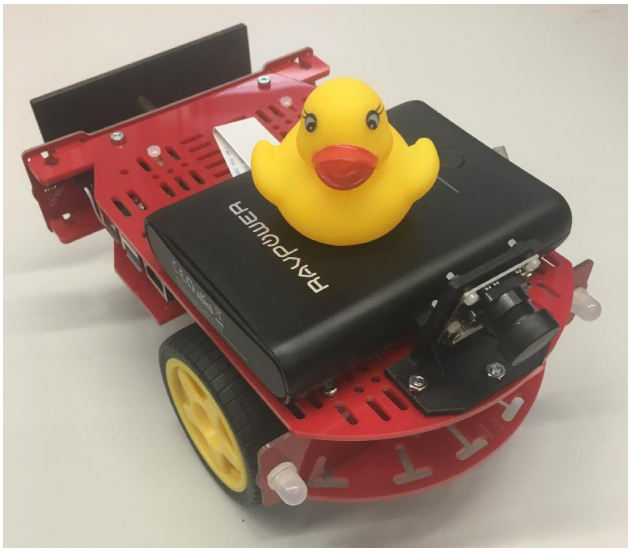


Fig. 0.1. The Duckiebot (version DB18) is a low-cost vehicle used as a proxy for a car. Its raspberry feeds the images from the camera to a *Docker* container that encapsulates the control software, receives the commands, and transmits them to the wheels' actuators.

Classical vs ML-Based Controllers

Historically, multiple solutions have been found to the autonomous driving problem using mathematical models with various degrees of success. Those techniques are called *classical methods* and include controllers such as bang-bang (sec. 1.2.4.1), PID (sec. 1.2.4.2) and KF/EKF (sec. 1.2.2.1). More recently, neural networks have been used for different components of the driving system such as object detection and semantic segmentation, although success has been demonstrated for full-fledged controllers based on machine learning, either Reinforcement Learning or Imitation Learning. Having access to challenges for benchmarking, it is relevant to analyze the performance, robustness, cost, and limitations of classical vs ML-based approaches. A primer on robotics notation (sec 1.1), basic ML techniques (sec 1.3) and ML-based controllers (sec. 1.4) is provided in the background section to help the reader better understand the techniques that are used to make such comparison in our work.

The need for Simulation

In order for roboticists to test their software and prevent hardware damage, a simulator is typically used. This allows quick deployment and execution of a controller in various environments, but also collection of big amount of annotated data, a crucial part of machine learning. The main challenge of simulated environments is the reality gap, induced by the discrepancy of the source and target domains. More context about simulators and the reality gap can be found in the background section (sec. 1.5.5). Our work will be directed towards the analysis of this gap, and how it can be quantified and minimized through simulator analysis and optimization.

Contributions

- **Proxy Relative Predictive Value:** One specific use of simulators is being a proxy for the real world as an evaluation environment. Simulators are often used this way to choose which agents are more promising before investing resources to perform real-world trials. This is frequently done by ranking agent performance for a given task. We claim that for that purpose, the value of a simulator as a predictor is better if the simulator can predict the relative ordering of agent performance conditioned on a task. We propose a new metric to assess the usefulness of a simulator as a predictor, the *Proxy Relative Predictive Value*.
- **Proxy Learning Value:** We also claim that the simulators are useful to save time and resources for the training of agents. We establish a new metric, the *Proxy Learning Value*, to assess the usefulness of a simulator as a teacher.

- **Simulator Optimization Instance in Duckietown:** We provide an analysis of the new metrics in the framework of Duckietown [**64**], a low-cost framework for research on autonomous vehicles. We set up and solve optimization problems for the simulator using both *PRPV* and *PLV*, providing an analysis of the optimal simulator obtained in both scenarios and how to reproduce for different frameworks.
- **Duckietown Autolabs:** Part of the work was concluded to help configure an environment for automatic evaluation of autonomous driving agents (chap. 5). The result is a distributed network of autolabs where docker containers encapsulating software solutions are evaluated seamlessly in a simulation and then in an embodied environment, enabling reproducible research with robots in the loop.

Thesis Layout

The first chapter of the thesis will contain preliminary information useful to understand our contributions, including simulation, sim-to-real and deep learning. Following will be the core of the work that was done as part of my master, the paper that was submitted at IROS2021 titled "On Assessing the Usefulness of Proxy Domains for Developing and Evaluating Embodied Agents". It will be preceded by a short introductory chapter that contains context and article details. In a similar fashion, chapter 4 will be a prologue to chapter 5, which contains the paper that I contributed to titled "Integrated Benchmarking and Design for Reproducible and Accessible Evaluation of Robotic Agents" which was submitted and accepted at IROS2020. While I was not a major contributor to the text in the second paper, it encloses the architecture work that I have been doing on the Duckietown platform in the first part of my Master's.

Chapter 1

Background

Before diving into our contributions, we briefly review some preliminary material required to grasp the value of the claims we make in our work. It will be useful for the reader to understand Markov Decision Processes and how they are used to model robot-environment interactions. We include a small section about classical approaches to solving autonomous driving, and then more modern, machine-learning-based solutions such as reinforcement learning, behavior cloning, and imitation learning. It will be useful to the reader to have a basic understanding of CNNs (sec. 1.3.2) and end-to-end autonomous driving methods (sec. 1.4.1) since these will be used in our work to assess the usefulness of a proxy when using such techniques. The last section of this chapter is about simulators and their uses and flaws. The reader should be familiar with those to understand the contribution of the research that we present subsequently.

1.1. Modeling in Robotics

In order to build a controller for a robot, it is required to encode the information that the robot is given. That includes what the robot sees (observations z), what the robot knows (set of past observations) and how the robot acts (controls u). It is common to formalize the problem of robotics controls as a *Markov Decision Process* (MDP), enabling precise mathematical formulation.

1.1.1. Markov Decision Process

An MDP is used to model a stochastic dynamical system in a sequential decision-making problem. When in a state s_t at time t, an agent submits an action $a_t \in \mathcal{A}(s)$ to the environment. In return, the environment provides the agent with a reward $r_{t+1} \in \mathcal{R}$ and a new state s_{t+1} as shown in figure 1.1.

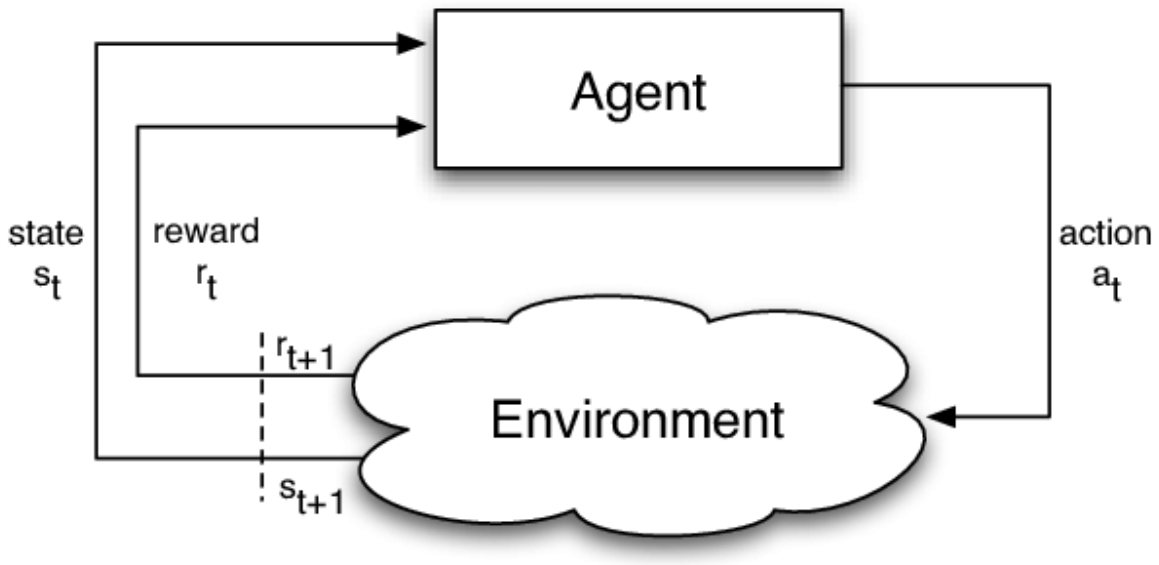


Fig. 1.1. Agent-environment interactions as modeled in an MDP. *Credits: de Lope, Javier, and Darío Maravall. "The knn-td reinforcement learning algorithm." International Work-Conference on the Interplay Between Natural and Artificial Computation. Springer, Berlin, Heidelberg, 2009.*

An MDP is a stochastic decision process, so the transitions between states are probabilistic. Given a state s and action a, we note the probability of ending in a state s' as

$$p(s'|s,a) \triangleq Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} \tag{1.1.1}$$

As seen in eq. 1.1.1, the probability of ending in a state s' at time $T = t$ depends on the previous state s_{t-1}, but not directly on any state before that. Since MDPs are an extension of *Markov Chains*, they also satisfy the *Markov Property*. The latter specifies that a state is entirely characterized by the previous state and action, and it has no memory about states or actions before.

In robotics, the MDPs often consist of a 5-tuple: (S,A,T,R,γ).

- $\boldsymbol{S}$: Set of possible states of the system. A state s_t is assumed to capture the information of previous transitions and states and as such, is independent of an anterior state s_{t-1}
- $\boldsymbol{A}$: Possible actions that the agent can take. At a given state $s \in S$, only a subset of action $A(s)$ are available to the agent to choose from
- $\boldsymbol{T}$: Transition probabilities. The transitions are given in the form of triplets. Given a state $s \in S$, action $a \in A(s)$ and a second state $s' \in S$, there is a probability p that the agent in state s taking action a ends up in state s'
- $\boldsymbol{R}$: Reward that the environment gives to the agent. In a state s, if the agent carries out action a, it will be rewarded with $r \in R$

- $\boldsymbol{\gamma}$: Discount factor. Using the discount factor ($\gamma \in [0,1]$), it is possible to specify the trade-off between an immediate reward and a long-term reward. A value of 1 means that the reward is the same, whether we receive it now or in future actions. A value close to zero means that it is important to obtain the reward as soon a possible, and planning to obtain a reward with subsequent actions will yield a discounted, lower reward.

1.1.2. Partially Observable Markov Decision Process

In robotics, it is common that the agent is not able to capture all the information about the state of the environment. In that case, we use a *Partially Observable Markov Decision Process* (POMDP). In addition to the 5 components that define a MDP, we append two more for POMDPs.

- $\boldsymbol{\Omega}$: Observations that can be made by the agents. In a robotic framework, it is possibly a grid of RGB-D pixels, a range of LIDAR scans, angular acceleration from an IMU, etc.
- $\boldsymbol{O}$: Conditional observations. Given an action a performed by the agent and an end-state s', the agent will make observation $o \in \Omega$ with probability $p(o|s',a)$. These observations are usually noisy approximations of parts of the real state of the environment.

1.2. Classical Methods for Autonomous Driving

Autonomous robotic agents are usually made of multiple components, as shown in fig. 1.2. The actuation, environment model, and sensing are handled by the environment or the simulator, and interactions with the robot are captured through control commands and sensor data exchanges.

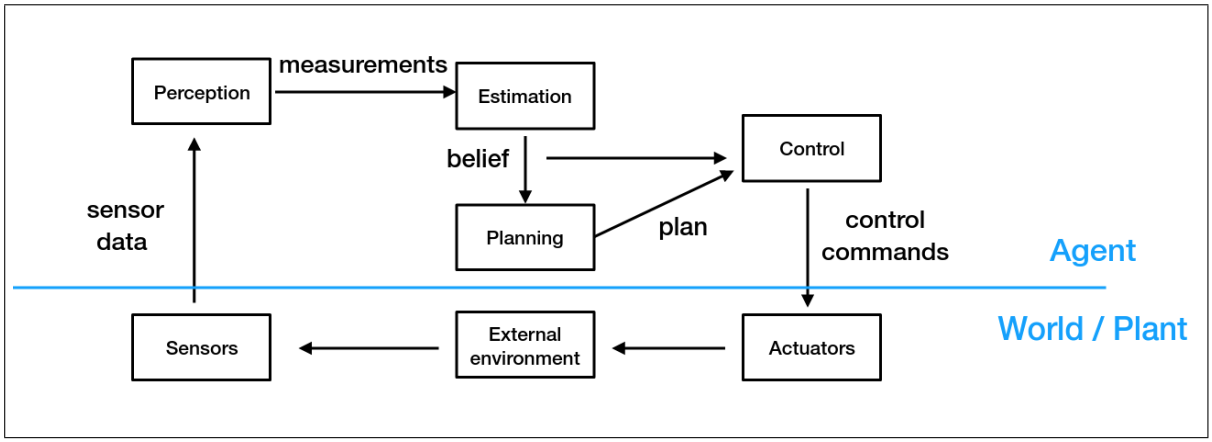


Fig. 1.2. Traditional components of a robotic system. *Credits: Liam Paul, "IFT6757 - Autonomous Vehicles (Duckietown)"*

1.2.1. Perception

The perception module is given sensor data and needs to analyze the data to find meaningful information. This is often in the form of lines, SIFT [**53**] or SURF [**5**] features, semantic maps, detected objects, etc. For our work, we will not make extensive use of the perception component since we will use the default line detection module provided with Duckietown, and in the case of our ML-based agent, we will not use any perception module.

1.2.2. Estimation

Given the measurements, we now want to estimate the pose of the robot in the 3D world. This is done by taking into account what we already know and updating the probability that the robot is in a given state with the new measurements. The calculation required to obtain the pose belief involves integrating the probability that we are in a certain state, given the measurements and previous state. Most of the time, this integral has no closed-form [**87**], so we require an alternate way to find the pose of the robot. One solution is to discretize the space to turn the integration into a sum. Another solution is to approximate the beliefs by normal distributions. The latter solution is used in the Kalman Filter, perhaps the most popular state-estimation algorithm.

1.2.2.1. Kalman Filter. A Kalman Filter is an online algorithm that takes noisy measurements and update the belief of the robot pose. The algorithm is divided in two parts. In the first step, the algorithm predicts what is the current state and its uncertainty using the knowledge it has from previous states and observations. During the second phase, it updates its belief using the new measurements.

The Kalman Filter assumes that the underlying dynamical system is linear. For the prediction and update equations of the Kalman Filter as well as the Extended Kalman Filter which supports non-linear dynamical systems, we refer the reader to more extensive work on the subject [**93**].

1.2.3. Planning

In classical robotic tasks, the agent is required to plan its path to navigate the environment and reach its goal. This can be done with various techniques, but one popular easily accessible version is A*, where the shortest path to reach a goal is found using heuristics. In our case, we don't need any planning system since the route is imposed in the task of lane-following.

1.2.4. Control

Once the robots has a good estimation of its pose and where it wants to go, it needs to figure control commands to send to the actuators such that it will end in its target state. For our example of lane-following, the commands would be actuators torques, or at a higher level, steering angles and forward velocity (throttle).

A naive controller to reach the target state would be the "bang-bang" controller (sec. 1.2.4.1). Elaborating from this solution, multiple improvements have been made and different techniques exist to build classical controllers. Before diving into algorithms powered by neural networks, it is relevant to have a brief understanding of the most popular classical techniques to drive a vehicle along a lane.

1.2.4.1. Bang-Bang controllers. Also known as "2-steps" or "On-Off" controllers, these controllers are perhaps the simplest. They simply alternate between two controls to obtain the desired state. In a lane following task, an example of a bang-bang controller would be an agent that turns right if it is at the left of the lane and left if it is at the right of the lane. Unfortunately, these types of solutions do not achieve good performance as they usually overshoot the target state (middle of the lane) and end up oscillating around it.

1.2.4.2. PID Controller. One way to improve upon the previous controller is to plan ahead how much the agent is going to overshoot the target state. A PID controller uses the difference between the target state and the current state (the error). The control vector that is returned from a PID is the sum of a proportional term, a derivative term, and an integral term. For each term, a gain parameter needs to be tuned.

- ***P***: The proportional term tells the robot to go left or right with a value proportional to how far it is to the middle of the lane. If the robot is far from the lane, it should take a sharper turn to reach the target state faster. Increasing this term would yield increased amplitude in the reaction to errors (fig. 1.3)
- ***I***: The integral term tries to drive the long-term error to zero. This is especially useful when there are systematic errors, such as a miscalibrated actuator, or wear on a wheel resulting in different friction.
- ***D***: The derivative term anticipates how quickly the agent is approaching the goal state, and attenuate the control to prevent overshoot and oscillation.

Thus, the controls u as a function of time t given by a PID controller are

$$u(t) = K_p\, e(t) + K_i \int_0^t e(t)\, dt + K_d \frac{d\, e(t)}{dt} \tag{1.2.1}$$

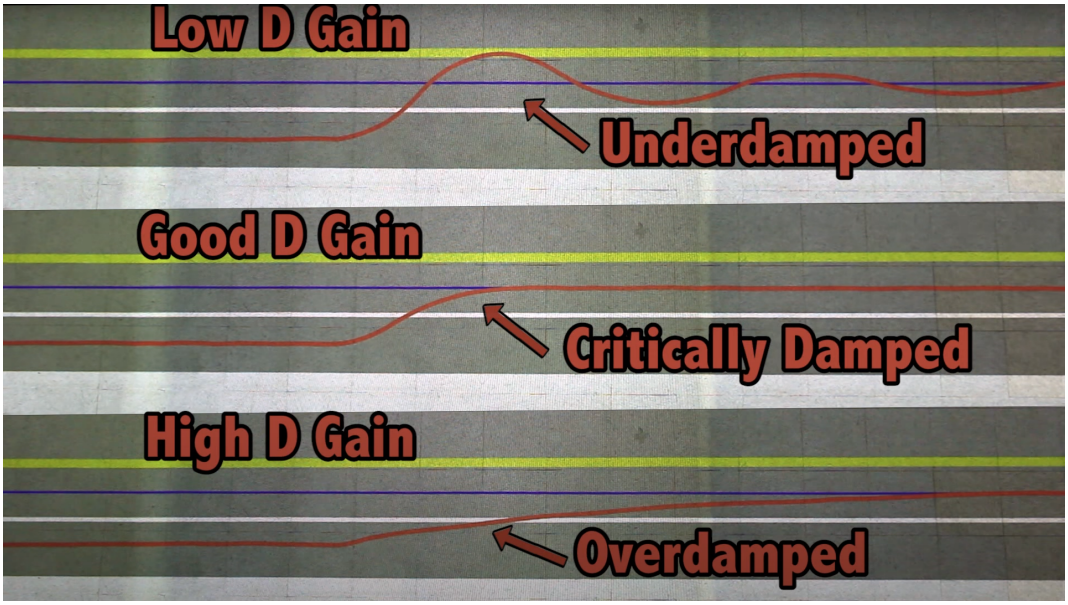


Fig. 1.3. Effects of varying the P-gain of a PID controller on an autonomous car. *Credits: MIT Aerospace Controls Lab: "Controlling Self Driving Cars", 2015*

1.3. Machine Learning and Deep Learning

It has been observed that artificial intelligence (AI) is a powerful tool, able to solve complex tasks such as chess [**40**]. However, when problem description becomes less trivial, it becomes impossible for a programmer to inform the machine of all the rules and possibilities. The game of Go, in contrast with chess, is much harder to optimize since it has an untractable number of possible states. Fortunately, neural networks have been succesfully solving such problems [**76**].

Deep learning is the solution to let a machine acquire its own knowledge, as opposed to leaving the job to the programmer to input all the rules and possibilities. The name *Deep* originates from the fact that the knowledge is stored as a hierarchy of concepts, from simple to complex. A graph representing the dependency between these concepts would be very *deep* [**35**]. The term *learning* is derived from the data that is fed to the network, effectively *learning* the concepts by itself.

1.3.1. Multi-layer Perceptron

The simplest (and most frequent) neural network is a Multi-Layer Perceptron (MLP) (fig. 1.4), also known as a feed-forward neural network. The input nodes carry information that is fed to the network. The arrows contain weights w_i, indicating how important that node is for the concept linked to the right-hand side of that arrow. The nodes in the middle are named *hidden layers* since they abstract away complex concepts that the programmer does not need to know. Often, these nodes represent data in a non-intuitive way and are

hard for a human to understand. It is frequent to have neural networks with many hidden layers, each of which becomes more complex and abstract. Finally, the rightmost nodes are the output nodes, which hold the information that the network inferred from the inputs. That might be one or multiple scalars, or a class such as *dog* or *cat*.

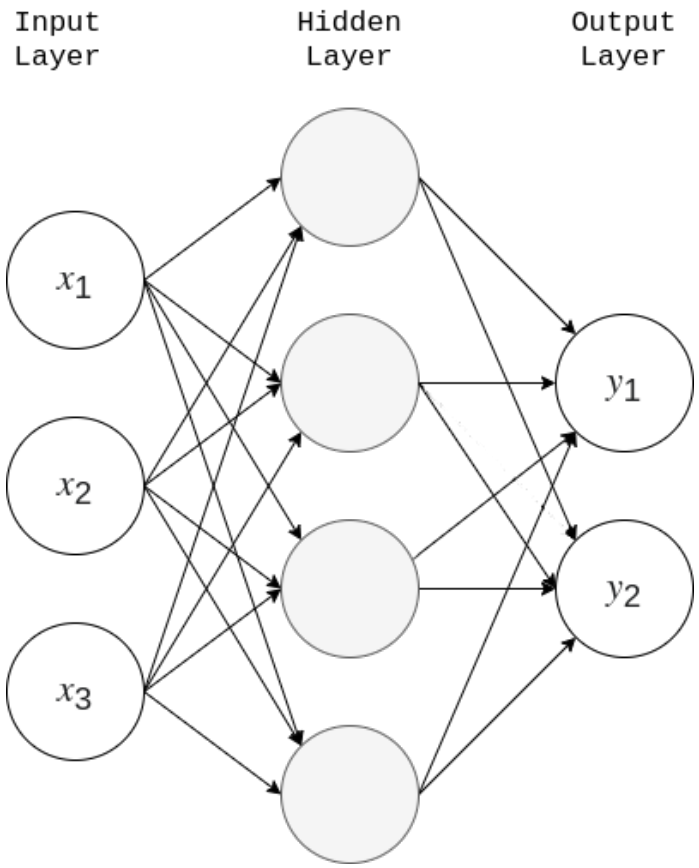


Fig. 1.4. Graph of a simple MLP with 3 input nodes, one hidden layer of 4 nodes and two output nodes.

In the simpler case, a neuron sums its weighted inputs, applies a non-linear activation function, and outputs the value. The use of an activation function allows the network to have non-linear hidden representation. The output of a node with k inputs is then given by

$$y = \varphi(\sum_{k=1}^{n} x_k \, w_k)$$

where φ is the activation function. To permit backpropagation of the error, the activation needs to be differentiable. Historically, the logistic function has been widely used but nowadays it is frequent to see alternatives such as ReLU, which is a piecewise linear function that keeps the property of linear function making it easy to optimize while also performing a non-linear transformation.

The loss function $\mathcal{L}$ is used to quantify how far away was the network's prediction from the true value. When we train the MLP, we are optimizing the weights to reduce the expected loss on new data samples. It is common to use the mean squared error for scalar outputs. In that case, we are trying to minimize the error according to:

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} ||\hat{y}_i - y_i||^2$$

over the whole dataset of m samples, where y is the network's output and $\hat{y}$ is the target value.

During the training of a network, the MLP is given tuples of input-output for which it will optimize the weights w_i for each link of the network. Once the network is given an example, it compares its predicted output to the target output according to the loss function and it performs *backpropagation* to propagate the error through the nodes.

Using the chain rule, the derivatives of the error with respect to the weight w_i are computed at each node and the parameters are updated using gradient descent. Depending on the learning rate (how quickly the model parameters are updated), the network is expected to converge to optimal parameter values to generate output with minimal loss.

1.3.2. Convolutional Neural-Network

Convolutional Neural Networks (CNNs) are a specific type of neural network that are used when the input data is in the form of an array. This is not restricted to, but especially useful for image inputs. Any neural network that has a *convolution* operation instead of one of the matrix multiplication is called a CNN.

A convolution is an operation on two functions x and w. The output is a third function s that describes how x affects w. For the purpose of neural networks, the first function (x) refers to the input while the second function (w) refers to the kernel (also known as the *filter*).

The weight matrix of a CNN is usually sparse, which leads to fewer multiplications required to train the model. This is due to the fact that the kernel is usually small, so an input node only affects a few neighboring nodes in the next layer (sparse connectivity). Additionally, the kernels are reused for multiple regions of the input, so the weights are shared.

1.3.2.1. Convolutional Layer. When a CNN is given a grid of pixels, the convolution operation happens by taking a kernel and sliding it across the image. The output of a convolutional layer is another array of the same dimension[1], for which every entry is the result of applying the kernel onto the output of that layer at the same index and neighboring region.

[1] assuming appropriate padding

It is possible to specify the size of the kernel, the number of pixels for the translation of the kernel at each iteration (also known as the stride), and whether or not to add padding to the image such that the kernel can reach the borders and corners. Those options are called *hyper-parameters* since they are specified to the network and not learned. It is common to tune these on a validation set using *grid-search* or other techniques.

The kernel is learned by the network as part of the training procedure.

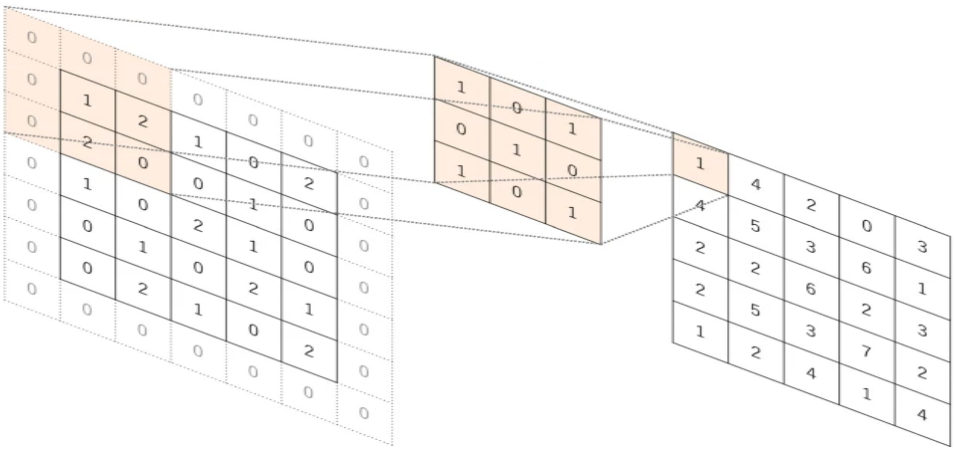


Fig. 1.5. The convolution of a kernel over a padded input matrix. *Credits: Yamashita, Rikiya, et al. "Convolutional neural networks: an overview and application in radiology." Insights into imaging 9.4 (2018): 611-629.*

1.3.2.2. Activation Layer. Similar to how it is done in a normal neural network, the activation function maps its input to a non-linear domain. A frequent activation function for CNNs is a rectified linear unit (ReLU). See the section about MLP (sec. 1.3.1) for more information.

1.3.2.3. Pooling Layer. The pooling layer summarizes what was seen in a region. That has the benefit of reducing the dimension of the information carried over to the next layer. Often, this results in a lot of resources saved at training time. By doing pooling between each convolution, a tractable set of dimensions can be reached once the data gets to the fully connected layer, for which the matrix multiplication is expansive.

Max-pooling is one of the most popular techniques, where every neighborhood of pixels (in the case of an image) will be represented by the pixel with the highest value. Averaging is another popular pooling operation, such as L^2-norm and distance-weighted average. These operations remove some of the noise in the input and also allow a small tolerance to translation in the input image for which they would give similar output. Both of these features might be wanted or not, in which case it is possible to omit the pooling layer completely and use a smaller image size to compensate for the fixed dimensions.

Using pooling layers, it is also possible to feed images of varying sizes to the network. Instead of pooling a specific amount of input pixels, a pooling layer can be configured to output summary for a fixed number of regions. This allows images of different sizes to be fed to the network while keeping the input to the classification layer the same.

1.3.2.4. Fully Connected Layer. The fully connected layer is usually at the end of the network architecture. It is a simple MLP, so the input is typically flattened before being fed to the fully connected layer. Since this layer is *fully connected* (i.e. all neurons are connected with all the inputs, unless otherwise specified), the number of weights is the number of input to that layer multiplied by the number of neurons in the fully connected layer. For this reason, it is useful to reduce the dimensionality of the data carried through the network in the previous layer. Otherwise, the number of parameters risks being unreasonably large, therefore requiring more data samples to train the network.

1.3.2.5. Softmax Layer. The final layer of a CNN is commonly the *Softmax Layer*, which applies the softmax activation function. This allows the network to outputs its prediction as a probability of each available category for classification (although this should not be used as a measure of confidence !).

1.4. Machine Learning Applications for Autonomous Driving

1.4.1. Behavior Cloning

End-to-end learning has been successfully used to teach cars to drive themselves [**9**]. Behavior Cloning is one such instance where the problem of autonomous driving is formulated as a supervised learning problem. It is required to have access to an *expert*, able to generate annotated data, also called a "teacher". The behavioral cloning algorithm will then attempt to learn a function that maps the input (commonly images from what is in front of the car) to the controls that the expert chooses. This is typically done using a CNN. However, the data is always assumed to be I.I.D, which is not always the case. More specifically, data samples collected from driving a car are usually correlated to other samples with nearby timestamps.

In order to be able to train a useful controller, it is required that the dataset generated by the expert has enough variability. The agent should be shown how to recover from difficult situations, like when it has made an error and ended up outside of the lane. Since at test time, the agent that controls the vehicle is different from the controller that was used at training time, the agent is bound to make some errors (also known as *covariate shift*). Being in a state that has not been seen in the training data, the error made by the controller at

test time will cascade [**3**]. One specific solution to help the controller recover from unseen states is dataset augmentation [**9**], where data collected from the expert is augmented using translation and rotation to increase the variability.

1.4.2. On-Policy vs Off-Policy Algorithms

A different way to solve the covariate shift is to query the expert using samples that the agent generated while executing its own learned policy. Dataset aggregation (DAgger) [**70**] is one specific method that aggregates from the agent's policy to the dataset before readjusting the neural network weights to fit the new, improved dataset. While this approach increases the variability of the dataset and removes the cascading error issue, it requires to be able to interactively query the expert on new data points. Such a technique is called "on-policy" since the dataset is generated from examples from executions of the learned policy. Conversely, the behavior cloning method described in the previous section is "off-policy" because the learned policy has no influence on the dataset.

1.4.3. Reinforcement Learning

Reinforcement Learning (RL) based robotic controllers try to learn an optimal policy with respect to the reward that they receive from the environment. The RL algorithm usually has to make a choice between exploration, where it will attempt new actions with considerable uncertainty about their reward (possibly getting higher reward than otherwise available), and exploitation, where a safe action with confident reward will be selected. Training such algorithms on expensive hardware is usually not a good idea since the agent might attempt to explore unsafe states, potentially resulting in damaged equipment. This is especially relevant in the case of autonomous driving, where crashing a car incurs considerable costs and possibly injuries of a human supervisor. For this reason, researchers looking to train RL-based controllers typically rely on simulated environments. RL methods exist both in the form of on-policy and off-policy algorithms and fall mainly into two categories:

Value-based. These methods learn a value function V (or state-action value function Q) that estimates *how good* it is to be in a state s (or in a state s taking action a in the case of the Q-value). The policy is then implicit: at each state, the action that is assigned the highest value is chosen. Popular value-based reinforcement learning algorithms include Q-learning [**91**], SARSA [**71**], DQN [**58**].

Policy-based. Policy-based reinforcement learning algorithms avoid estimating the value function and directly learn the policy. REINFORCE [**94**] and DPG [**77**] algorithms are popular examples that falls in this category.

1.5. Simulators

1.5.1. Success of Data-Driven Techniques

The success of deep learning for tasks with vision in the loop can be attributed to the availability of big labeled datasets [**81**]. Some well-known annotated datasets in the field of robotics including KITTY [**32**], Cityscapes [**26**] and Oxford RobotCar [**54**] have been especially useful to improve state-of-the-art performance for tasks such as optical flow, object tracking, SLAM, visual odometry, and semantic segmentation. Using deep learning to achieve these tasks induces a dependency on a big dataset and out-of-distribution examples are often expected to be wrongly classified or yield wrong results. For most real-world applications, the public datasets do not represent exactly the target environment and having a model that overfits these datasets is expected to perform poorly. The user is then left with two choices, either collecting a new dataset in their target environment, which is often costly and unpractical, or adapting a pre-trained network. In both cases, they will need to collect annotated examples in the desired environment.

For this reason, simulated environments have seen an increase in popularity. Instead of collecting and annotating thousands or millions of examples manually, a simulator can automatically generate an infinite amount of custom and labeled data for a minimal cost. They allow reproducible experiments and additional customization of the environment, which is especially useful to adjust parameters to hardware changes or enable domain randomization. Amongst others, robots have been successful in solving a Rubik's cube [**1**], piloting a drone in a new environment [**72**] and grasping objects [**21**] thanks to training in simulation.

1.5.2. Fidelity-Efficiency Trade-Off

Data collected from a simulator is expected to be of lower quality than real-world data. Nonetheless, it is usually a trade-off that we are willing to make when training deep learning agents. For such data-driven techniques, efficiency is crucial and fidelity can usually be neglected for some aspects that are not relevant to the task. As such, being able to generate low-quality annotated data at a high rate is often preferred to manually obtaining real-world data. The same can be said about the level of fidelity of simulators: a less realistic simulator is sometimes preferred to a full-fledged one that accurately simulates fluids and lightning, given that the latter has a higher data generation cost in terms of computational resources.

1.5.3. Flaws of Simulated Environments

In addition to saving resources, using simulators has the benefit of being safe. Training a model on a real robot has the risk of damaging the hardware. During the exploration, a policy might try to reach states that are not safe for the robot. In the case of autonomous

driving, training an algorithm directly on a car can be a hazard for the supervisor (human), the environment (pedestrians), and the hardware (expansive car).

On the other hand, it often happens that simulators contain glitches. By definition, an optimization algorithm will try to be the most efficient possible, which means exploiting bugs, glitches and/or inaccuracies in a simulation to get better results according to the scoring criteria. This behavior has been observed in [**48**] where an agent that was trying to jump as high as possible according to the minimum height of its feet. The agent was able to optimize this metric by learning how to somersault, without performing a real jump. This kind of behavior can reach high performance in the simulated environment but is not useful for any task on the real robot that would require it to jump over an obstacle. Similar behavior was observed in [**4**] where agents learn to abuse the simulator physics to *surf* objects and in [**45**] where the robot was able to generate a trajectory that clips through wall corners to shorten its path. Using a simulated environment requires careful tuning and analysis to prevent undesirable behavior when optimizing a policy.

1.5.4. The *I.I.D.* Assumption

Most predictive models assume that the data is *independent and identically distributed* (IID). In the case of robotics, the first part of the assumption, *independent* data, is usually resolved by the fact that we are assuming the *Markov property*. The second part of the *IID* assumption means that there cannot be trends or sudden changes in the target distribution that are not captured by the data. Unfortunately, by using simulated data we are not respecting this assumption, unless the simulator is perfectly faithful (or the target domain is also the simulation). Our analysis is not about training models in circumstances where the *IID* assumption does not hold, for that we refer the reader to other work in that direction: [**80, 73, 51**]. We will instead focus our research on observing the usefulness of using out of distribution data for generating models.

1.5.5. The Reality Gap

The discrepancy between the simulator and the embodied task usually cause a performance drop once the policies are transferred to the real robot, a phenomenon known as the "Reality Gap". Fundamentally, the source of the performance discrepancy resides in the graphics, dynamics or environment model of the simulation. However, even with careful tuning and improvement of a simulator, it would not be possible to completely remove the reality gap since there will always be unmodeled (or incorrectly modeled) effects at lower level. Aerodynamics, fluids dynamics and friction are some example of interactions that are often disregarded or approximated in simulation, resulting in performance discrepancy once a policy is transferred to the real world. It is often said that "no simulator is perfect" [**59, 46**]

and as such, reducing the reality gap through improvement of the simulation is a limited solution. Following this idea, multiple work have been trying to "cross the gap" by mitigating or reducing the impact of the domain discrepancy. The most popular technique to achieve good performance transfer is Domain Randomization, which consist of using different visual or dynamics parameters at each training episode, preventing the learning algorithm to overfit to the simulated environment. In the best case scenario, the target environment will appears as an instance sampled from the same distribution. Data Augmentation, Meta-learning, Domain Adaptation, Learning common representations and Transferability Optimization are other available techniques that have had success in improving sim-to-real performance. A more in-depth analysis of these method is presented as part of the first research paper in section 3.2.1

Chapter 2

Prologue to first paper

2.1. Article details

On Assessing the Usefulness of Proxy Domains for Developing and Evaluating Embodied Agents, by Anthony Courchesne, Andrea Censi and Liam Paull. The article was submitted and accepted at *International Conference on Intelligent Robots and Systems* (IROS) 2021.

Personal Contribution

The problem was first identified by Liam Paull. Liam came up with the first version of the metrics proposed and we worked on the solutions and wrote the paper together. I took care of the coding, training the neural networks, performing the real and simulated robot runs, and the coding required for the experiments. Andrea Censi handled the base of the Duckietown infrastructure and provided additional analysis of the problem, which ended up being moved to an ulterior work.

2.2. Context

Simulators have seen a steep increase in popularity for robotics, but the transfer to the real robot is an arduous process that usually yield unpredictable results. In an effort to better understand the causes of the reality gap and how to assess and mitigate it, we offer a novel analysis of the value of a proxy environment. By providing clear metrics, new optimization angles are available to maximize the usefulness of our proxy environment, with respect to the task at hand.

A big part of the work done as part of my Master's reside in the infrastructure of Duckietown [**64**] and its simulator [**17**], both of which were used extensively for the experiments in this paper. We also contributed to and used the Autolabs [**85**] (included in 5), although since

the situation with the pandemic made the infrastructure unavailable for extended periods, the use we were able to make of them was limited.

2.3. Contributions

We offer a novel analysis of the domain shift between a proxy and target environment. We establish two different uses for proxy environments and offer new metrics to quantify the value of a simulator for each of them. In the case of a proxy used to predict which agent perform the best, we suggest the Proxy Relative Predictive Value (PRPV) and argue that the value of a proxy as a predictor should be task-dependent and agent agnostic. When the proxy is used as an intermediate tool for learning agents, we offer the Learning Proxy Value (PLV) and claim that in this case, the metric should be specific to both a task and a learning agent. We demonstrate the usefulness of our metrics empirically in the context of Duckietown and compare them to other available metrics available to quantify the environment discrepancy. We show how our metrics can be used to tune a proxy and improve its predictivity.

Chapter 3

On Assessing the Usefulness of Proxy Domains for Developing and Evaluating Embodied Agents

Abstract

In many situations it is either impossible or impractical to develop and evaluate agents entirely on the target domain on which they will be deployed. This is particularly true in robotics, where doing experiments on hardware is much more arduous than in simulation. This has become arguably more so in the case of learning-based agents. To this end, considerable recent effort has been devoted to developing increasingly realistic and higher fidelity simulators. However, we lack any principled way to evaluate how good a "proxy domain" is, specifically in terms of how *useful* it is in helping us achieve our end objective of building an agent that performs well in the target domain. In this work, we investigate methods to address this need. We begin by clearly separating two uses of proxy domains that are often conflated: 1) their ability to be a faithful predictor of agent performance and 2) their ability to be a useful tool for learning. In this paper, we attempt to clarify the role of proxy domains and establish new *proxy usefulness* (PU) metrics to compare the usefulness of different proxy domains. We propose the *relative predictive PU* to assess the predictive ability of a proxy domain and the *learning PU* to quantify the usefulness of a proxy as a tool to generate learning data. Furthermore, we argue that the value of a proxy is conditioned on the *task* that it is being used to help solve. We demonstrate how these new metrics can be used to optimize parameters of the proxy domain for which obtaining ground truth via system identification is not trivial.

3.1. Introduction

Developing and evaluating agents for physically embodied systems such as robots in the setting that they are meant to be deployed in can be costly, dangerous, and time consuming. As a result, it is often desirable to have some *proxy* of the target task domain, which can be a simulation environment or some simpler smaller scale real environment. The fidelity or accuracy of this proxy domain is important in the sense that we need it to be a *faithful predictor* of performance on the target domain.

Recent breakthroughs in machine learning have increased the popularity of data-driven approaches to solve tasks with embodied agents. Two prevalent paradigms include reinforcement [**55**] and imitation [**98**] learning. As a result, the value of data has increased dramatically. Obtaining data from robots is costly since it requires deployment of hardware, operation of the robot and time-consuming rollouts of policies. Moreover, the policy exploration process might end up in hardware damage while unsafe states are explored. This further motivates the need for a proxy domain, but the objectives here are somewhat different. In this case we want to *efficiently* acquire knowledge in the proxy domain that can be *transferred* to the target domain.

There may be other advantages of the proxy domain for the purposes of agent learning, such as: the availability of ground-truth data such as robot localization, reduced cost of data collection, ability to generate and randomize large numbers of samples, customization of scenarios to facilitate the learning of edge cases, parallelization of the data generation process, and others.

Any domain used as a proxy for a target domain introduces a set of discrepancies, often resulting in performance reduction. The "reality gap" is a term that is typically used in the context of sim-to-real, although the concept can be generalized to any domain transfer. For this reason, we will instead use the term "domain gap" as a more generic case. The domain gap has been observed in multiple works, and the community has found multiple ways to mitigate its effects (i.e. "crossing the gap"), without actually explicitly quantifying it. As a result, algorithms that show good performance in some settings (e.g. when the domain discrepancy is small) may completely fail in others, and there are no methods of predicting transferability a priori.

In this work, we quantify the usefulness of a proxy domain by providing *proxy usefulness* (semi)metrics. We make a clear distinction between proxies used to predict task performance (used to select agents according to their performance) and data-generating proxies (used to generate data samples or policies). In the first case, we propose the Proxy Relative Predictivity Value (PRPV), a metric to quantify the predictivity of a proxy, enabling researchers to find the most predictive proxy available to them. We also prescribe a metric to assess the usefulness of a proxy to generate learning data or trained agents, giving the possibility to

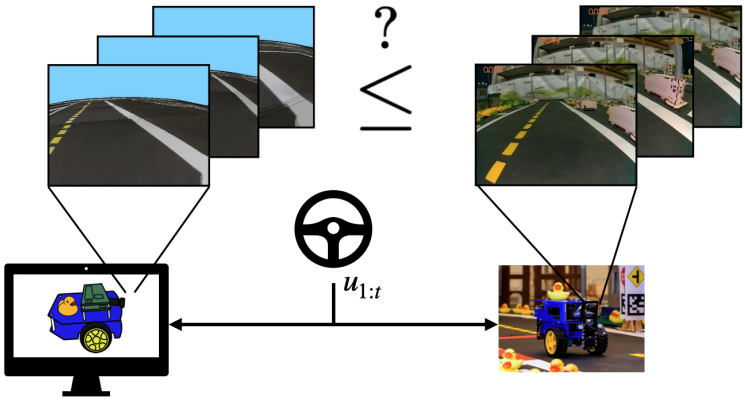


Fig. 3.1. How do we compare the value of two different instantiations of the same task? This is a common setting in robotics where it is easier to develop and evaluate on a simulator or proxy domain before deploying on real hardware (the target domain). If we start the agent at the same state in both domains and execute the same sequence of control signals, we will get different output measurements due to discrepancies in dynamics, appearance, rendering, and many other factors. We argue that directly comparing these observations is not an appropriate measure of the proxy's value, since it does not consider the consequence of the actions under the task specification. Instead, we propose measures for assessing the value quantitatively both in terms of how useful the proxy domain is in terms of predictivity and as a tool for learning agents.

researchers to compare different data-generating domains and select the one that yields the best agents.

Our new metrics allow robotics practitioners to tune some parameters of their proxy domain for which system identification is non-trivial and ground truth value for the target domain is often not available, such as observation blur, input delay, field of view, etc.

3.2. Related work

In the most common configuration, the source domain is a simulator and the target domain is a real robot, in which case the policy transfer is called "sim-to-real". However, it is also frequent to see "sim-to-sim" ([**43, 34, 99**]), or even "real-to-real", such as in the case of [**64**] where small, cheap robots are used to represent an expensive car, or in [**96**] where authors used real-to-real to validate their sim-to-real performance. Our analysis of the domain discrepancies applies to all of these cases.

3.2.1. Crossing the Domain Gap

Multiple techniques exist to mitigate the effects of the domain gap. One such promising technique that has received a lot of attention lately is domain randomization [**88**]: during training, a different proxy is sampled from a domain family (see 3.3) for each run, preventing the model from overfitting to a specific instance. In the best-case scenario, the target domain will appear as another sample from the same distribution. Data augmentation is another

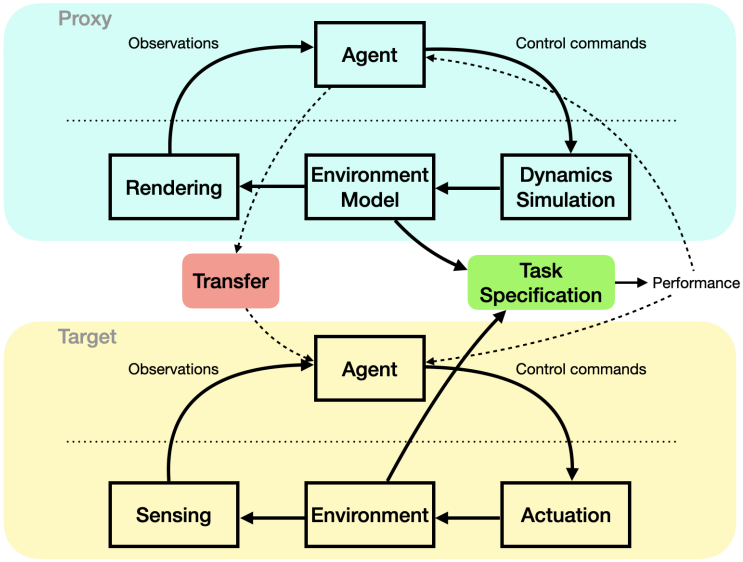


Fig. 3.2. The agent-environment interface for proxy (simulation) and target (real robot) domains. An agent receives observations and generates control commands. The **predictive** value of a simulator lies in its ability to faithfully reproduce an estimate of the task performance. A simulator may also be used in a **learning** paradigm. In this case, the value of the simulator lies in how many fewer trials we need to perform on the real robot to achieve equivalent performance.

popular solution [**75, 63, 12**] where a model is trained to augment data generated by a proxy to match that of the target domain. Meta-learning and domain adaptation are also often used to transfer the knowledge acquired in a source domain to a different domain [**98, 11**]. The authors in [**60, 59**] show that it is possible to find which controllers are most likely to transfer well to the target setup using only a few sample experiments on the real robot, a method they call "The transferability approach". Crossing the domain gap was also shown to be possible by learning a common representation for both source and target domains, either by imposing a common state representation using an autoencoder and a disciminator network [**101**] or by forming weakly aligned pairs of source and target data, effectively transferring the annotations to the target images [**89**].

3.2.2. Optimizing Proxies

In our work, we are more interested in the different ways to analyze the domain gap and improve the proxy rather than circumvent its effects. The first step to reduce the gap imposed by a proxy is to do proper system identification [**84**]. For a specific domain pair, it is possible to carry out standardized tests to find inconsistencies between models [**65**]. It is common [**16, 20, 83**] to use a task performance metric to optimize proxy parameters such as actuator gains, masses, static friction, etc. Recent papers have also shown success in improving their proxies by adding a module that learns inverse dynamics to adapt the action that the proxy outputs to the expected action in the target domain [**18, 38**].

3.2.3. Quantifying Domain Discrepancies

Being able to quantify the domain gap may lead to better prediction of the transferability of agents. As such, multiple attempts have been made by the community to obtain a metric to quantify this gap. One promising work in that direction establishes the sim-to-real disparity: a metric specific to an agent that predicts how well it will transfer to the real robot [**46**]. By conducting only a few experiments both on the real robot and in simulation, it is possible to learn a surrogate model that will estimate the fitness function on the real robot via interpolation. However, it is assumed that we have access to a quick way to compute the distance between two controllers in a domain such that similar controllers will have a similar transferability (e.g. behavioral features), which is often not the case. Moreover, we are interested in a way to quantify the domain gap at the level of a simulator instead of at the agent level. We believe that the domain gap should be agnostic to the agent.

The "ν-gap" [**90**] is a metric used in robust control theory to analyze the discrepancy between two feedback control system. It has been used [**79**] to quantify the discrepancy between robotic systems without any knowledge about their dynamics since it compares two different controllers in a black-box fashion. The metric is computed by measuring the largest chordal distance between the two controllers projected onto the Riemann sphere. It is then possible to minimize it to find the most representative domain. This metric has the limitation of comparing only the dynamics and being only available for linear systems.

A version of "simulator fidelity" has been used as a metric in a multi-fidelity simulator environment [**29**] to define the fidelity of a domain Σ_i to a target domain Σ_j. The fidelity is defined by the maximum error in the optimal value function. Using this metric, it is possible to optimize the training of reinforcement-learning-based agents using multiple simulators, taking into account the cost of generating data for high-fidelity simulators. On the other hand, this definition is inconvenient when used to assess a simulator since the value function is not always trivially computed, especially in the real world. We believe that the definition of the domain gap should not be tightly coupled with machine learning, and it should be independent of the algorithm used.

In [**45**], the authors were able to create a high-fidelity virtual domain for their task using real-to-sim. Given agents and a task, they used the Pearson correlation coefficient between runs on the real robot and in simulation to evaluate the reality gap. This allowed them to find that the noise model for their dynamics was wrong and that learning-based agents were overfitting to the simulator domain by using glitches. In their work, they frequently refer to rank inversion to validate their empirical results, which is an idea that is also used in [**49**]. Like the former, we support that the predictive ability of a simulator lay in its capacity to predict the performance ordering of the agents. In our work, we further define the domain gap based on rank inversion. Using a relative metric for the reality gap allows us to use

multiple performance metrics to get a domain gap measure through partial ordering. We additionally propose a separate and orthogonal measure of a simulator's value for learning.

3.3. Preliminaries

We will consider a domain to encapsulate the actuation, environment and sensing, as shown in Fig. 3.2. For a simulated domain, those components translate to a simulation of the dynamics, an environment model and rendering, respectively. The various components in the domain may include tunable parameters, θ. A domain may also optionally provide a scalar reward $r \in \mathcal{R} \subseteq \mathbb{R}$ signal, where a reward function maps an action and the internal environment state to a scalar value. Note that it is entirely possible that a reward can be provided in the proxy domain but not in the target domain since we don't have direct access to the internal states needed to calculate it. Consequently, a domain $S \in \mathcal{S}$ can be considered as something that maps control commands $u \in \mathcal{U}$ to observations $z \in \mathcal{Z}$, conditioned on some environment model state $x \in \mathcal{X}^{env}$ and parameters $\theta \in \Theta$ ($S : \mathcal{U} \times \mathcal{X}^{env} \times \Theta \mapsto \mathcal{Z} \times \mathcal{R}$).

We will refer to a single domain generated by a specific set of parameters as a **domain instance** S_θ and the set of all domains that are possible to achieve by varying the parameters as a **domain family**, S_Θ.

A **task**, $T \in \mathcal{T}$, is specified through one or more **evaluation metrics**, M, which map a trajectory of N states to a real-valued number:

$$T \triangleq \{M_i\}_{i=1}^{m}, \quad M_i : \mathcal{X}^N \to \mathbb{R} \tag{3.3.1}$$

An **agent**[1] contains the algorithm that is used to generate control commands from the history of observations, the history of control commands, some initial state $x_0 \in \mathcal{X}^{agent}$ (at time t, $A : \mathcal{Z}^{t-1} \times \mathcal{U}^{t-1} \times \mathcal{X}^{agent} \mapsto \mathcal{U}$, or if the state is assumed to be Markovian then $A : \mathcal{Z} \times \mathcal{X}^{agent} \mapsto \mathcal{U}$). Presumably, the algorithm informing this agent is designed to optimize the specified task evaluation metrics. A **learning agent** is able to adapt its behaviour over time by means of a learning algorithm (A at time k is not necessarily the same as A at time $k+1$). However, we assume here that, for a stationary domain and task, the learning agent will *converge* to a stationary agent for some k large enough.

Given a sequence $u_{0:n}$ of control values (generated by any means), a domain instance S_θ can generate a dataset D_{S_θ}, conditioned on an initial state x_0, which is a collection of n labelled data samples each consisting of a tuple of observation z and command u ($D_{S_\theta} = \{(z_i, r_i, u_i)\}_{i=1}^{n}$ where $(z_i, r_i) = S_\theta(x_i^{env}, u_i)$). It is also possible that the parameters, θ, are randomized over the domain family during dataset generation. In this case we have $D_{S_\Theta} = \{(z_i, r_i, u_i)\}_{i=1}^{n}$ where $(z_i, r_i) = S_\theta(x_i^{env}, u_i)$ and $\theta \sim \Theta$. Such datasets are useful as

[1] We distinguish an agent from the more standard notion of *policy* in that a policy maps internal states to actions

demonstrations for the learning agent, for example in imitation learning algorithms [**70**] which try to reproduce the behaviour of an expert.

In an on-policy reinforcement learning setting, the control actions are selected at every timestep: $u_i = A(z_i, x_i^{agent})$. The domain will update its state according to the command and will return the next observation and optionally a reward to the agent. Normally the agent will be able to use the reward to update its algorithm. Again the on-policy rollouts can be executed on the same domain instance or on random samples from the simulator family (as is the case in domain randomization [**88**]) [2].

Our objective in this work is to provide measures to quantify the *usefulness* of a proxy domain. We will use the term "Proxy Usefulness (PU)" to quantify the value of a proxy domain.

3.3.1. A Naive Measure of Proxy Usefulness

Naively, one could define the PU of a proxy domain by its discrepancies with the target domain it is expected to represent. Following, the usefulness of a proxy domain would be inversely proportional to its domain gap with the target domain and, according to (Fig. 3.2), would be defined by:

Def. 1 (Proxy Observation Discrepancy (POD)). A proxy is deemed more useful inasmuch as the "difference" in the resulting observations produced by the proxy's and the target's robots for a predefined sequence of control commands is small.

For a sequence of control values $u_{0..n}$ and initial state, this could simply be calculated as:

$$||S_\theta^{proxy}(x_i^{env}, u_i) - S_\theta^{target}(x_i^{env}, u_i)|| \tag{3.3.2}$$

where in this case we are primarily concerned with the observations that are output and not the rewards.

There are several issues with Def. 1:

(1) It combines in an opaque way the various sources of the discrepancy. Referring to Fig. 3.2, there could be a "gap" in the dynamics, the environment model (for example how other agents move in the environment), or in the generation of sensor data based on a rendering model. Moreover, errors in upstream models will compound.

(2) It is agnostic to the task that the agent is trying to solve. Many proxies should be deemed perfectly faithful if a trivial task is chosen, but that will not be the case here.

(3) It presupposes that the fidelity in the target domain is needed. In practice, we only require a form of task-conditioned fidelity: only the things that are important to solve the task at hand must be faithfully reproduced in the proxy.

(4) It in no way represents how useful the proxy is for learning.

[2] The instances may not be sampled randomly, as is the case in [**56**]

In the remainder of this paper, we offer alternative ways to evaluate the PU to address these issues.

We make a clear distinction between two different uses of a proxy: 1) The *predictivity* value of a proxy domain (given a target domain) is its ability to generate accurate predictions about the performance of agents in the target domain. The *teaching* value of a proxy domain encapsulates how useful it is as a tool to train *learning agents* that perform well on the target domain.

3.4. The Proxy as a Predictor

The first "value" of a proxy domain is as a tool to predict. However, different from Def. 1, we argue that the domain's ability to predict *task performance* rather than exact observations is what is relevant:

Def. 2 (**Proxy Predictivity Value (PPV)**)**.** Given a task T defined by evaluation metrics $M_{1:m}$, and an agent A that generates trajectory $x^{1:N}_{proxy}$ in the proxy domain and $x^{1:N}_{target}$ in a target domain given equivalent starting conditions, then we define the PPV of a proxy S_θ as the discrepancy of the resulting evaluation metrics:

$$\text{PPV}(S_\theta, A) \triangleq \sum_{i=1}^{m} \beta_i |M_i(x^{1:N}_{proxy}) - M_i(x^{1:N}_{target})| \tag{3.4.1}$$

where the β_i terms are weighting constants that can account for mismatched units or possibly increased importance of one metric over another. A proxy is deemed more useful if it has a lower PPV.

By Def. 2, a proxy domain can be considered *perfectly faithful* to a target domain for a given task if the PPV is zero for all possible agents. This definition is in some sense a generalization of the Sim-vs-Real Correlation Coefficient (SRCC) [**45**] to the case where there are multiple metrics that define the task, except with a 1-norm distance instead of the bivariate correlation.

The need for the β constants in Def. 2 is undesirable since it allows some room for subjectivity that can effect the results. This can be avoided by considering that in many cases we are interested in *comparing* agents rather than finding exact evaluations of the metrics. As a result, we can consider a relaxation of Def. 2 to the relative case. Given that a task may contain several evaluation metrics, agents can be arranged in a partial ordering whose binary relation $\leq$ is defined by dominance along all of the available metrics:

$$A_1 \geq A_2 \rightarrow M_i(X_1) \geq M_i(X_2) \quad \forall i \tag{3.4.2}$$

where X_j is shorthand for the trajectory produced by agent A_j (either in the proxy or in target domain).

Def. 3 (**Proxy Relative Predictive Value (PRPV)**)**.** Given K agents, the relative predictive ability of a proxy S_θ is defined by its ability to accurately predict the binary relations between agents that would be present in the target domain. Let $\mathcal{A}^{proxy} = [\alpha_{ij}^{proxy}]_{i,j=1..K}$ be a matrix whose entries are given by:

$$\alpha_{ij}^{proxy} = \begin{cases} 1 & A_i^{proxy} \geq A_j^{proxy} \\ 0.5 & A_i^{proxy} \not\geq A_j^{proxy} \quad \& \quad A_j^{proxy} \not\geq A_i^{proxy} \\ 0 & A_i^{proxy} \leq A_j^{proxy} \end{cases} \tag{3.4.3}$$

where A_j^{proxy} (A_i^{proxy}) is agent j (i) applied to the proxy domain. We similarly construct $\mathcal{A}^{target}$. Then the PRPV of a proxy domain is given by the 1-norm between the two matrices that represent the relations in the two partial orders [**25**]:

$$\text{PRPV}(S_\theta, A_{1:K}) = \sum_{i,j=1}^{K} |\alpha_{ij}^{proxy} - \alpha_{ij}^{target}| \tag{3.4.4}$$

According to Def. 3, a proxy domain is now perfectly faithful if it produces the identical partial order over agents that would be produced if the agents were run on the real robot. This is closely related to the concept of "rank inversion" [**50**].

Note that in the case of both PPV and PRPV, the value of the domain is *conditioned* on the task and *agnostic* to the agent (only requires some method of generating trajectories). Also note that in practice the performance of the agent in a simulated domain or (especially) in the real domain will be stochastic and therefore PPV and PRPV should be redefined as metrics over distributions and approximated by sequences of trials, but we omit this here for clarity.

3.5. The Proxy as a Teacher

Orthogonal from the proxy domain's predictivity, it may have value as a tool for agents that *learn* (3.3). That proxy domain now becomes a part of the agent generation process since, as shown by the dashed lines in Fig. 3.2, the task performance may be fed back to the agent. We can assess the usefulness of the proxy domain by evaluating the performance of the agents that it trains on the target domain, compared to agents that learn entirely on the target domain. A proxy domain is deemed more valuable if it reduces the number of trials that are needed in the target domain. One naive option to evaluate a domain transfer method would be to consider the zero-shot (or N-shot) performance on the target domain. However, similar to the argument we made in Def. 2, the outputs of these metrics may not calibrate well to the actual learning that has taken place. Instead, we define the usefulness for learning explicitly as what we are trying to minimize through using the simulator for training: the

number of trials on the target domain required to achieve equivalent performance as we would achieve if we had not pre-trained in the proxy domain.

Def. 4 (Proxy Learning Value (PLV)). Consider that a learning agent, A^{target} trained entirely in the target domain is able to achieve a performance of $M_{i..m}^{target}$ on task T *at convergence* using dataset S^{target}. An agent $A^{proxy \to target}$ pre-trained on the proxy domain and then transferred to the target domain and fine-tuned until it achieves an equivalent performance $M_{i..m}^{proxy \to target} \geq M_{i..m}^{target}$ using dataset on the target domain $S^{proxy \to target}$. Then, the PLV is given by:

$$\text{PLV} = |D^{target}| - |D^{proxy \to target}| \tag{3.5.1}$$

Note here that, in contrast to the predictivity, the usefulness of the proxy as a teacher is conditioned on *both the task and the learning algorithm* used to train the agent. In addition, different learning algorithms may leverage the proxy domain in different ways. Behavior cloning methods may generate a dataset that includes expert trajectories from both domains [**9**]. In this case the usefulness is determined by the reduction in the size of the dataset, D^{target} from the target domain (an example we demonstrate in Sec. 3.7.2). As noted in Sec. 3.3, the proxy domain dataset can be generated from a single domain instance or over the domain family for increased robustness. In either case, the definition of the usefulness value is unchanged.

In reinforcement learning, the agent is learning through interaction with the environment [**19**]. In this case the data is generated through trials and this number of trials is what we seek to minimize. Again, it is entirely possible that the training episodes in the proxy environment randomize over the domain family, as is the case in domain randomization [**88**].

In some cases, both off-policy data and on-policy rollouts may be used. Such is the case in approaches that leverage domain adversarial transfer [**101, 6**]. Off-policy data is used to learn the mapping from proxy domain to the source domain (for example using a discriminator). At test time, this mapping is applied but then fine-tuning is achieved with on-policy trials. A generalization of the PLV may consider these two cases (on-policy and off-policy) as being scaled differently. A natural priority would be to minimize the number of on-policy trials on the target domain at the expense of off-policy data, which may be easier and safer to obtain. As such we can generalize (3.5.1) to:

$$\begin{aligned} \text{PLV} = \eta_{on}(|D_{on}^{target}| - |D_{on}^{proxy \to target}|) \\ + \eta_{off}(|D_{off}^{target}| - |D_{off}^{proxy \to target}|) \end{aligned} \tag{3.5.2}$$

where η_{on} and η_{off} represent the relative importance of off and on-policy data.

3.6. Optimizing the Proxy

Using any of the proxy domain metrics defined above, we can optimize the set of parameters θ of the proxy instance.

In the case of predictivity, we desire to minimize the PRPV:

$$\theta^*(A_{1:K}) = \arg\min_{\theta} \mathrm{PRPV}(S_\theta, A_{1:K}) \tag{3.6.1}$$

which will yield the parameters that are most predictive of the task metrics. The residual of this optimization defines the irreducible domain gap and is due to the fact that the proxy models may be limited in their capacity. For example, if wheel slip is not modeled in the proxy domain but it exists in the target domain, we will always incur error.

In the case of teaching, we desire to maximize the PLV. It is possible that we wish to find the single best proxy instance, but more recent methods leverage randomization for increased robustness. In this case, we may wish to optimize for the optimal distribution or sequence of domain instances (e.g., as in [**56**]), a problem that can be formulated as curriculum learning:

$$p(\theta)^* = \arg\max_{p(\theta)} \mathrm{PLV}(\theta) \tag{3.6.2}$$

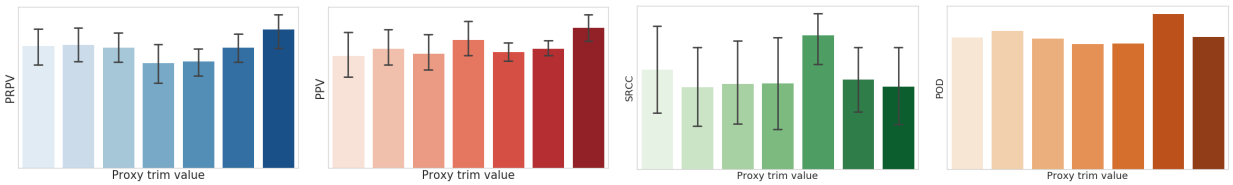
or possibly over the domain itself:

$$S^* = \arg\max_{S} \mathrm{PLV}(S) \tag{3.6.3}$$

These optimization may be solved with gradient-based methods in the case that the proxy domain is *differentiable* [**61, 41**]. Evaluating the loss function may incur significant cost, however, particularly in the case of the PLV where training an agent in the proxy domain and evaluating it in the target domain is needed. For these cases, an approach such as Bayesian Optimization or some approximation may be more appropriate [**78**].

3.7. Experiments

We evaluate the proposed metrics in the context of Duckietown [**64**], a platform to run vehicle controllers both in a simulator and on a real robot seamlessly. The platform offers a simulated domain ([**17**]) where domain randomization is available, as well as multiple customizable features (see Fig. 3.5). A challenge server is in place such that the users can easily submit a Docker container to the server, which will evaluate it and returns footage of the runs as well as performance metrics. Duckietown now additionally offers the evaluation of runs on the real robot at the Autolab through the challenge server [**85**]. The Autolab is an embodied domain monitored by watchtowers. As such, users can receive the ground truth position of the robot at each timestamp as well as a precise trajectory of the performance of their agent.



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

Fig. 3.3. Comparison of proxy usefulness metrics. From left to right: PRPV, PPV, SRCC, POD. For each proxy, the metric is reported for a linear range of trim value from -0.3 to 0.3. The ground truth, 0.0, is always in the center.

3.7.1. Duckietown Simulator Predictivity

We collected 10 agent submissions from a lane-following challenge from the AI Driving Olympics (AIDO) [**102**]. Each submission consist of a controller that is designed to drive a Duckiebot along a lane, following the center as closely as possible. The environment configuration in our experiments was a simple 3-by-3 Duckietown map, in which every run would last 60 seconds or until the robot crashes. For a given Duckiebot trajectory, the task is specified through two performance metrics: M_1: distance traveled along a lane and M_2: survival time, capped at 60 seconds. Since each agent had multiple runs in both the proxy and target domain, the metrics reported are actually means of multiple entries[3].

The target domain was set to be the embodied lane following AIDO domain, while the proxy domains were instances of the Duckietown simulator [**17**]. We selected a list of parameters consisting of *trim value*, which controls the wheel trim, *command delay*, which simulates latency in the control loop, *blur time*, which is used to simulate camera blur, and *camera angle*, which is the pitch angle of the camera. For some of these parameters (namely the trim value and camera angle) we know the ground truth value on the real Duckiebot. In the case of the trim value, the robot is made to go straight using an odometric calibration procedure, therefore the ground truth value in the simulator should be 0.0. For the camera angle, we can determine it though extrinsic camera calibration.

As a result, we can verify that our metrics are correct since the best predictivity should correspond to the ground truth value. We investigate this for the case of trim in Fig. 3.3, and compare the scores for SRCC (Fig. 3.3c) [**45**], POD (Fig. 3.3d), PPV (Fig. 3.3b) and our proposed PRPV (Fig. 3.3a). We can see from the plots that the PRPV identifies the correct trim value, and also shows monotonic decrease in predictivity as the trim value increases. By writing our metric in terms of agent ordering explicitly we are able to capture the essence of how well the agents are able to perform the task.

[3]The results can be found at `https://challenges-stage.duckietown.org/humans/challenges/anc-01-LF-sim-validation/leaderboard`

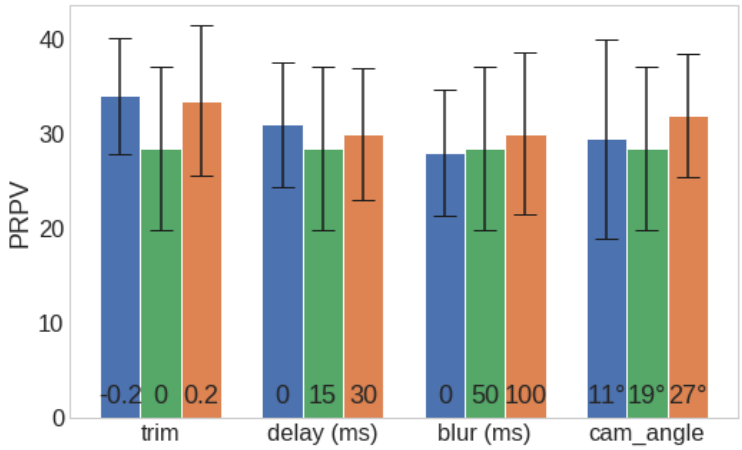


Fig. 3.4. Evaluation of different instances of gym_duckietown as a proxy for the AIDO embodied challenge according to the PRPV. The green bars represent the parameter value that was used for AIDO. Using our metric we can now determine which of our parameters were correct and which should be modified.

We apply a similar approach to the other tunable parameters. For each simulator instance, we report the PRPV in fig 3.4 for three different parameter values centered on the ones that were actually used in the recent AIDO competition.

As expected, the results in fig. 3.4 show that having a non-zero trim value highly reduces the predictivity of our simulator. We also observe that the *command delay* that was initially set to 50 ms seems to be good, same for the *camera angle*. However, in the case of the *blur time*, our simulator could make better predictions if we decrease the parameter value. To obtain the optimal simulator instance, a second iteration of that experience could be repeated, using the new information to choose the range of the parameters to explore.

3.7.2. Duckietown Simulator for Learning

To compute the PLV, we collected annotated data from an expert that has access to ground truth both in the proxy and target domains (fig. 3.5). We then used the algorithm of the controller that placed first during AI-DO3 [**68**], which is based on imitation learning [**9**], to obtain a learning-based agent capable of following a Duckietown lane. Again, the environment configurations was set to be a 3-by-3 Duckietown map, where the agent has to drive within the outer lane for 60 seconds. We note the performance of an agent by the amount of time it can drive, where each second spent with one wheel outside the lane is subtracted and each second spent with both wheels outside the lane is subtracted twice. If the agent drives out of the map entirely, the run ends.

Computing the PLV requires comparison of the performance of an agent with and without the proxy domain. We trained the neural network based on various amount of real world samples, each time deploying on the real robot and observing its performance. We observed

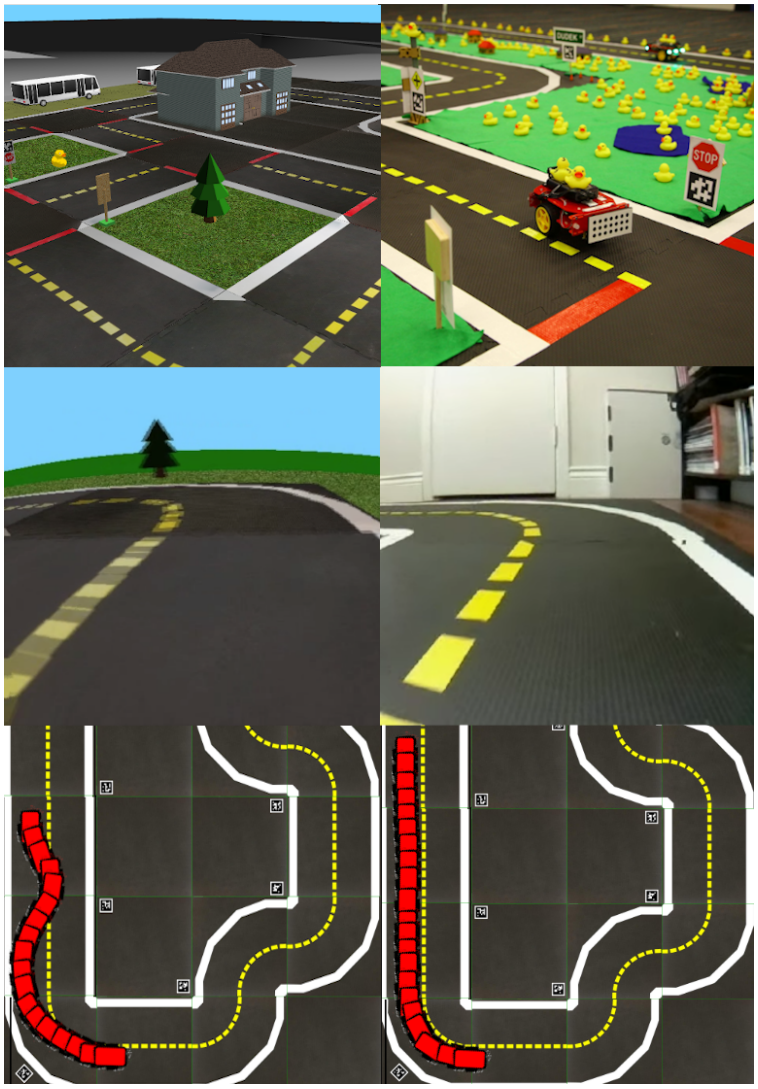


Fig. 3.5. Proxy (left) and target (right) domain of Duckietown, displaying the environment, observation and trajectories.

that the agents converge at a performance of around 55 seconds score (that is, they survive the whole 60 seconds run and get around 5 seconds of penalty), which was obtained with 9000 annotated data points from the target domain. Afterwards, we performed the same experiment, but we initialized the weights of the network from a neural network trained entirely with data from a simulator-based proxy domain.

The experiment was done for three different proxy instances where we modified the angle of the simulated camera to 3°, 19° and 35°, respectively. The learning curves for the four cases are shown in Fig. 3.6 and results are presented in Table 3.1. We see here that the zero-shot score on the target domain is not a representative value for how useful the simulator is for learning. The zero-shot scores for 'sim_ca03' and 'sim_ca35' are very low compared to 'sim_ca19'. However, they are both still very useful since they dramatically reduce the

Proxy	zero-shot score	target domain samples	% reduction
sim_ca03	4.0	1000	88
sim_ca19	47.4	250	97
sim_ca35	5.4	750	92
none	0	9000	0.0

Table 3.1. Zero shot score and number of target domain samples required to achieve real world performance at convergence on the robot according to proxy used for pre-training.

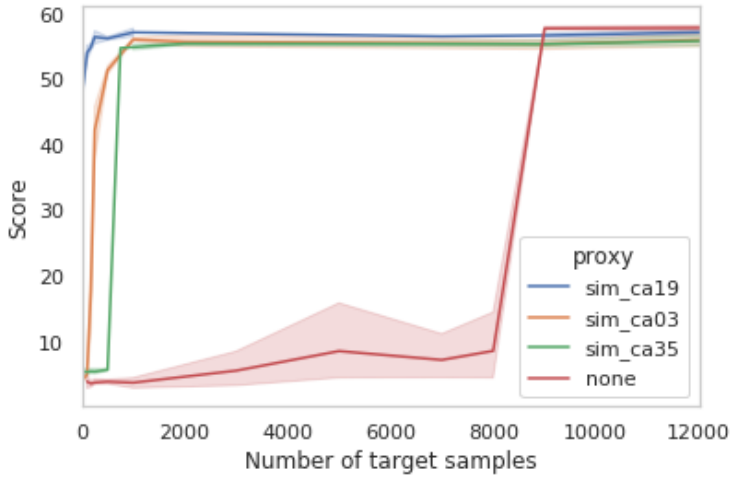


Fig. 3.6. Performance of an imitation learning agent in a 3x3 duckietown map lane-following challenge with different proxy domains

amount of target domain data needed compared to training entirely on the target domain. We also see here that these two different uses of a simulator (predictivity and teaching) may tell us different things. A simulator that is not particularly predictive (since the camera pitch angle is way off) may still be useful for learning if the learning algorithm is able to adapt properly.

3.8. Conclusion and Future Work

We introduce new metrics to assess the usefulness of proxy domains for agent learning. In a robotics setting it is common to use simulators for development and evaluation to reduce the need to deploy on real hardware. We argue that it is necessary to to take into account the specific task when evaluating the usefulness of the the proxy. We establish novel metrics for two specific uses of a proxy. When the proxy domain is used to predict performance in the target domain, we offer the PRPV to assess the usefulness of the proxy as a predictor, and we argue that the task needs to be imposed but not the agent. When a proxy is used to generate training data for a learning algorithm, we propose the PLV as a metric to assess usefulness of the source domain, which is dependent on a specific task and a learning algorithm. We demonstrated the use of these measures for predicting parameters in the

Duckietown environment. Future work will involve more rigorous treatment of the optimization problems posed to find optimal parameters, possibly in connection with differentiable simulation environments.

Chapter 4

Prologue to second paper

4.1. Article details

Integrated Benchmarking and Design for Reproducible and Accessible Evaluation of Robotic Agents, by Jacopo Tani, Andrea F. Daniele, Gianmarco Bernasconi, Amaury Camus, Aleksandar Petrov, Anthony Courchesne, Bhairav Mehta, Rohit Suri, Tomasz Zaluska, Matthew R. Walter, Emilio Frazzoli, Liam Paull, Andrea Censi, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020

Personal contribution

I worked extensively on the software architecture of the platform. I was specifically helping develop and maintain the simulator and the interface with the challenge infrastructure. I conducted multiple projects such as upgrading the major OS distribution, ROS, and python versions of the repositories and containers, uniformizing the communication interface between the controller and the environment using UNIX named pipes and ROS, generating 3D models, developing an exercise infrastructure and more. For the paper, I contributed to the discussions as well as the sections about the simulation.

4.2. Context

Benchmarking in robotics has always been a challenge since hardware is costly and often subject to breakage. Scientific claims in robotics are typically made within a single environment, or multiple simulated environment. There is a need for robot-in-the-loop testing and low-cost, reproducible setups.

4.3. Contributions

In this work, we provide a new design where reproducibility is the top priority. The autolabs are publicly available autonomous environments for evaluation of specific autonomous driving tasks. The presence of multiple autolabs in different locations with the same protocol definition, along with the software containerization, enables highly reproducible scientific experiments. Software solutions wrapped in a container are submitted to a simulation, and then to a real robot.

Chapter 5

Integrated Benchmarking and Design for Reproducible and Accessible Evaluation of Robotic Agents

Abstract

As robotics matures and increases in complexity, it is more necessary than ever that robot autonomy research be *reproducible*. Compared to other sciences, there are specific challenges to benchmarking autonomy, such as the complexity of the software stacks, the variability of the hardware and the reliance on data-driven techniques, amongst others. In this paper, we describe a new concept for reproducible robotics research that integrates development and benchmarking, so that reproducibility is obtained "by design" from the beginning of the research/development processes. We first provide the overall conceptual objectives to achieve this goal and then a concrete instance that we have built: the DUCKIENet. One of the central components of this setup is the Duckietown Autolab, a remotely accessible standardized setup that is itself also relatively low-cost and reproducible. When evaluating agents, careful definition of interfaces allows users to choose among local versus remote evaluation using simulation, logs, or remote automated hardware setups. We validate the system by analyzing the repeatability of experiments conducted using the infrastructure and show that there is low variance across different robot hardware and across different remote labs.[2]

5.1. Introduction

Mobile robotics poses unique challenges that have precluded the establishment of benchmarks for rigorous evaluation. Robotic systems are *complex* with many different interacting

[2]The code used to build the system is available on the Duckietown GitHub page (`https://github.com/duckietown`).

Fig. 5.1. The Duckietown Autolab: We augment Duckietowns [**64**] with localization and automatic charging infrastructure. We modify Duckiebots (inset) to facilitate detection by the localization system and auto-charging.

components. As a result, evaluating the individual components is not a good proxy for full system evaluation. Moreover, the outputs of robotic systems are temporally correlated (due to the system dynamics) and partially observable (due to the requirement of a perception system). Disentangling these issues for proper evaluation is daunting.

The majority of research on mobile robotics takes place in idealized laboratory settings or in unique uncontrolled environments that make comparison difficult. Hence, the value of a specific result is either open to interpretation or conditioned on specifics of the setup that are not necessarily reported as part of the presentation. The issue of reproducibility is exacerbated by the recent emergence of *data-driven* approaches, the performance of which can vary dramatically and unpredictably across seemingly identical environments (e.g., by only varying the random seed [**39**]). It is increasingly important to fairly compare these data-driven approaches with the more classical methods or hybrids of the two.

Most existing methods for evaluating robotics operate on individual, isolated components of the system. For example, evaluating robot perception is comparatively straightforward and typically relies on annotated datasets [**33, 27, 15, 82**]. However, performance on these benchmark datasets is often not indicative of an algorithm's performance in practice. A common approach to analyzing robot control algorithms is to abstract away the effects of perception [**100, 67**] and assume that the pose of the robot is known (e.g., as determined using an external localization system, such as a motion capture setup [**57**]).

Simulation environments are potentially valuable tools for system-level evaluation. Examples such as CARLA [**31**], AirSim [**74**], and Air Learning [**47**] have recently been developed

for this purpose. However, a challenge with simulation-based evaluation is that it is difficult to quantify the extent to which the results extend to the real world.

Robotics competitions have been excellent testbeds for more rigorous evaluation of robotics algorithms [**2**]. For example, the DARPA urban challenge [**14**], the DARPA robotics challenge [**44**], and the Amazon Picking Challenge [**28**], have all resulted in massive development in their respective sub-fields (autonomous driving, humanoid robotics, and manipulation respectively). However, the cost and multi-year effort required to join these challenges limit participation.

A very promising recent trend that was spearheaded by the Robotarium at Georgia Tech [**66, 95**] is to provide remotely accessible lab setups for evaluation. This approach has the key advantage that it enables access for anyone to submit. In the case of the Robotarium, the facility itself cost 2.5M $US, and would therefore be difficult to replicate. Additionally, while it does allow users flexibility in the algorithms they can run, it does not offer any standardized evaluation.

In this paper, we propose to harmonize all the above-mentioned elements (problem definition, benchmarks, development, simulation, annotated logs, experiments, etc.) in a "closed-loop" design that has minimal software and hardware requirements. This framework allows users to define benchmarks that can be evaluated with different modalities (in simulation or on real robots) locally or remotely. Importantly, the design provides immediate feedback from the evaluations to the user in the form of logged data and scores based on the metrics defined in the benchmark.

We present the Decentralized Urban Collaborative Benchmarking Network (DUCKIENet), an instantiation of this design based on the Duckietown platform [**64**] that provides an accessible and reproducible framework focused on autonomous vehicle fleets operating in model urban environments.

The DUCKIENet enables users to develop and test a wide variety of different algorithms using available resources (simulator, logs, cloud evaluations, etc.), and then deploy their algorithms locally in simulation, locally on a robot, in a cloud-based simulation, or on a real robot in a remote lab. In each case, the submitter receives feedback and scores based on well-defined metrics.

The DUCKIENet includes Duckietown Autolabs (DTAs), remote labs that are also low-cost, standardized and fully documented with regards to assembly and operation, making this approach highly scalable. Features of the DTAs include an off-the-shelf camera-based localization system and a custom automatic robot recharging infrastructure. The accessibility of the hardware testing environment through the DUCKIENet enables experimental benchmarking that can be performed on a network of DTAs in different geographical locations.

In the remainder of this paper we will summarize the objectives of our integrated benchmarking system in Section 5.2, describe an instantiation that adheres to these objectives, the DUCKIENet, in Section 5.3, validate the performance and usefulness of the approach in Section 5.4 and finally present conclusions in Section 5.5.

5.2. Integrated Benchmarking and Development for Reproducible Research

Benchmarking should not be considered an isolated activity that occurs as an afterthought of development. Rather, it should be considered as an integral part of the research and development process itself, analogous to test-driven development in software design. Robotics development should be a feedback loop that includes problem definition, the specification of benchmarks, development, testing in simulation and on real robot hardware, and using the results to adjust the problem definition and the benchmarks (Fig. 5.2).

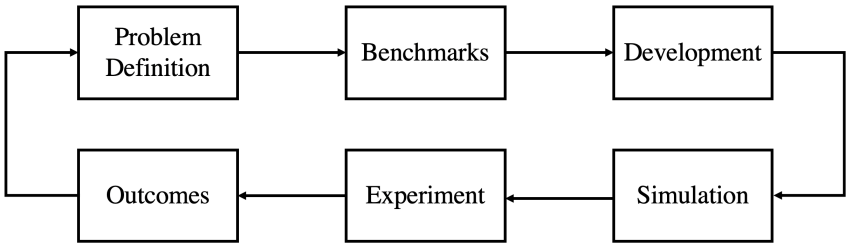


Fig. 5.2. Integrating Benchmark Design: Designing robotic benchmarks should be more tightly integrating into robotic system development.

In this paper we describe a system built to explore the benefits and the challenges of harmonizing development, benchmarking, simulation and experiments to obtain reproducibility "by design". In this section we give an overview of the main challenges and design choices.

5.2.1. Reproducibility

Our main objective is to achieve "reproducibility". At an abstract level, reproducibility is the ability to reduce variability in the evaluation of a system such that experimental results are more credible [**36**]. More specifically, there are three distinct aspects to reproducibility, all equally important:

(1) An *experiment* is reproducible if the results obtained at different times and by different people are similar;
(2) An *experimental setup* is reproducible if it is possible to build a copy of the same setup elsewhere and obtain comparable results;
(3) An *experimental platform* is reproducible if it is relatively easy, in terms of cost and complexity, to replicate.

5.2.1.1. Software. Modern autonomy stacks consist of many modules and dependencies, and the reproducibility target implies being able to use the same software possibly many years later.

Standardized software archival systems such as Git solve part of the problem, as it is easy to guarantee storage and integrity of the software over many years [**7**]. However, even if the core implementation of the algorithm is preserved, the properties of the system change significantly in practice due to the external dependencies (libraries, operating system, etc.). In extreme cases "bit rot" causes software to stop working simply because the dependencies are no longer maintained [**8**].

Containerization technologies such as Docker [**30**] have dramatically reduced the effort required to create reproducible software. Containerization solves the problem of compilation reproducibility, as it standardizes the environment in which the code runs [**92**]. Furthermore, the ability to easily store and retrieve particular builds ("images") through the use of registries, solves the problem of effectively freezing dependencies. This allows perfect reproducibility for a time span of 5–10 years (until the images can no longer be easily run using the current version of the containerization platform).

5.2.1.2. Hardware. In addition to issues related to software, a central challenge in the verification and validation of robot autonomy is that the results typically depend on under-defined properties of the particular hardware platforms used to run the experiments. While storage and containerization make it possible to standardize the version and build of software, hardware will never be fully reproducible, and using only one instance of a platform is impossible due to wear and tear.

As a result, we propose to make several samples from a *distribution* of hardware platforms that are used for evaluation. Consequently, metrics should be considered statistically (e.g., via mean and covariance), similarly to recent standard practices in reinforcement learning that evaluate over a set of random seeds [**39**]. This incentivizes algorithms that are robust to hardware variation, and will consequently be more likely to work in practice. Also, one can define the distribution of variations in simulations such that configurations are sampled by the benchmark on each episode. While the distribution as a whole might change (due to different components being replaced over the years), the distributional shift is easier to detect by simply comparing the performance of an agent on the previous and current distributions.

Cost can also quickly become the limiting factor for the reproducibility of many platforms. Our objective is to provide a robotic platform, a standardized environment in which it should operate, and a method of rigorous evaluation that is relatively inexpensive to acquire and construct.

5.2.1.3. Environment. We use an analogous approach to deal with variability in the environment. The first and most crucial step for reproducibility of the setup is to standardize

the formal description of the environment. Nevertheless, it is infeasible to expect that even a well described environment can be exactly reproduced in every instance. As discussed in Section 5.3, our approach is to use a distributed set of experimental setups (the DTAs) and test agents across different installations that exist around the world. Similar to the way that robot hardware should be randomized, proper evaluation should include sampling from a distribution over environments.

5.2.2. Agent Interoperability

Open source robotics middleware such as ROS [**69**] and LCM [**42**] are extremely useful due to the tools that they provide. However, heavily leveraging these software infrastructures is an impediment to long-term software reproducibility since versions change often and backward/forward compatibility is not always ensured. As a result, we propose a framework based on low-level primitives that are extremely unlikely to become obsolete, and instead provide *bridges* to popular frameworks and middlewares. Specifically, components in our infrastructure communicate using pipes (FIFOs) and a protocol based on Concise Binary Object Representation (CBOR) (a binarized version of JavaScript Object Notation) [**10**]. These standardized protocols have existed for decades and will likely persist long into the future. It is then the job of the infrastructure to interconnect components, either directly or using current middleware (e.g., ROS) for transport over networks. In the future, the choice of middleware might change without invalidating the stored agents as a new bridge can be built to the new middleware. An additional benefit of CBOR is that it is self-describing (i.e., no need of a previously agreed schema to decode the data) and allows some form of extensibility since schema can change or be updated and maintain backwards compatibility.

5.2.3. Robot-Agent Abstraction

Essential to our approach is defining the interface between the *agent* and the *robot* (whether the robot is in a simulator or real hardware). Fundamentally, we define this interface in the following way (Fig. 5.3):

- **Robots** are nodes that consume actuation commands and produce sensor observations.
- **Agents** are nodes that consume sensor observations and produce actuation commands.

Combining this with containerization and the low-level reproducible communication protocol introduced in Section 5.2.2 results in a framework that is reliable and enables the easy exchange of either agents or robots in a flexible and straightforward manner. As described in Section 5.3, this approach is leveraged in the DUCKIENet to enable zero friction transitions from local development, to remote simulation, and to remote experiments in DTAs.

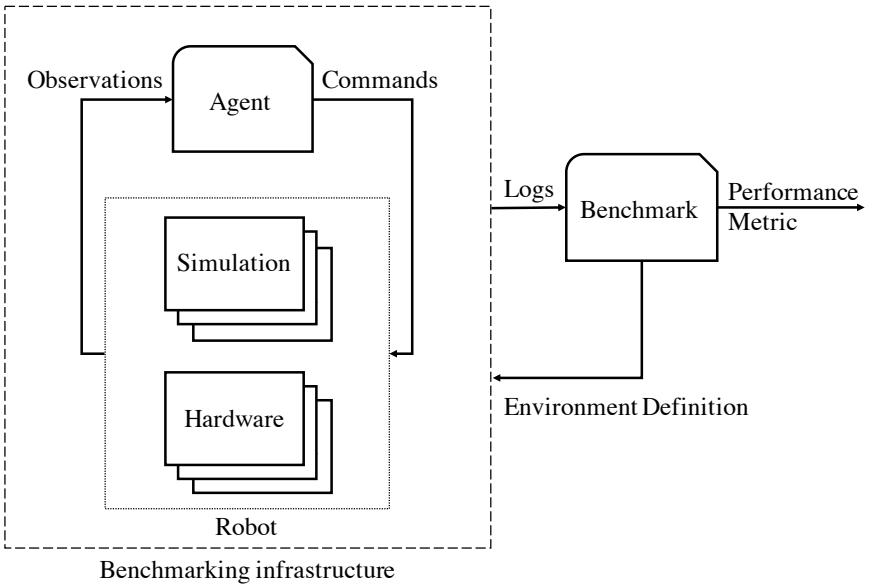


Fig. 5.3. Abstractions and Interfaces: An interface must be well-specified so that components on either side of the interface can be swapped in and out. The most important is the Agent-Robot interface, where the robot can be real hardware or a simulator. The interface between the benchmark and the infrastructure enables a clear definition of benchmark tasks (in terms of metrics) separately from the means by which they are evaluated.

5.2.4. Robot-Benchmark Abstraction

A benchmarking system becomes very powerful and extensible if there is a way to distinguish the infrastructure from the particular benchmark being run. In similar fashion to the definition of the interface for the robot system, we standardize the definition of a benchmark and its implementation as containers (Fig. 5.3):

- **Benchmarks** provide environment configurations and consume robot states to evaluate well-define metrics.
- **Infrastructure** consumes environment configurations and provides estimates of robot state, either directly (e.g., through a simulator) or through other external means.

5.3. The DUCKIENet

In this section we describe an instantiation of the high-level objectives proposed in Section 5.2 that we refer to as DUCKIENet, which:

- comprises affordable robot hardware and environments,
- adheres to the proposed abstractions to allow easy evaluation on local hardware or in simulation,
- enables evaluation of user-specified benchmarks in the cloud or in a remote, standardized lab.

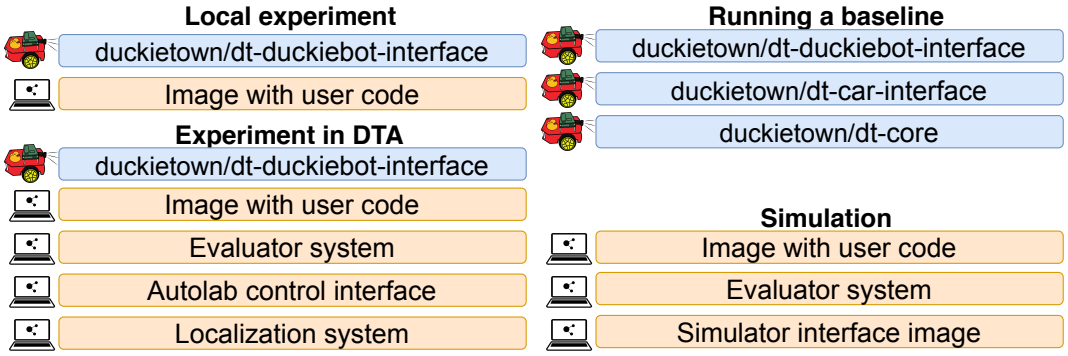


Fig. 5.4. Use-cases: The well-defined interfaces enable execution of an agent in different settings, including locally in hardware, in a DTA, with a standard baseline, or in a simulator. Different icons indicate where a container is executed (either on the robot or on the laptop).

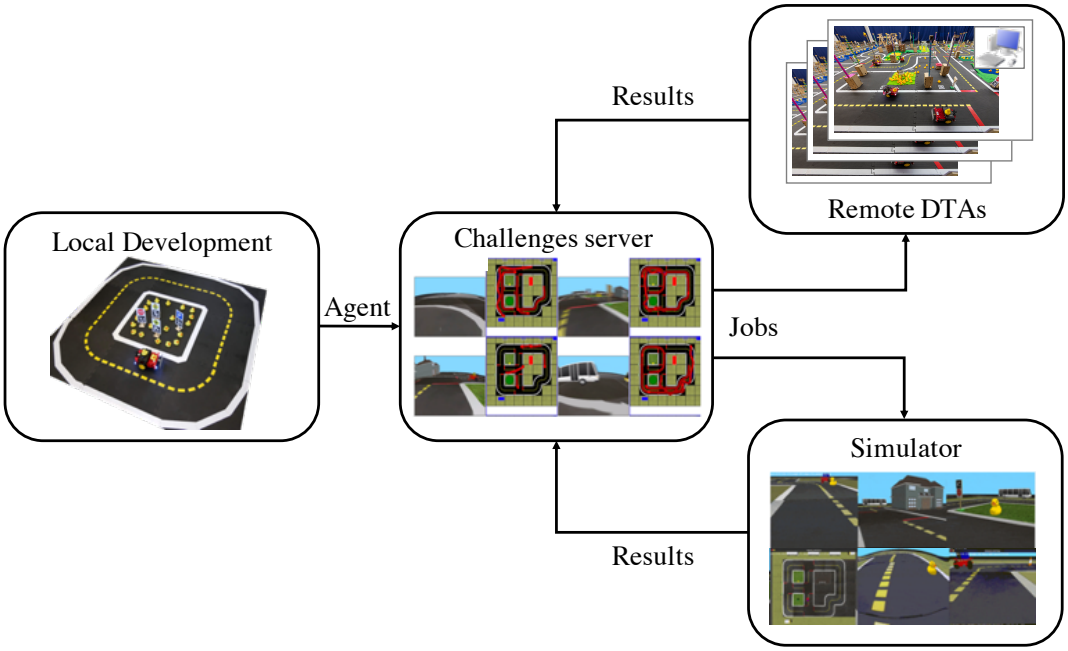


Fig. 5.5. The user workflow: Local development can take place on the real hardware or in a simulator. Agents are submitted to the Challenges server, which marshals agents to be submitted in simulation challenges to the cloud simulation, or agents that should be evaluated on real hardware to a DTA. The results of these evaluations are combined with the benchmark definition and made available to the submitter.

5.3.1. The Base Platform

The DUCKIENet is built using the Duckietown platform, which was originally developed in 2016 and has since been used for education [**86**], research [**102**] and outreach [**23**].

5.3.1.1. The Duckietown Hardware Platform. The Duckietown hardware consists of Duckiebots and Duckietown urban environments. Duckiebots are differential-drive mobile

robots with a front-facing camera as the only sensor, five LEDs used for communication, two DC-motors for actuation, and a Raspberry Pi for computation. Duckietowns are structured modular urban environments with floor and signal layers. The floor layer includes five types of road segments: straight, curve, three-way intersection, four-way intersection, and empty. The signal layer includes traffic signs and traffic lights. Traffic lights have the same hardware as the Duckiebots, excluding the wheels, and are capable of sensing, computing, and actuation through their LEDs.

5.3.1.2. The Duckietown Software Architecture. We implement the Duckietown base software as ROS nodes and use the ROS topic messaging system for inter-process communication. The nodes are organized in Docker images that can be composed to satisfy various use-cases (Fig. 5.4). Specifically, the components that correspond to the "robot" are the actuation and the sensing, and all other nodes correspond to the agent. The Duckietown simulator [**17**] is a lightweight, modular, customizable virtual environment for rapid debugging and training of agents. It can replace the physical robot by subscribing to commands and producing sensor observations. We provide an agent interface to the OpenAI Gym [**13**] protocol that can be called at a chosen frequency to speed up training. In the case of reinforcement learning, rewards can be specified and are returned, enabling one to easily set up episodes for training.

The process of marshalling containers requires some specific configurations. To ease this process, we have developed the Duckietown Shell which is a wrapper around the Docker commands and provides an interface to the most frequent development and evaluation operations [**24**]. Additionally, the Duckietown Dashboard provides an intuitive high-level web-based graphical user interface [**22**]. The result is that agents that are developed in one modality (e.g., the simulator) can be evaluated in the cloud, on real robot hardware, or in the DTA with one simple command or the at click of a button.

5.3.2. System Architecture Overview

The DUCKIENet (Fig. 5.5) is comprised of: (i) a Challenges server that stores agents, benchmarks, and results, computes leaderboards, and dispatches jobs to be executed by a distributed set of evaluators; (ii) a set of local or cloud-based evaluation machines that run the simulations; and (iii) one or more DTAs, physical instrumented labs that carry out physical experiments (Sec. 5.3.3).

The system has:

(1) **Super-users** who define the benchmarks. The benchmarks are defined as Docker containers submitted to the Challenges server. These users can also set permissions for access to different benchmarks, their results, and the scores.

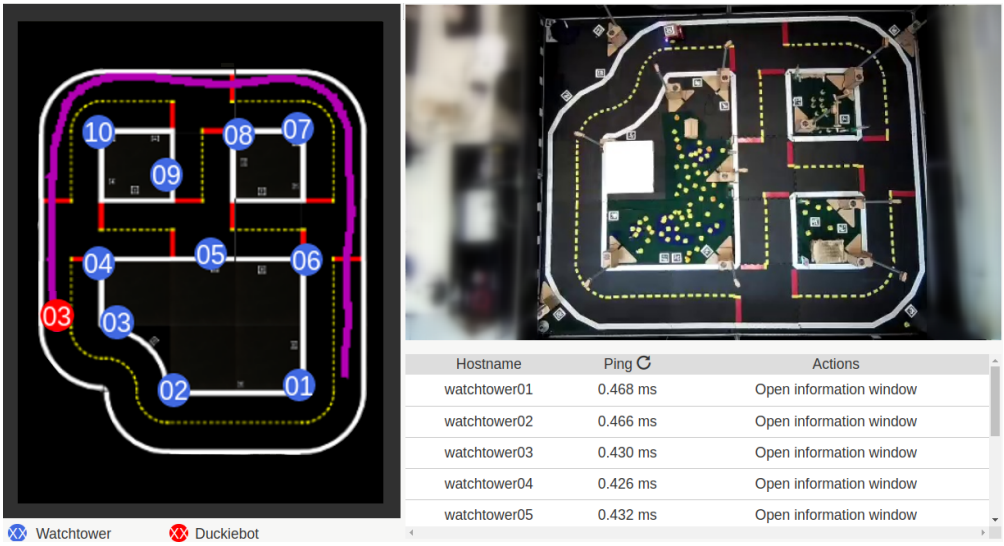


Fig. 5.6. The DTA GUI provides a control interface to human operators, including a representation of the map and the Watchtowers (left) and an overhead live feed of the physical Duckietown with diagnostic information about the experiment in progress (right).

(2) **Regular developers** are users of the system. They develop agents locally and submit their agent code as Docker containers. They can observe the results of the evaluation, though the system supports notions of testing and validation, which hides some of the results.

(3) **Simulation evaluators** and **experimental evaluators** query the Challenges server for jobs to execute. They run jobs as they receive them and communicate the results back to the Challenges server.

5.3.3. The Duckietown Automated Laboratory (DTA)

The DTAs are remotely accessible hardware ecosystems designed to run reproducible experiments in a controlled environment with standardized hardware (Sec. 5.3.1.1). DTAs comprise: (a) a set of Duckietown environments (maps); (b) a localization system that estimates the pose of every Duckiebot in the map (Sec. 5.3.3.1); (c) a local experiment manager, which receives jobs to execute from the Challenges server and coordinates the local Duckiebots, performs diagnostics, and collects logs from the system; (d) a human operator responsible for tasks that are currently not automated (e.g., resetting experiments and "auto"-charging); and (e) a set of Duckiebots modified with a upwards-facing AprilTag [**62**] and charge collection apparatus, as shown in Figure 5.1. A graphical UI (Fig. 5.6) makes the handling of experiments more user-friendly (Sec. 5.3.3.2).

We evaluate the reproducibility of the DTAs themselves by evaluating agents across different labs at distributed locations worldwide, as described in Section 5.4.

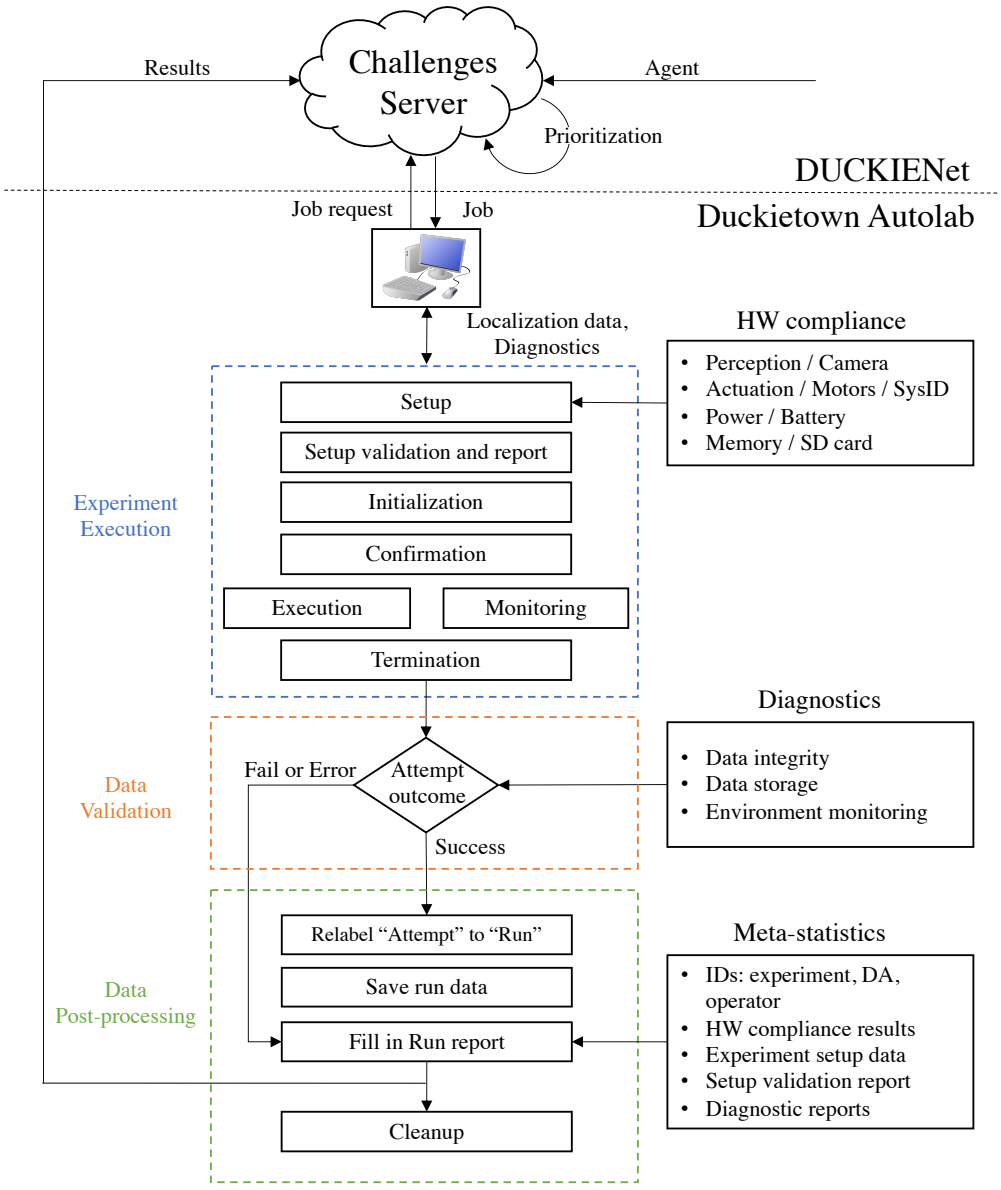


Fig. 5.7. Evaluation workflow: An experiment is triggered by an agent being submitted to the Challenges server. The experiment is run in the DTA following a predefined and deterministic procedure. The results are then uploaded to the Challenges server and returned to the submitter.

5.3.3.1. Localization. The goal of the localization system is to provide pose estimates of all Duckiebots during experiments. This data is post-processed according to metrics specified in the benchmarks to evaluate agent performance. The localization system comprises a network of Watchtowers, i.e., low-cost observation structures using the same sensing and computation as the Duckiebots. The Watchtowers are placed such that their fields-of-view, which are restricted to a local neighbor region of each Watchtower, overlap and collectively cover the entire road network. A set of calibration AprilTags are fixed to the road layer

of the map and provide reference points from the camera frame of each Watchtower to the map's fixed reference frame.

The localization system is decentralized and vision-based, using the camera both the streams of the network of Watchtowers and those of the Duckiebots themselves. In each image stream the system detects AprilTags, which are placed on the Duckiebots as well as in other locations around the city, e.g., on traffic signs at intersections. For each AprilTag, the system extracts their camera-relative pose. Using the resulting synchronized sequences of AprilTag-camera relative pose estimates, a pose graph-based estimate of the pose history of each AprilTag, using the g^{2}o library [**37**], is returned.

5.3.3.2. Operator Console. DTAs are equipped with a graphical user interface based on the \compose\ web framework (Fig. 5.6). Similarly to the Duckietown Dashboard, it is designed to provide an intuitive interface between the operator and the DTA. Available functionalities include: various diagnostics (e.g., network latency, room illumination, camera feeds, etc.), the ability to run jobs (experiments) with a few clicks and compute, visualize and return results to the Challenges server, and the control of the environment illumination.

5.3.4. Defining the Benchmarks

Benchmarks can be defined either to be evaluated in simulation or on real robot hardware. Both types require the definition of: (a) the environment, through a standardized map format; (b) the "non-playing characters" and their behavior; and (c) the evaluation metrics, which can be arbitrary functions defined on trajectories.

Each benchmark produces a set of scalar scores, where the designer can choose the final total order for scoring by describing a priority for the scores as well as tolerances (to define a lexicographic semiorder). The system allows one to define "benchmarks paths" as a directed acyclic graph of benchmarks with edges annotated with conditions for progressions. One simple use case is to prescribe that an agent be first evaluated in simulation before being evaluated on real robot hardware. It is possible to set thresholds on the various scores that need to be reached for progression. For competitions such as the AI Driving Olympics [**102**], the superusers can set up conditions such as "progress to the next stage only if the submission does better than a baseline".

5.3.5. DTA Operation Workflow

The Challenges server sends a job to a DTA that provides the instructions necessary to carry out the experiment (Fig. 5.7). The DTA executes each experiment in a series of three steps: (i) experiment execution, (ii) data validation, and (iii) data post-processing and report generation.

Before the DTA carries out an experiment, it first verifies that the robots pass a hardware compliance test that checks to see that their assembly, calibration, and the functionality of critical subsystems (sensing, actuation, power, and memory) are consistent with the specifications provided in the description of the experiment.

Having passed the compliance test, the experiment execution phase begins by initializing the pose of the robot(s) in the appropriate map, according to the experiment definition. The robot(s) that runs the user-defined agent is categorized as "active", while the remaining "passive" robots run baseline agents. During the initialization step, the appropriate agents are transferred to each robot and initialized (as Docker containers). The localization system assigns a unique ID to each robot and verifies that the initial pose is correct. Once the robots are ready and the DTA does not detect any abnormal conditions (e.g., related to room illumination and network performance), the DTA sends a confirmation signal to synchronize the start of the experiment. The robots then execute their respective agents until a termination condition is met. Typical termination conditions include a robot exiting the road, crashing (i.e., not moving for longer than a threshold time), or reaching a time limit.

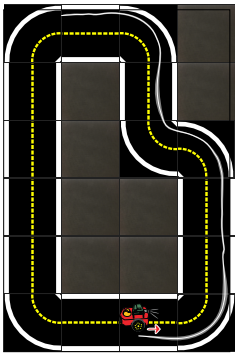
During the validation phase, the DTA evaluates the integrity of the data, that it has been stored successfully, and that the experiment was not been terminated prematurely due to an invalidating condition (e.g., an unexpected change in environment lighting). Experiments that pass this phase are labeled as successful runs.

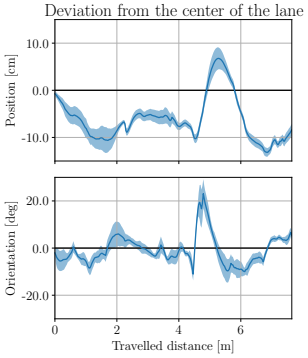
In the final phase, the DTA compiles experiment data and passes it to the Challenges server. The Challenges server processes the data according to the specifications of the experiment and generates a report that summarizes the run, provides metrics and statistics unique to the run, as well as information that uniquely identifies the experiment, physical location, human operator, and hardware compliance. The entire report including the underlying data as well as optional comparative analysis with selected baselines are then shared with the user who submitted the agent. The Challenges server then performs cleanup in anticipation of the next job.

5.4. Validation

We perform experiments to demonstrate (a) the repeatability of performance within a DTA, (b) the reproducibility of experiments across different robots in the same DTA, and (c) reproducibility across different facilities.

To assess the repeatability of an experiment, we run the same agent, in this case a lane-following (LF) ROS-based baseline, on the same Duckiebot and in the same DTA. To evaluate inter-robot reproducibility, we run the same experiment on multiple Duckiebots, keeping everything else fixed. Finally, we demonstrate that the same experiments yield similar results when run in different DTAs across the world.





Robots	Location	Repetitions	AVG MPD	STD MPD	AVG MOD	STD MOD
1	ETHZ	9	-6.2	1.3	-0.8	2.8

Fig. 5.8. Repeatability: Experiments repeatedly run on the same robot. Plots include mean and standard deviation. The execution of the same agent is low variance across runs.

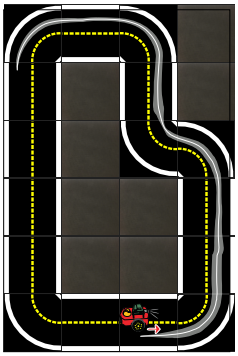
We define two metrics to measure repeatability: Mean Position Deviation (MPD) and Mean Orientation Deviation (MOD). Given a set of trajectories, MPD at a point along a trajectory is computed as the mean lateral displacement of the Duckiebot from the center of the lane. Similarly, MOD represents the mean orientation with respect to the lane orientation. For each set of experiments, we compute the average and the standard deviation along the trajectory itself.

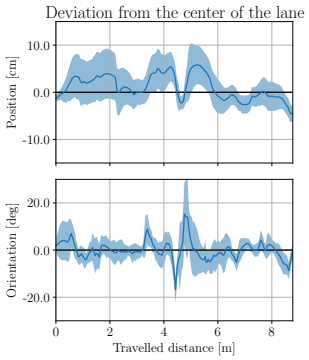
5.4.1. Experiment Repeatability

To demonstrate experiment repeatability, we run a baseline agent for the LF task nine times, on the same Duckiebot, with the same initial conditions and in the same DTA. The results are in Figure 5.8, with the mean and standard deviation of the robot's position on the map. We also include plots of the MPD and MOD metrics. Given the vetted robustness of the baseline agent, we expect repeatable behavior. This is supported by standard deviations of the MPD (1.3 cm) and the MOD (2.8 deg), which show that there is low variability in agent performance when run on the same hardware.

5.4.2. Inter-Robot Reproducibility

Given the low-cost hardware setup, we expect a higher degree of variability if the same agent is run on different robots. To evaluate inter-robot reproducibility, we run the LF baseline three times each, on three different Duckiebots. Experiments are nominally identical, and performed in the same DTA. Although the behavior of each Duckiebot is expected to





Robots	Location	Repetitions	AVG MPD	STD MPD	AVG MOD	STD MOD
3	ETHZ	3×3	1.1	3.4	-0.4	5.2

Fig. 5.9. Inter-Robot Reproducibility: Experiments on three different but similar robots. Results show that there is more variation when the same agent is run on different hardware.

be repeatable given the result of the previous section, we expect some shift in the performance distribution to hardware nuisances such as slight variations in assembly, calibration, manufacturing differences of components, etc.

Figure 5.9 visualizes mean and standard deviation of the trajectories for all runs. The experiments reveal a standard deviation for the MPD of 3.4 cm and a standard deviation for the MOD of 5.2 deg. These results show higher deviations than for the single Duckiebot repeatability test, as expected.

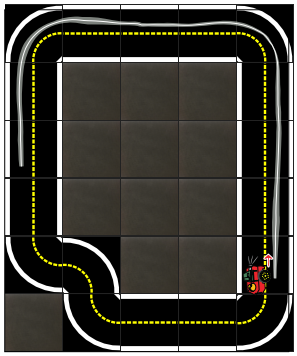
5.4.3. DTA Reproducibility

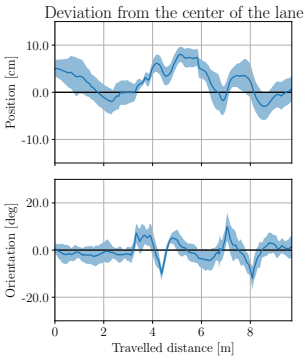
To demonstrate DTA reproducibility, we run a total of twelve experiments in two different DTAs with nominally identical conditions except for the hardware, the geographic location, the infrastructure, and the human operators. Results are shown in Figure 5.10. We obtain a standard deviation for the MPD of 2.5 cm, and a standard deviation for the MOD of 3.9 deg. Although variance is higher than the single Duckiebot repeatability test, which is to be expected, it is lower than that of the experiments run on three different robots, reinforcing the notion that hardware variability is measurable across different DTAs on the DUCKIENet.

5.4.4. Limitations

Finally, we discuss some limitations of the DUCKIENet framework, all of which are on our development roadmap.

The scenarios used to evaluate the reproducibility of the platform are relatively simple. With the Duckietown setup, we are able to produce much more complex behaviors





Robots	Location	Repetitions	AVG MPD	STD MPD	AVG MOD	STD MOD
1×2	ETHZ/TTIC	6×2	2.2	2.5	-0.4	3.9

Fig. 5.10. DTA Reproducibility Experiments in two different DTAs: ETH Zürich and TTI-Chicago. Results show that the infrastructure is reproducible across setups, since experiments in two different DTAs yield similar results.

and scenarios, such as multi-robot coordination, vehicle detection and avoidance, sign and signal detection, etc. These more complex behaviors should also be benchmarked. We have also only considered here metrics that are evaluated over a single agent, but multi-agent evaluation is also needed.

Finally, we have not analyzed the robustness to operator error (e.g., mis-configuration of the map compared to the simulation) or in case of hardware error (e.g., one camera in the localization system becomes faulty). This is necessary to encourage widespread adoption of the platform, which requires the components to be well-specified and capable of self-diagnosing configuration and hardware errors.

5.5. Conclusions

We have presented a framework for integrated robotics system design, development, and benchmarking. We subsequently presented a realization of this framework in the form the DUCKIENet, which comprises simulation, real robot hardware, flexible benchmark definitions, and remote evaluation. These components are swappable because they are designed to adhere to well-specified interfaces that define the abstractions. In order to achieve this level of reproducibility, we have relied heavily on concepts and technologies including formal specifications and software containerization. The combination of these tools with the proper abstractions and interfaces yields a procedure that can be followed by other roboticists in a straightforward manner.

Our contention is that there is a need for stronger efforts towards reproducible research for robotics, and that to achieve this we need to consider the evaluation in equal terms as the

algorithms themselves. In this fashion, we can obtain reproducibility *by design* through the research and development processes. Achieving this on a large-scale will contribute to a more systemic evaluation of robotics research and, in turn, increase the progress of development.

Chapter 6

Conclusion

This thesis has addressed core issues in the method that robotic research is conducted. We have enabled researchers with a new design for robotics environments to promote ease of reproducibility. In this effort, we hope that claims made for state-of-the-art techniques in autonomous driving are going to be easily validated and benchmarked. The autolabs are made available to the public as one such platform, offering low-cost robots and official, standardized evaluation of common autonomous driving tasks. We demonstrate that the variability between various autolabs is low and as such, the experiments can be reproduced faithfully at any remote autolabs.

We use that new platform to conduct our research and analysis about the reality gap, and more generally the usefulness of surrogate environments as proxies for an embodied challenge. We claim that a proxy environment is never going to be a perfectly faithful representation of the real environment, but also that it does not need to. We argue that even low-fidelity environments can be useful, as long as some relevant components are modeled properly and the task is imposed. Following this claim, we provide a novel metric, the PRPV, which allows us to quantify the usefulness of a proxy when predicting the relative ordering of a set of agents. Furthermore, we offer the LPV, a second metric that is used to quantify the value of a proxy as a training tool. We demonstrate that both these new metrics can be used to optimize a proxy environment to better predict the relative performance of agents, or reduce the number of real robot trials required to train an agent.

Limitations and Future Work

We think that the goal of removing the reality gap entirely is not realistic, but there is still work to be done to reduce it. A lot of work in robotics has neglected the gap or provided methods to mitigate it, but we hope that our analysis will encourage researchers to better understand the causes and effect of that discrepancy. While we offer tools to assess the value and optimize a proxy, precisely predicting the effect of the transfer is not yet possible.

In our work, we deliberately neglected the overhead of generating data. The proxy that we were using had such low overhead compared to the target environment that generating a dataset was never a problem. We are aware that some proxy environments (i.e. high-fidelity simulators) might require significant time investments to produce data. It can also be the case that a proxy is a trade-off between hardware cost and time, such that the proxy is safe but more time-consuming than target environment rollouts. In that case, it might be useful to take the resource cost into consideration while analyzing the proxy value metrics.

Moreover, our metric provide a total order on proxy instances. In the case that we want to account for more than one proxy value metric, resource costs, or different types of prediction errors, it might be beneficial to establish a partial order on proxies.

We provided a way to optimize proxy parameters within a given family. In the case that the environment gap irreducible within the given family (in which case it would be caused by non-tunable features), we do not provide any advice on how to improve the proxy. Figuring out a proxy family optimization method would be a good path for future work.

The experiments were done exclusively on the Duckietown platform. It would be relevant to conduct similar experiments with other platforms such as CARLA [**31**], and other embodied robotic tasks such as grasping.

References

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas *et al.* : Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] Francesco Amigoni, Emanuele Bastianelli, Jakob Berghofer, Andrea Bonarini, Giulio Fontana, Nico Hochgeschwender, Luca Iocchi, Gerhard Kraetzschmar, Pedro Lima, Matteo Matteucci *et al.* : Competitions for benchmarking: Task and functionality scoring complete performance assessment. *IEEE Robotics & Automation Magazine*, 22(3):53–61, 2015.

[3] J Andrew Bagnell : An invitation to imitation. Rapport technique, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 2015.

[4] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew et Igor Mordatch : Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[5] Herbert Bay, Tinne Tuytelaars et Luc Van Gool : Surf: Speeded up robust features. *In European conference on computer vision*, pages 404–417. Springer, 2006.

[6] H. Bharadhwaj, Z. Wang, Y. Bengio et L. Paull : A data-efficient framework for training and sim-to-real transfer of navigation policies. *In 2019 International Conference on Robotics and Automation (ICRA)*, pages 782–788, 2019.

[7] John D Blischak, Emily R Davenport et Greg Wilson : A quick introduction to version control with Git and GitHub. *PLoS Computational Biology*, 12(1), 2016.

[8] Carl Boettiger : An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[9] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang *et al.* : End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[10] Carsten Bormann : CBOR. `http://cbor.io/`. Accessed: 2020-02-29.

[11] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige *et al.* : Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *In 2018 IEEE international conference on robotics and automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[12] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan et Dilip Krishnan : Unsupervised pixel-level domain adaptation with generative adversarial networks. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang et Wojciech Zaremba : OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[14] Martin Buehler, Karl Iagnemma et Sanjiv Singh : *The DARPA urban challenge: Autonomous vehicles in city traffic*, volume 56. Springer, 2009.

[15] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan et Oscar Beijbom : NuScenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[16] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff et Dieter Fox : Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *In 2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[17] Maxime Chevalier-Boisvert, Florian Golemo, Yanjun Cao, Bhairav Mehta et Liam Paull : Duckietown environments for openai gym. `https://github.com/duckietown/gym-duckietown`, 2018.

[18] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel et Wojciech Zaremba : Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

[19] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun et Alexey Dosovitskiy : End-to-end driving via conditional imitation learning. *In 2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4693–4700. IEEE, 2018.

[20] Jack Collins, Ross Brown, Jurgen Leitner et David Howard : Traversing the reality gap via simulator tuning. *arXiv preprint arXiv:2003.01369*, 2020.

[21] Jack Collins, David Howard et Jurgen Leitner : Quantifying the reality gap in robotic manipulation tasks. *In 2019 International Conference on Robotics and Automation (ICRA)*, pages 6706–6712. IEEE, 2019.

[22] The Duckietown Community : The Duckietown dashboard. `https://docs.duckietown.org/DT19/opmanual_duckiebot/out/duckiebot_dashboard_setup.html`. Accessed: 2020-03-01.

[23] The Duckietown Community : Duckietown project. `https://www.duckietown.org/`. Accessed: 2020-01-22.

[24] The Duckietown Community : The Duckietown shell. `https://github.com/duckietown/duckietown-shell`. Accessed: 2020-03-01.

[25] Wade D. Cook, Moshe Kress et Lawrence M. Seiford : An axiomatic approach to distance on partial orderings. *RAIRO - Operations Research - Recherche Opérationnelle*, 20(2):115–122, 1986.

[26] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth et Bernt Schiele : The cityscapes dataset for semantic urban scene understanding. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[27] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth et Bernt Schiele : The Cityscapes dataset for semantic urban scene understanding. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[28] Nikolaus Correll, Kostas E Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M Romano et Peter R Wurman : Analysis and observations from the first Amazon Picking Challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188, 2016.

[29] Mark Cutler, Thomas J Walsh et Jonathan P How : Reinforcement learning with multi-fidelity simulators. *In 2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3888–3895. IEEE, 2014.

[30] Inc. Docker : Docker: Empowering app development for developers.

[31] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez et Vladlen Koltun : CARLA: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[32] Andreas Geiger, Philip Lenz, Christoph Stiller et Raquel Urtasun : Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[33] Andreas Geiger, Philip Lenz et Raquel Urtasun : Are we ready for autonomous driving? the kitti vision benchmark suite. *In Computer Vision and Pattern Recognition, 2012 IEEE Conference on*, pages 3354–3361, 2012.

[34] Florian Golemo, Adrien Ali Taiga, Aaron Courville et Pierre-Yves Oudeyer : Sim-to-real transfer with neural-augmented robot simulation. *In Conference on Robot Learning*, pages 817–828. PMLR, 2018.

[35] Ian Goodfellow, Yoshua Bengio, Aaron Courville et Yoshua Bengio : *Deep learning*, volume 1. MIT press Cambridge, 2016.

[36] Steven N. Goodman, Daniele Fanelli et John P. A. Ioannidis : What does research reproducibility mean? *Science Translational Medicine*, 8(341), 2016.

[37] Giorgio Grisetti, Rainer Kümmerle, Hauke Strasdat et Kurt Konolige : g^2o: A general framework for (hyper) graph optimization. *In IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, pages 9–13, 2011.

[38] Josiah Hanna et Peter Stone : Grounded action transformation for robot learning in simulation. *In Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[39] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup et David Meger : Deep reinforcement learning that matters. *In Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 3207–3214, 2018.

[40] Feng-Hsiung Hsu : *Behind Deep Blue: Building the computer that defeated the world chess champion.* Princeton University Press, 2002.

[41] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley et Frédo Durand : Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[42] Albert S. Huang, Edwin Olson et David Moore : LCM: Lightweight communications and marshalling. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4057–4062, Taipei, Taiwan, October 2010.

[43] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell et Konstantinos Bousmalis : 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*, pages 12627–12637, 2019.

[44] Matthew Johnson, Brandon Shrewsbury, Sylvain Bertrand, Tingfan Wu, Daniel Duran, Marshall Floyd, Peter Abeles, Douglas Stephen, Nathan Mertins, Alex Lesman *et al.* : Team IHMC's lessons learned from the DARPA robotics challenge trials. *Journal of Field Robotics*, 32(2):192–208, 2015.

[45] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova et Dhruv Batra : Sim2real predictivity: Does evaluation in

simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5(4):6670–6677, 2020.

[46] Sylvain Koos, Jean-Baptiste Mouret et Stéphane Doncieux : The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145, 2012.

[47] Srivatsan Krishnan, Behzad Borojerdian, William Fu, Aleksandra Faust et Vijay Janapa Reddi : Air learning: An AI research platform for algorithm-hardware benchmarking of autonomous aerial robots. *arXiv preprint arXiv:1906.00421*, 2019.

[48] Joel Lehman, Jeff Clune, Dusan Misevic, Christoph Adami, Lee Altenberg, Julie Beaulieu, Peter J Bentley, Samuel Bernard, Guillaume Beslon, David M Bryson *et al.* : The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *Artificial Life*, 26(2):274–306, 2020.

[49] Antoine Ligot et Mauro Birattari : On mimicking the effects of the reality gap with simulation-only experiments. *In International Conference on Swarm Intelligence*, pages 109–122. Springer, 2018.

[50] Antoine Ligot et Mauro Birattari : *On Mimicking the Effects of the Reality Gap with Simulation-Only Experiments: 11th International Conference, ANTS 2018, Rome, Italy, October 29–31, 2018, Proceedings*, pages 109–122. 01 2018.

[51] Regina Y Liu *et al.* : Bootstrap procedures under some non-iid models. *The annals of statistics*, 16(4):1696–1708, 1988.

[52] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun et Davide Scaramuzza : Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019.

[53] David G Lowe : Object recognition from local scale-invariant features. *In Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[54] Will Maddern, Geoffrey Pascoe, Chris Linegar et Paul Newman : 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[55] Maja J Matarić : Reinforcement learning in the multi-robot domain. *In Robot colonies*, pages 73–83. Springer, 1997.

[56] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal et Liam Paull : Active domain randomization. *In Proceedings of the Conference on Robot Learning*, pages 1162–1176, 2020.

[57] Pierre Merriaux, Yohan Dupuis, Rémi Boutteau, Pascal Vasseur et Xavier Savatier : A study of Vicon system positioning performance. *Sensors*, 17(7), 2017.

[58] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski *et al.* : Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[59] Jean-Baptiste Mouret et Konstantinos Chatzilygeroudis : 20 years of reality gap: a few thoughts about simulators in evolutionary robotics. *In Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1121–1124, 2017.

[60] Jean-Baptiste Mouret, Sylvain Koos et Stéphane Doncieux : Crossing the reality gap: a short introduction to the transferability approach. *arXiv preprint arXiv:1307.1870*, 2013.

[61] J. Krishna Murthy, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, Liam Paull, Florian Shkurti, Derek Nowrouzezahrai et Sanja Fidler : gradsim: Differentiable simulation for

system identification and visuomotor control. *In International Conference on Learning Representations*, 2021.

[62] Edwin Olson : AprilTag: A robust and flexible visual fiducial system. *In IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407, Shanghai, China, 2011.

[63] Alexander Pashevich, Robin Strudel, Igor Kalevatykh, Ivan Laptev et Cordelia Schmid : Learning to augment synthetic images for sim2real policy transfer. *arXiv preprint arXiv:1903.07740*, 2019.

[64] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang *et al.* : Duckietown: an open, inexpensive and flexible platform for autonomy education and research. *In 2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1497–1504. IEEE, 2017.

[65] C Pepper, S Balakirsky et C Scrapper : Robot simulation physics validation. *In Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*, pages 97–104, 2007.

[66] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron et Magnus Egerstedt : The Robotarium: A remotely accessible swarm robotics research testbed. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1699–1706, 2017.

[67] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron et Magnus Egerstedt : The Robotarium: A remotely accessible swarm robotics research testbed. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1699–1706, 2017.

[68] Chude (Frank) Qian : Teaching cars to drive themselves. `http://www.igvc.org/design/2019/1.pdf`. Accessed: 2021-02-14.

[69] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler et Andrew Y. Ng : ROS: An open-source Robot Operating System. *In International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*, 2009.

[70] Stéphane Ross, Geoffrey Gordon et Drew Bagnell : A reduction of imitation learning and structured prediction to no-regret online learning. *In Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[71] Gavin A Rummery et Mahesan Niranjan : *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[72] Fereshteh Sadeghi et Sergey Levine : Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[73] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller et Wojciech Samek : Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 2019.

[74] Shital Shah, Debadeepta Dey, Chris Lovett et Ashish Kapoor : AirSim: High-fidelity visual and physical simulation for autonomous vehicles. *In Proceedings of the International Conference on Field and Service Robotics (FSR)*, pages 621–635, 2018.

[75] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang et Russell Webb : Learning from simulated and unsupervised images through adversarial training. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116, 2017.

[76] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot *et al.* : Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[77] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra et Martin Riedmiller : Deterministic policy gradient algorithms. *In International conference on machine learning*, pages 387–395. PMLR, 2014.

[78] Jasper Snoek, Hugo Larochelle et Ryan P Adams : Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[79] Michael J Sorocky, Siqi Zhou et Angela P Schoellig : Experience selection using dynamics similarity for efficient multi-source transfer learning between robots. *arXiv preprint arXiv:2003.13150*, 2020.

[80] Ingo Steinwart et Andreas Christmann : Fast learning from non-iid observations. *Advances in neural information processing systems*, 22:1768–1776, 2009.

[81] Chen Sun, Abhinav Shrivastava, Saurabh Singh et Abhinav Gupta : Revisiting unreasonable effectiveness of data in deep learning era. *In Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[82] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine *et al.* : Scalability in perception for autonomous driving: Waymo open dataset. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2446–2454, 2020.

[83] Jie Tan, Zhaoming Xie, Byron Boots et C Karen Liu : Simulation-based design of dynamic controllers for humanoid balancing. *In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2729–2736. IEEE, 2016.

[84] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez et Vincent Vanhoucke : Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[85] Jacopo Tani, Andrea F Daniele, Gianmarco Bernasconi, Amaury Camus, Aleksandar Petrov, Anthony Courchesne, Bhairav Mehta, Rohit Suri, Tomasz Zaluska, Matthew R Walter *et al.* : Integrated benchmarking and design for reproducible and accessible evaluation of robotic agents. *arXiv preprint arXiv:2009.04362*, 2020.

[86] Jacopo Tani, Liam Paull, Maria T Zuber, Daniela Rus, Jonathan How, John Leonard et Andrea Censi : Duckietown: An innovative way to teach autonomy. *In International Conference EduRobotics 2016*, pages 104–121, 2016.

[87] Sebastian Thrun : Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

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

[89] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko et Trevor Darrell : Adapting deep visuomotor representations with weak pairwise constraints. *In Algorithmic Foundations of Robotics XII*, pages 688–703. Springer, 2020.

[90] Glenn Vinnicombe : Frequency domain uncertainty and the graph topology. *IEEE Transactions on Automatic Control*, 38(9):1371–1383, 1993.

[91] Christopher John Cornish Hellaby Watkins : Learning from delayed rewards. 1989.

[92] Jonathan Weisz, Yipeng Huang, Florian Lier, Simha Sethumadhavan et Peter Allen : Robobench: Towards sustainable robotics system benchmarking. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3383–3389, 2016.

[93] Greg Welch, Gary Bishop *et al.* : An introduction to the kalman filter. 1995.

[94] Ronald J Williams : Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[95] Sean Wilson, Paul Glotfelter, Li Wang, Siddharth Mayya, Gennaro Notomista, Mark Mote et Magnus Egerstedt : The Robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems. *IEEE Control Systems Magazine*, 40(1):26–44, 2020.

[96] Fei Xia, Chengshu Li, Kevin Chen, William B Shen, Roberto Martín-Martín, Noriaki Hirose, Amir R Zamir, Li Fei-Fei et Silvio Savarese : Gibson env v2: Embodied simulation environments for interactive navigation. *Stanford University, Tech. Rep.*, 2019.

[97] Fei Xia, William B Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín et Silvio Savarese : Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5(2):713–720, 2020.

[98] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel et Sergey Levine : One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018.

[99] Wenhao Yu, C Karen Liu et Greg Turk : Policy transfer with strategy optimization. *arXiv preprint arXiv:1810.05751*, 2018.

[100] Yangguang Yu, Zhihong Liu et Xiangke Wang : An introduction to formation control of UAV with Vicon system. *In Proceedings of the International Conference on Robotic Sensor Networks (ROSENET)*, pages 181–190, 2020.

[101] Fangyi Zhang, Jürgen Leitner, Zongyuan Ge, Michael Milford et Peter Corke : Adversarial discriminative sim-to-real transfer of visuo-motor policies. *The International Journal of Robotics Research*, 38(10-11):1229–1245, 2019.

[102] Julian Zilly, Jacopo Tani, Breandan Considine, Bhairav Mehta, Andrea F Daniele, Manfred Diaz, Gianmarco Bernasconi, Claudio Ruch, Jan Hakenberg, Florian Golemo *et al.* : The ai driving olympics at neurips 2018. *arXiv preprint arXiv:1903.02503*, 2019.