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

Jacopo Tani[1,*], Andrea F. Daniele[2], Gianmarco Bernasconi[1], Amaury Camus[1], Aleksandar Petrov[1], Anthony Courchesne[3], Bhairav Mehta[3], Rohit Suri[1], Tomasz Zaluska[1], Matthew R. Walter[2], Emilio Frazzoli[1,4], Liam Paull[3], Andrea Censi[1]

***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.**[†]

Fig. 1: **The Duckietown Autolab**: We augment Duckietowns [1] with localization and automatic charging infrastructure. We modify Duckiebots (inset) to facilitate detection by the localization system and auto-charging.

I. Introduction

Mobile robotics poses unique challenges that have precluded the establishment of benchmarks for rigorous evaluation. Robotic systems are *complex* with many different interacting 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 [2]). 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 [3]–[6]. 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 [7], [8] 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 [9]).

Simulation environments are potentially valuable tools for system-level evaluation. Examples such as CARLA [10], AirSim [11], and Air Learning [12] 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 [13]. For example, the DARPA urban challenge [14], the DARPA robotics challenge [15], and the Amazon Picking Challenge [16], 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 [17], [18] 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.

[1] ETH Zürich, Switzerland; [2] Toyota Technological Institute at Chicago, Chicago, USA; [3] Mila, Université de Montréal, Montréal, Canada; [4] nuTonomy, an Aptiv company, Boston, USA; * Corresponding author: tanij@ethz.ch.

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

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 [1] 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 II, describe an instantiation that adheres to these objectives, the DUCKIENet, in Section III, validate the performance and usefulness of the approach in Section IV and finally present conclusions in Section V.

II. 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. 2).

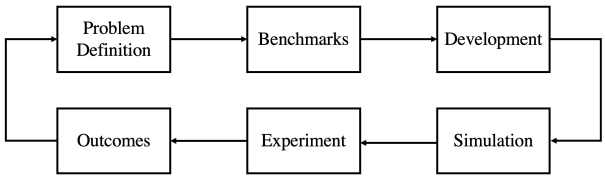


Fig. 2: **Integrating Benchmark Design**: Designing robotic benchmarks should be more tightly integrating into robotic system development.

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.

A. 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 [19]. 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.

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 [20]. 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 [21].

Containerization technologies such as Docker [22] 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 [23]. 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).

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 [2]. 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.

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 III, 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.

B. Agent Interoperability

Open source robotics middleware such as ROS [24] and LCM [25] 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) [26]. 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.

C. 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. 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 II-B 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 III, this approach is leveraged in the DUCKIENet to enable zero friction transitions from local development, to remote simulation, and to remote experiments in DTAs.

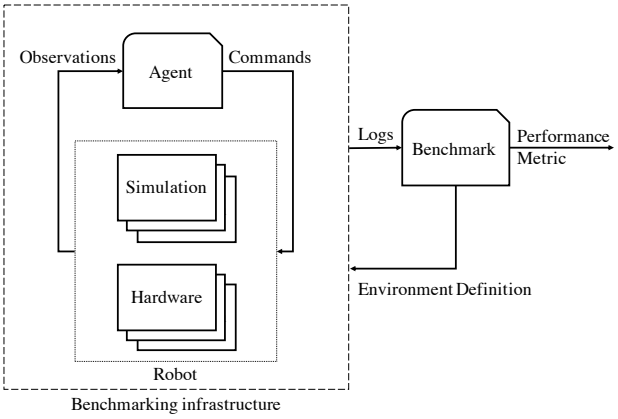


Fig. 3: **Abstractions and Interfaces**: An interface must be well-specified so that components on either side of the interface can be swapped in and out. The most important is the Agent-Robot interface, where the robot can be real hardware or a simulator. The interface between the benchmark and the infrastructure enables a clear definition of benchmark tasks (in terms of metrics) separately from the means by which they are evaluated.

D. 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. 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.

III. The DUCKIENet

In this section we describe an instantiation of the high-level objectives proposed in Section II 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.

A. The Base Platform

The DUCKIENet is built using the Duckietown platform, which was originally developed in 2016 and has since been used for education [27], research [28] and outreach [29].

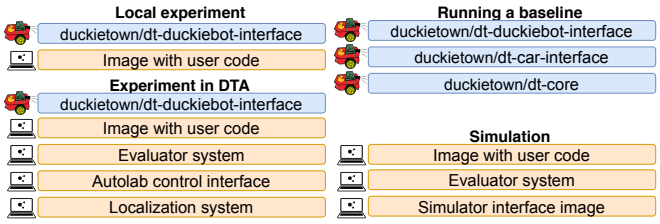


Fig. 4: **Use-cases**: The well-defined interfaces enable execution of an agent in different settings, including locally in hardware, in a DTA, with a standard baseline, or in a simulator. Different icons indicate where a container is executed (either on the robot or on the laptop).

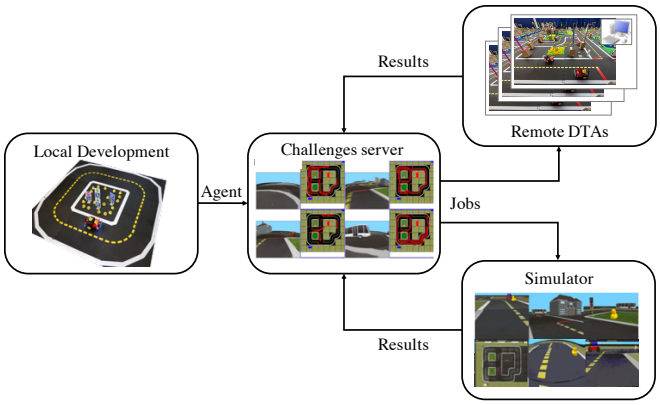


Fig. 5: **The user workflow**: Local development can take place on the real hardware or in a simulator. Agents are submitted to the Challenges server, which marshals agents to be submitted in simulation challenges to the cloud simulation, or agents that should be evaluated on real hardware to a DTA. The results of these evaluations are combined with the benchmark definition and made available to the submitter.

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.

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. 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 [30] 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 [31] 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

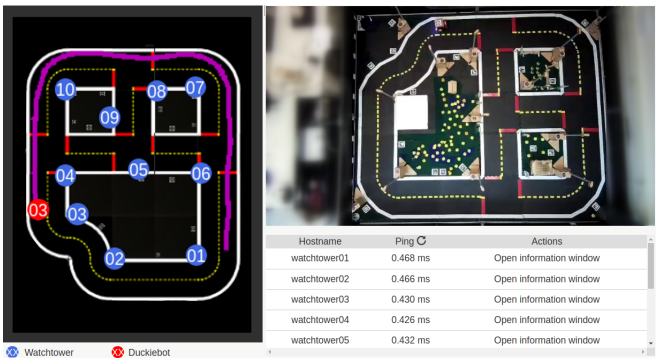


Fig. 6: **The DTA GUI** provides a control interface to human operators, including a representation of the map and the Watchtowers (left) and an overhead live feed of the physical Duckietown with diagnostic information about the experiment in progress (right).

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 [32]. Additionally, the Duckietown Dashboard provides an intuitive high-level web-based graphical user interface [33]. 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.

B. System Architecture Overview

The DUCKIENet (Fig. 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. III-C). 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.
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.

C. 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. III-A.1). DTAs comprise: (a) a set of Duckietown environments (maps); (b) a localization system that estimates the pose of every Duckiebot

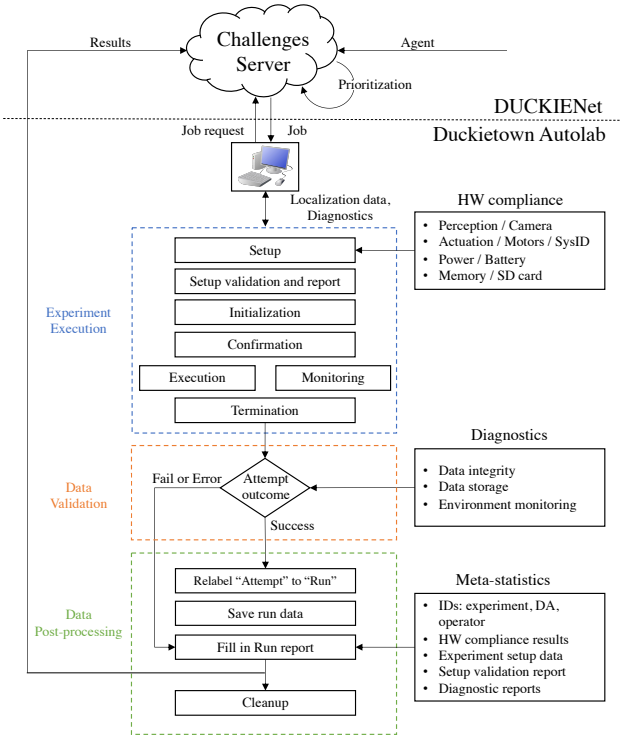


Fig. 7: **Evaluation workflow**: An experiment is triggered by an agent being submitted to the Challenges server. The experiment is run in the DTA following a predefined and deterministic procedure. The results are then uploaded to the Challenges server and returned to the submitter.

in the map (Sec. III-C.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 [34] and charge collection apparatus, as shown in Figure 1. A graphical UI (Fig. 6) makes the handling of experiments more user-friendly (Sec. III-C.2).

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

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^2o library [35], is returned.

2) Operator console: DTAs are equipped with a graphical user interface based on the \compose\ web framework (Fig. 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.

D. 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 [28], the superusers can set up conditions such as "progress to the next stage only if the submission does better than a baseline".

E. DTA Operation Workflow

The Challenges server sends a job to a DTA that provides the instructions necessary to carry out the experiment (Fig. 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.

F. The Artificial Intelligence Driving Olympics (AI-DO)

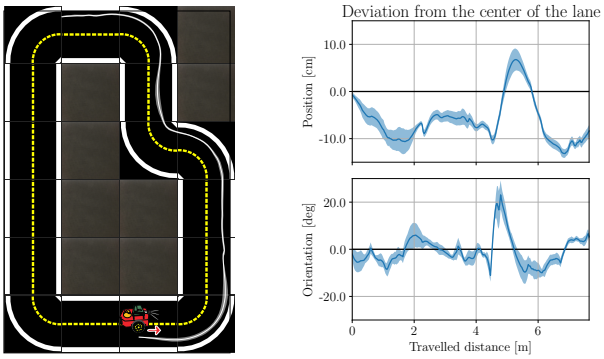
One of the key applications of our infrastructure is to host competitions. The AI Driving Olympics (AI-DO) [28] is a bi-annual competition that leverages the DUCKIENet to compare the ability of various agents, ranging from traditional estimation-planning-control architectures to end-to-end learning-based architectures, to perform autonomous driving tasks of increasing complexity. By fixing the environment and the hardware, agents and tasks can be modified to benchmark performance across different challenges. Most robotics challenges offer *either* embodied or simulated experiments. The DUCKIENet enables seamless evaluation in *both* along with a visualization of performance metrics, leaderboards, and access to the underlying raw data. The DUCKIENet makes the process of participating accessible, with open-source baselines and detailed documentation.

IV. 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.

We define two metrics to measure repeatability: Mean Position Deviation (MPD) and Mean Orientation Deviation



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

Fig. 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.

(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.

A. 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 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.

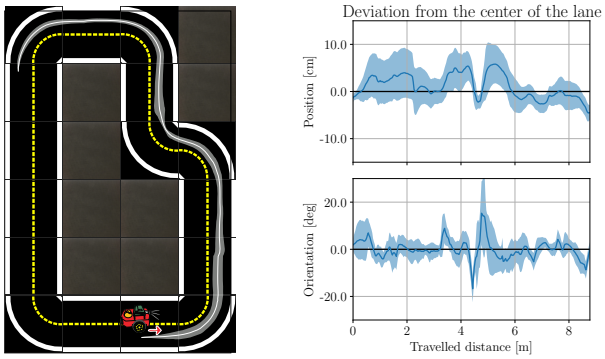
B. Inter-Robot Reproducibility

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 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 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.

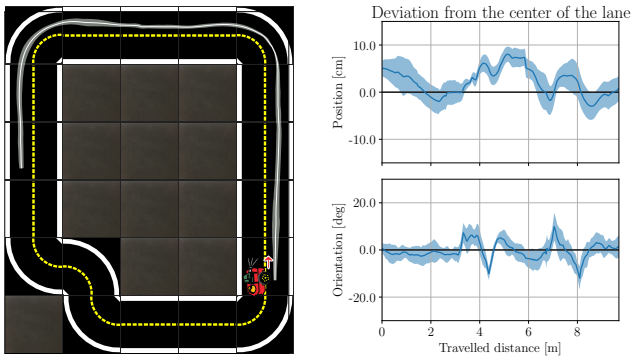
C. DTA Reproducibility

To demonstrate DTA reproducibility, we run a total of twelve experiments in two different DTAs with nominally



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

Fig. 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.



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. 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.

identical conditions except for the hardware, the geographic location, the infrastructure, and the human operators. Results are shown in Figure 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.

D. 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 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.

V. 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.

Acknowledgements

Liam Paull and Anthony Courchesne are supported by CIFAR (CCAI Chair Program), NSERC (Discovery Grant Program). Bhairav Mehta is additionally supported by Mila and IVADO Excellence Scholarship Fund. Andrea F. Daniele and Matthew R. Walter are supported in part by the National Science Foundation under grant IIS-1638072 and by the Robotics Consortium of the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program Cooperative Agreement W911NF-10-2-0016.

We also thank Josefine Quack, Eric Lu, Ben Weber, Florian Golemo, Chude Qian, all students who ran experiments on the system, as well as Marc-Andre Corzillius, Marcus Aaltonen, and everyone at ETH's Institute for Dynamic Systems and Control (IDSC) who supported us.

References

[1] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang *et al.*, "Duckietown: an open, inexpensive and flexible platform for autonomy education and research," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1497–1504.

[2] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2018, pp. 3207–3214.

[3] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3354–3361.

[4] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3213–3223.

[5] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "NuScenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[6] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. 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)*, 2020, pp. 2446–2454.

[7] Y. Yu, Z. Liu, and X. Wang, "An introduction to formation control of UAV with Vicon system," in *Proceedings of the International Conference on Robotic Sensor Networks (ROSENET)*, 2020, pp. 181–190.

[8] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The Robotarium: A remotely accessible swarm robotics research testbed," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1699–1706.

[9] P. Merriaux, Y. Dupuis, R. Boutteau, P. Vasseur, and X. Savatier, "A study of Vicon system positioning performance," *Sensors*, vol. 17, no. 7, 2017.

[10] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[11] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Proceedings of the International Conference on Field and Service Robotics (FSR)*, 2018, pp. 621–635.

[12] S. Krishnan, B. Borojerdian, W. Fu, A. Faust, and V. J. Reddi, "Air learning: An AI research platform for algorithm-hardware benchmarking of autonomous aerial robots," *arXiv preprint arXiv:1906.00421*, 2019.

[13] F. Amigoni, E. Bastianelli, J. Berghofer, A. Bonarini, G. Fontana, N. Hochgeschwender, L. Iocchi, G. Kraetzschmar, P. Lima, M. Matteucci *et al.*, "Competitions for benchmarking: Task and functionality scoring complete performance assessment," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 53–61, 2015.

[14] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: Autonomous vehicles in city traffic*. Springer, 2009, vol. 56.

[15] M. Johnson, B. Shrewsbury, S. Bertrand, T. Wu, D. Duran, M. Floyd, P. Abeles, D. Stephen, N. Mertins, A. Lesman *et al.*, "Team IHMC's lessons learned from the DARPA robotics challenge trials," *Journal of Field Robotics*, vol. 32, no. 2, pp. 192–208, 2015.

[16] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman, "Analysis and observations from the first Amazon Picking Challenge," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 1, pp. 172–188, 2016.

[17] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The Robotarium: A remotely accessible swarm robotics research testbed," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1699–1706.

[18] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, and M. Egerstedt, "The Robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems," *IEEE Control Systems Magazine*, vol. 40, no. 1, pp. 26–44, 2020.

[19] S. N. Goodman, D. Fanelli, and J. P. A. Ioannidis, "What does research reproducibility mean?" *Science Translational Medicine*, vol. 8, no. 341, pp. 341ps12–341ps12, 2016. [Online]. Available: https://stm.sciencemag.org/content/8/341/341ps12

[20] J. D. Blischak, E. R. Davenport, and G. Wilson, "A quick introduction to version control with Git and GitHub," *PLoS Computational Biology*, vol. 12, no. 1, 2016.

[21] C. Boettiger, "An introduction to Docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[23] J. Weisz, Y. Huang, F. Lier, S. Sethumadhavan, and P. Allen, "Robobench: Towards sustainable robotics system benchmarking," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3383–3389.

[24] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[25] A. S. Huang, E. Olson, and D. Moore, "LCM: Lightweight communications and marshalling," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010, pp. 4057–4062.

[26] "CBOR," http://cbor.io/, accessed: 2020-02-29.

[27] J. Tani, L. Paull, M. T. Zuber, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: An innovative way to teach autonomy," in *International Conference EduRobotics 2016*, 2016, pp. 104–121.

[28] J. Zilly, J. Tani, B. Considine, B. Mehta, A. F. Daniele, M. Diaz, G. Bernasconi, C. Ruch, J. Hakenberg, F. Golemo *et al.*, "The AI Driving Olympics at NeurIPS 2018," in *The NeurIPS'18 Competition*. Springer, 2020, pp. 37–68.

[29] "Duckietown project," https://www.duckietown.org/, accessed: 2020-01-22.

[30] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, and L. Paull, "Duckietown environments for OpenAI Gym," https://github.com/duckietown/gym-duckietown, 2018.

[31] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv preprint arXiv:1606.01540*, 2016.

[32] "The Duckietown shell," https://github.com/duckietown/duckietown-shell, accessed: 2020-03-01.

[33] "The Duckietown dashboard," https://docs.duckietown.org/DT19/opmanual_duckiebot/out/duckiebot_dashboard_setup.html, accessed: 2020-03-01.

[34] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 3400–3407.

[35] G. Grisetti, R. Kümmerle, H. Strasdat, and K. Konolige, "g^2o: A general framework for (hyper) graph optimization," in *IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, 2011, pp. 9–13.

[22] I. Docker. Docker: Empowering app development for developers. [Online]. Available: https://www.docker.com