Article

A Gamified Simulator and Physical Platform for Self-Driving Algorithm Training and Validation

Georgios Pappas ^{1,2,3,†}, Joshua E. Siegel ^{4,*}, Konstantinos Politopoulos ¹ and Yongbin Sun ⁵

¹ Department of Electrical and Computer Engineering, National Technical University of Athens, 15780 Athens, Greece; geopap1987@hotmail.com (G.P.); policent@central.ntua.gr (K.P.)

² Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI 48824, USA

³ Laboratory of Educational Material & Educational Methodology, Open University of Cyprus, Nicosia 2252, Cyprus

⁴ Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA

⁵ Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA; yb_sun@mit.edu

* Correspondence: jsiegel@msu.edu

† These authors contributed equally to this work.

Abstract: We identify the need for an easy-to-use self-driving simulator where game mechanics implicitly encourage high-quality data capture and an associated low-cost physical test platform. We design such a simulator incorporating environmental domain randomization to enhance data generalizability and a low-cost physical test platform running the Robotic Operating System. A toolchain comprising a gamified driving simulator and low-cost vehicle platform is novel and facilitates behavior cloning and domain adaptation without specialized knowledge, supporting crowdsourced data generation. This enables small organizations to develop certain robust and resilient self-driving systems. As proof-of-concept, the simulator is used to capture lane-following data from AI-driven and human-operated agents, with these data training line following Convolutional Neural Networks that transfer without domain adaptation to work on the physical platform.

Keywords: gamification; simulation; transfer learning; self-driving vehicles; autonomous systems; training systems; data collection and annotation

Citation: Pappas, G.; Siegel, J.E.; Politopoulos, K.; Sun, Y. A Gamified Simulator and Physical Platform for Self-Driving Algorithm Training and Validation. *Electronics* **2021**, *10*, 1112. <https://doi.org/10.3390/electronics10091112>

Academic Editor: Arturo de la Escalera Hueso

Received: 26 March 2021

Accepted: 4 May 2021

Published: 8 May 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Deep Learning requires Big Data to learn behaviors from infrequent edge cases or anomalies. One application making use of Big Data is vehicle automation, where information representing diverse scenarios is necessary to train algorithms resilient to infrequent and high-impact events.

While manufacturers log fleet data (e.g., Tesla capture customer telemetry [1,2]), it is difficult to validate that data are “clean” (e.g., common drunk, drowsy, drugged or distracted drivers might negatively impact a lane-holding algorithm). Even sober drivers with a poor understanding of the centerline of their vehicle may taint data. Further, the perception of “appropriate” morals and ethics is a driver of automated vehicle growth [3]. To meet the needs of robust, diverse and high quality data reflecting safe and ethical driving, the capture of known-good and large-scale data is necessary. “Wisdom of the crowd” requires massive scale, particularly in critical systems [4], so companies may instead capture data from costly trained drivers to maximize quality at the expense of quantity.

There is a need to capture large volumes of high-quality data with minimal supervision, and for inexpensive physical test platforms to validate real-world edge-case performance. This manuscript proposes gamified simulation as a means of collecting bulk data for self-driving and a platform based on commodity hardware for real-world algorithm validation. Such a simulator could capture human- and AI-enabled data to train vision-

based self-driving models and validate trained models' domain adaptation from simulator to real-world without explicit transfer learning.

Simulation is well-established [1], and small-scale hardware already tests algorithms in lieu of full-scale vehicles [5,6], particularly for algorithms that may be dangerous or costly to test on a full-scale physical platform, such as collision avoidance or high-speed and inclement weather operation. Our approach furthers proven techniques to allow the generation of data from *unskilled* drivers and through the validation of resulting algorithms on lower cost and more widely accessible hardware than is used today. This enables large scale, rapid data capture and real-world model validation that may then be translated to costlier and higher fidelity test platforms including full-scale vehicles.

We develop such a system and demonstrate its ability to generate data from untrained human drivers as well as AI-controlled agents. The simulated environment mimics a real-world laboratory, and game mechanics implicitly motivate quality data capture: players compete for high scores or the best time to collect line-following training data. The result is a human-in-the-loop simulation enabling inexpensive (<\$500 platform + <\$150 game asset costs), rapid (fewer administrative hurdles to clear for test approval), and high-quality (crowdsourced, configurable and supervised by game mechanics) data collection for certain use cases relative to conventional simulated and full-scale data capture environments and trained safety drivers. Our holistic solution integrates domain randomization, automated data generation, and thoughtful game mechanics to facilitate the bulk capture of useful data and transferrability to a near-disposable platform.

Our solution is unique in enforcing overt and latent rules in data collection: we learn from humans who have adapted to drive well in complex scenarios, while ensuring the data collected are of high quality by encouraging capture of useful data, automatically discarding low-quality data, disallowing specific actions, and enabling increased scale. Scoring mechanics encourage users to collect information for both common and edge-case scenarios, yielding valuable insight into common and low-frequency, high-risk events while the physical platform democratizes access to hardware-in-the-loop validation for budget-constrained developers.

This manuscript's primary contributions are the toolchain comprising the simulator and physical validation hardware. These support a data-generating framework capable of simulating scenarios infeasible to capture in reality, with the benefit of semi-supervised (enforced by game mechanics) and crowd-sourced (from untrained and unskilled users) data provided by humans testable on real images. Indeed, crowdsourcing has demonstrated success within transportation-for mode identification, parking location, and more [7]. The unique combination of implicitly supervisory game mechanics and crowdsourcing enables an end-to-end software and hardware training and testing platform unlike those used in contemporary research. Proof-of-concept Convolutional Neural Networks, while effective demonstrators of the simulator's ability to generate data suitable for domain adaptation to real hardware without domain adaptation, are peripheral contributions validating our claims of effective simulator and platform design.

2. Prior Art

2.1. Simulation and Gamification

One challenge in training AI systems is data availability, whether limited due to the scale of instrumented systems or the infrequency of low-likelihood events. To capture unpredictable driving events, Waymo, Tesla, and others collect data from costly highly-instrumented fleets [1,2,8]. To generate data at a lower cost, Waymo uses simulation to increase training data diversity [1] including not-yet-encountered scenarios. Simulation has been used to generate valuable data particularly for deep learning [9,10]. However, simulated data abide by implicit and explicit rules, meaning algorithms may learn latent features that do not accurately mirror reality and may miss "long tail" events [11]. Though Waymo's vehicles have reduced crash rates relative to those of human drivers [12], there is room for improvement, particularly in coping with chaotic real-world systems operated by

irrational agents. The use of thoughtfully-designed games lowers data capture cost relative to physical driving, with game mechanics, increased data volume, and scenario generation supporting the observation of infrequent events.

Researchers modified Grand Theft Auto V (GTA V) [13] to capture vehicle speed, steering angle, and synthetic camera data [14] and Franke used game data to train radio controlled vehicles to drive [15]. However, GTA V is inflexible, e.g., with respect to customizing vehicle or sensor, and data require manual capture and labeling.

Fridman's DeepTraffic supports self-driving algorithm development [16]. However, DeepTraffic generates 2D information, and while DeepTraffic crowdsources data, it does not crowdsource *human-operated* training data.

The open source VDrift [17] was used to create an optical flow dataset used to train a pairwise CRF model for image segmentation [18]. However, the tool was not designed with reconfigurability or data output in mind.

Another gamified simulation is SdSandbox and its derivatives [19–21] that generate steering/image pairs from virtual vehicles and environments. These tools generate unrealistic images, may or may not be human-in-the loop, and are not designed to crowdsource training data.

CARLA [22] is a cross-platform game-based simulator emulating self-driving vehicles and offering programmable traffic and pedestrian scenarios. While CARLA is a flexible research tool, users are unmotivated to collect targeted data, limiting the ability to trust data “cleanliness”. Further, there is no physical analog to the in-game vehicle. A solution with game mechanics motivating user performance and a low-cost physical test platform would add research value.

Other, task-specific elements of automated driving have been demonstrated in games and simulation, e.g., pedestrian detection [23,24] and stop sign detection [25]. Some simulators test transferrability of driving skills across varied virtual environments [26]. These approaches have not been integrated with physical testbeds, which could yield insight into real-world operations and prove simulated data may be used to effectively train self-driving algorithms.

There is an opportunity to create an easy-to-use gamified simulator and associated low-cost physical platform for collecting self-driving data and validating model performance. A purpose-built, human-in-the-loop, customizable simulator capable of generating training data for different environments and crowdsourcing multiple drivers' information would accelerate research, tapping into a large userbase capable of generating training data for both mundane and long-tail events.

Such a simulator could create a virtualized vehicle, synthetic sensor data, and present objectives encouraging “good behavior” or desirable (for training) “bad behavior”. For example, a user could gain points from staying within lane markers, or by collecting coins placed along a trajectory, or by completing laps as quickly as possible with collision penalties. Collected data would aid behavior-cloning, self-driving models in which input and output relationships are learning without explicit controller modeling. The simulator could be made variable through the use of randomized starting locations, dynamic lighting conditions, and noisy surface textures, compared with more deterministic traditional simulators.

Algorithms trained on the resulting synthetic data will be more likely to learn invariant features, rather than features latent to the simulator design. The result will be improved algorithms capable of responding well to infrequent but impactful edge cases missed by other tools, while a physical test platform will validate model performance in the real-world and capture data for transfer learning (if necessary).

2.2. Training Deep Networks with Synthetic Data

To prove the simulator's utility, we generate synthetic images and train deep learning behavior cloning models, then test those models on a low-cost hardware platform. This section explores training models using synthetic data and porting models to the real-world.

Neural network training is data-intensive and typically involves manually collecting and annotating input. This process is time consuming [27] and requires expert knowledge for some labels [28,29]. Generating high-quality, automatically labeled synthetic data can overcome these limitations. Techniques include Domain Randomization (DR) and Domain Adaptation (DA).

DR hypothesizes that a model trained on synthetic views augmented with random lighting conditions, backgrounds, and minor perturbations will generalize to real-world conditions. DR's potential has been demonstrated in image-based tasks, including object detection [30], segmentation [31] and 6D object pose estimation [32,33]. These methods render textured 3D models onto synthetic or real image backgrounds (e.g., MS COCO [34]) with varying brightness and noise. The domain gap between synthetic and realistic images is reduced by increasing the generalizability of the trained model (small perturbations increase the likelihood of the model converging on latent features).

Synthetic data also facilitates 3D vision tasks. For example, FlowNet3D [35] is trained on a synthetic dataset (FlyingThings3D [36]) to learn scene flow from point clouds, and generalizes to real LiDAR scans captured in the KITTI dataset [37]. Im2avatar [38] reconstructs voxelized 3D models from synthetic 2D views from ShapeNet [39], and the trained model produces convincing 3D models from realistic images of PASCAL3D+ dataset [40].

Domain Adaptation (DA) uses a model trained on one source data distribution and applies that model to a different, related target distribution—for example, applying a lane-keeping model trained on synthetic images to real-world images for the same problem type. In cases, models may be retrained on some data from the new domain using explicit transfer learning. In our case, we use DA to learn a model from a simulated distribution of driving data and adapt that model to a real-world context without explicit retraining.

For generating realistic data to support transfer learning without explicit capture of real-world data, Generative Adversarial Networks (GANs) [41] have been used to generate realistic data [42], 3D pose estimators [43] and grasping algorithms [44]. This work shows promising results, but the adapted images present unrealistic details and noise artifacts.

We aim to develop a simulator based on a game engine capable of generating meaningful data to inform Deep Learning self-driving behavior cloning models capable of real-world operation, with the benefit of being able to crowdsource human control and trusting the resulting input data as being “clean”. These models clone human behavior to visually-learn steering control based on optical environmental input such as road lines seen in RGB images. Such a system may learn the relationship between inputs, such as monocular RGB camera images, and outputs, such as steering angles. In this respect, both a vision model and an output process controller are implicitly learned.

3. Why Another Simulator?

While there are existing simulators [45], our Gamified Digital Simulator (GDS) has three key advantages.

GDS is designed for ease-of-use and self-supervision. A user can collect data (images and corresponding steering angles) without technical knowledge of the vehicle, simulator, or data capture needs. This expands GDS's potential userbase over traditional simulators.

GDS provides an in-game AI training solution that can, without track knowledge, collect data independently. This allows some data to be generated automatically and with near-perfect routing as described in Section 4.10.

GDS offers a resource-efficient simulation, adapting the graphical quality level of scene elements depending on the importance of each element in training a model. Elements such the track and its texture (Section 4.1) are high quality whereas the background (Section 4.3) uses low-polygon models and low-resolution textures. This reduces the processing power needed to run the GDS, expanding its applicability to constrained compute devices.

4. The Gamified Digital Simulator: Development Methodology

This section details GDS's development. In Figure 1, we overview the GDS toolchain. The virtual car can be driven both by a human (using a gamepad) or a rule-based AI. The simulator generates two types of data: (a) images from a virtual camera simulating the physical vehicle and (b) a CSV file with speed and steering angle details. To test that these virtual data are effective for training behavior cloning algorithms, we capture data and train models to operate a virtual vehicle agent in the GDS environment and test this model on the physical platform to determine whether the learned models are effective in a different (physical) environment.

Figure 1. The end-to-end toolchain couples a virtual environment with a real-world platform and track setup. Synthetic images, trained by in-game AI and/or human operators, inform self-driving models. These models use behavior cloning to mimic observed behaviors without explicit system modeling.

The GDS is built upon the cross-platform Unity3D Game Engine. The game consists of four scenes: (a) Main Menu, (b) Track Selection, (c) User Input Mode and (d) AI Mode. The Main Menu and Track Selection scene canvases and UI elements help the user transition between modes. From the Track Selection scene, a user selects one of two identical playable scenes (tracks/environments), one with a human-operated vehicle and the other with an AI-operated vehicle. The AI Scene utilizes an in-game AI or an external script, e.g., running TensorFlow [46] or Keras [47] AI models, as described in Sections 4.10 and 7.

We first explore the application design, and then describe the methodology for transforming the game into a tool for generating synthetic data for training a line following model. Gamification allows non-experts to provide high-volume semi-supervised training data. This approach is unique relative to conventional simulation in that it provides a means of crowdsourcing data from goal-driven humans, expediting behavior cloning from the “wisdom of the crowd”.

While designing the GDS, we considered it as both a research tool and a game with purpose [48,49] and therefore include elements to enhance the user experience (UX). The fun is “serious fun” (Figure 2), as defined by UX expert Nicole Lazzaro [50,51], where users play (or do boring tasks) to make a difference in the real world.

While the simulator cannot be released open-source due to license restrictions on some constituent assets, it is capturing data for academic studies related to training AVs and human-AV interaction and these data will be made available to researchers. The detail provided in this section should allow experienced game developers to recreate a similar tool without requiring extensive research. For researchers interested in implementing a similar software solution, the authors are happy to share source code with individuals or

groups have purchased the appropriate license to the required assets—please contact the authors for more information.

Figure 2. “The Four Keys to Fun” by Nicole Lazzaro (simplified). GDS belongs primarily to the “serious fun” category. Adapted from ref. [50].

4.1. Road Surface

A road surface GameObject simulates a test track (Figure 3). We developed modular track segments using Unity’s cuboid elements and ProBuilder (Figure 4). Modules were given a realistic texture (Figure 5) that changes based upon ambient lighting and camera angles, minimizing the likelihood that the neural network learns behaviors based on tessellated edges.

Figure 3. The initial test track was rectangular and designed as a simple test for the simulator and data collection system.

To reduce trained model overfit, we developed “data augmentation” using Unity’s *lerp* function to add Gaussian noise creating speckles appearing over time. A second script manipulates the in-game lighting sources randomly so that vehicles repeating trajectories capture varied training data. Both techniques support Domain Randomization and improve generalizability for domain adaptation (e.g., sim2real).

4.2. Game Mechanics

Gamification encourages players to abide by latent and overt rules to yield higher-quality, crowdsourcable data.

On the rectangular track, we created GameObject coins within the lane markers. Players were intrinsically motivated to collect coins for a chance to “win”. Figure 6, though this approach allowed non-sequential collection, with users exiting the track boundaries and reentering without penalty. While we could disregard non-consecutive pickups, failing to do so would contaminate data.

Figure 4. The track comprises adjacent modular components.

Figure 5. This asphalt texture was tessellated across the road surface in later versions of the simulator.

Figure 6. An early version of the GDS featured collectable coins worth points to incentive the driver to stay on the road.

We subsequently developed a system of colliders (green in Figure 7) invisible to the camera prevent buggy from traveling outside the white lane makers, ensuring that collected data are always within the lane, reducing contamination. Crash data and preceding/trailing frames were also eliminated from training data.

Figure 7. We developed an invisible collider system to implicitly force drivers to stay on the road surface.

4.3. Environment

The game has two purposes: to create a compelling user experience (UX) conducive to long play, and to create dynamic conditions to enhance domain randomization and prevent the neural network model fitting to environmental features.

The game environment is a series of GameObjects and rendering lighting parameters. Objects include rocky surfaces and canyon models (Figure 8a) scaled and arranged to create a rocky mountain (Figure 8b). We included assets from the Unity Asset Store to excite players and create dynamic background images. These include a lava stream particle effect (Figure 8c,d) and animated earthquake models (Figure 8e,f). Combined assets create the experience of driving near a volcano. The dynamic background provides “noise” in training images, ensuring the neural network fits to only the most-invariant features. Finally, a night Skybox is applied (Figure 8g). Game environment elements deemed non-critical to the simulation are low-polygon for efficiency.

Figure 8. These figures show the elements comprising the simulated environment and surroundings.

The perceived hostility of the environment implicitly conveys the game mechanics—the buggy must not exit the track boundaries, or it will fall to its doom. This mechanic allows users to pick up and play the game without instruction.

4.4. Representative Virtual Vehicle

The virtual vehicle is a multi-part 3D model purchased from Unity's Asset store and customized with CAD from the physical vehicle's camera mount.

To make the buggy drivable, we created independent wheel controllers (Figure 9) using a plugin that simulates vehicle physics. Using this model, we set parameters including steering angles, crossover speed, steering coefficient, Ackermann percentage, flipover behavior, forward and side slip thresholds, and speed limiters. These parameters allowed us to tailor the in-game model to mimic the physical vehicle.

Figure 9. Each wheel has its own controller running a high-fidelity physics model. This allowed us to control steering angle, friction coefficient, and more.

4.5. Simulated Cameras

The GDS uses a multi-camera system to simultaneously render the user view and export training data. The user camera provides a third person perspective, while a second camera placed at the front of the buggy simulates the RC vehicle's real camera in terms of location, resolution and field of view (FoV). This camera generates synthetic images used for training the neural network model and is calibrated in software. A third camera is placed above the vehicle, facing downwards. This camera provides an orthographic projection and, along with a RenderTexture, creates a mini-map.

Two additional cameras behave akin to SONAR or LiDAR and calculate distance to nearby objects using Unity's Raycasting functionality. These cameras support in-game AI used to independently generate synthetic training data (Section 4.10).

The location of the cameras is linked to the buggy's body. Each camera has differing abilities to "see" certain GameObjects as determined using Unity Layers. For example, the primary camera sees turn indicators directing the user, but the synthetic forward imaging camera ignores these signs when generating training data. In the coin demo, the user could see the coins, but these were invisible to the data-generating camera.

4.6. Exported Data

We attached a script to the buggy to generate sample data at regular intervals, including a timestamp, the buggy's speed, and the local rotation angle of the front wheels. This information is logged to a CSV in the same format used by the physical platform. Another

script captures images from the synthetic front-facing camera at each timestep to correlate the steering angle and velocity with a particular image. The CSV file and the images are stored within the runtime-accessible StreamingAssets folder.

4.7. Simulator Reconfigurability

In order to simulate the configuration of the physical vehicle, we needed to iterate tests to converge on parameters approximating the physical vehicle. To speed the process of tuning the simulator for other vehicles, options can be changed by editing an external file. Multi-display users can configure options on a second monitor (Figure 10). Configuration includes:

1. **Low speed steering angle**-the absolute value of the maximum angle at which the center of the steered wheels is maximum when the vehicle is in low-speed mode
2. **High speed steering angle**-same as above, but in high-speed mode
3. **Crossover speed**-the speed at which the vehicle changes from low- to high-speed mode
4. **Steer coefficient (front wheels)**-the steering multiplier between the steering angle and the actual wheel movement
5. **Steer coefficient (back wheels)**-same as above. Can be negative for high-speed lane changes (translation without rotation)
6. **Forward slip threshold**-slip limit for transition from static to sliding friction when accelerating/braking
7. **Side slip threshold**-same as above, but for steering
8. **Speed limiter**-maximum vehicle speed allowable (also reduces available power to accelerate)
9. **Vertical Field of View**-in degrees, to match physical vehicle camera
10. **Sampling Camera Width**-ratio of width to height of captured image
11. **Sampling Rate**-rate, in Hz, of capture of JPG images and logging to CSV file

Figure 10. When the simulator is connected to a multi-display host, the second display allows users to adjust options in real-time. This greatly speeds up tuning the buggy model to match the physical model, and helps make the simulator more extensible to other vehicle types.

We use another external file to set the starting coordinates of the buggy (Figure 11), helping test humans and AI alike under complicated scenarios (e.g., starting immediately in front of a right turn where the horizontal line is exactly perpendicular to the vehicle, or starting perpendicular to the lane markers).

4.8. 2D Canvas Elements

2D GameObjects show the buggy's realtime speed and wheel angle. There are also two RenderTexture elements, one displaying a preview of the synthetic image being captured (the "real cam" view) and one showing the track on a "minimap". Throttle and brake status

indicators turn green when a user presses the associated controls (Figure 12). This feature makes it easier for newcomers to quickly capture useful training data.

Figure 11. This figure shows the starting position options selectable from an external file. Each number refers to the center of the tile, while a starting angle can also be passed as a parameter to the game engine.

4.9. Unity C# and Python Bridge

We test models in the virtual environment before porting them to a physical vehicle. Deep learning frameworks commonly run in Python, whereas Unity supports C# and JavaScript.

We developed a bridge between Unity and Python by allowing GDS to read an external file containing commanded steering angle, commanded velocity, and AI mode (in-game AI—using the cameras described in Section 4.10, or external AI from Section 6, where the commanded steering and velocity values are read from the file at every loop).

This approach allows near-realtime control from an external model. Python scripts monitor the StreamingAssets folder for a new image, process this image to determine a steering angle and velocity, and update the file with these new values for controlling steering and velocity at 30+ Hz.

Figure 12. 2D Canvas Elements include speed and steering displays, a minimap, the forward camera view, and a minimap.

Using the PyGame library [52], we also “pass through” non-zero gamepad values, allowing for semi-supervised vehicle control (AI with human overrides). Upon releasing

the controller, the external AI resumes control. This allows us to test models in-game and helps us “unstick” vehicles to observe a model after encountering a complex scenario.

4.10. User Controlled Input and In-Game AI Modes

There are two game modes: User Control and AI Control.

In User Control, users interface with gamepads’ analog joysticks to control the vehicle.

Under AI Control, the buggy is controlled by in-game AI or an external model (Section 4.9).

In-game AI captures data without user input. This AI uses a three-camera system (the front camera [the same used to capture the synthetic view image] and two side cameras rotated $\pm 60^\circ$ relative to the y -axis) as input. Each camera calculates the distance between its position and the invisible track-border colliders.

The camera orientations and “invisible” distance measurements are represented in Figure 13 (green represents the longest clear distance, and red lines indicate more-obstructed pathways). The car moves in the direction of the longest free distance. If the longest distance comes from the front camera, the buggy moves straight. If it comes from a side cameras, then it centers itself in the available space. When the front distance falls under a threshold, the buggy brakes in advance of a turn.

As Unity cannot emulate joystick input, we use the bridge from Section 4.9 to both write and read output for controlling the buggy. The in-game AI model knows ground truth, such as the external positioning of the track’s invisible colliders, to effectively measure relative positioning. This allows the virtual buggy to drive predictably and create valuable samples without human involvement. This method generates trustable unsupervised training data for the deep learning network described in Section 6.

Figure 13. This figure shows the cameras measuring the distance to the environment via raycasting. The green line projected from the front camera indicates the longest unoccupied distance to an object, while the two red lines indicate the distance measures for each side camera. The in-game AI keeps these distances roughly equal to center the buggy in the lane.

5. Integrating GDS with an End-to-End Training Platform

The GDS is part of an end-to-end training system for self driving also comprising a physical platform (used for model validation) and a physical training environment (duplicated in the simulated world). Each element is detailed in the following subsections.

5.1. A Physical Self-Driving Test Platform

Self-driving models used to automate vehicle control are ultimately designed for use in physical vehicles. Full-size vehicles, however, are costly—with a self-driving test platform potentially costing in excess of \$250,000, and requiring paid skilled operators for data capture. We therefore created a lower-cost, easy-to-operate and smaller-scale physical vehicle platform and environment mirroring the GDS world to validate model performance.

We considered the TurtleBot [53] and DonkeyCar [54] platforms; both are low-cost systems. The TurtleBot natively supports the ROS middleware and Gazebo simulation tool, however the kinematics of the differential-drive TurtleBot do not mirror conventional cars. The DonkeyCar offers realistic Ackermann steering and a more powerful powertrain with higher top speed, better mirroring passenger vehicles. While the DonkeyCar platform is compelling, it suffers from some limitations: it is primarily a modularized compute module featuring one RGB camera, and this module may be installed onto a range of hardware that may introduce undesirable variability. Integrating the robotics, software, and mobility platform in a tightly-coupled package eliminates variability introduced by changing mobility platforms. The DonkeyCar also does not natively run ROS, limiting research extensibility.

We therefore developed a self-driving platform using hardware similar to the DonkeyCar, and a software framework similar to that of the TurtleBot—a 1/10th scale radio-controlled car chassis with Ackermann steering, running ROS. Our hardware platform is a HobbyKing Short Course Truck, which limits variability in experimental design and provides a platform suitable for carrying a larger and more complex payload, as well as capable of operating robustly in off-road environments. The large base, for example, has room for GPS receivers, has the ability to carry a larger battery (for longer runtimes or powering compute/sensing equipment), and can easily support higher speeds such that future enhancements such as LiDAR can be tested in representative environments. The total cost of the platform is approximately \$400 including batteries, or \$500 for a version with large storage, a long-range gamepad controller, and a Tensor Accelerator onboard.

Unlike other large-scale platforms like F1/10th [55], AutoRally [56] or the QCar [5], which cost more than our proposed platform, we offer low-cost extensibility, off-road capability, and ruggedness.

Computing is provided by a Raspberry Pi 3B+ (we have also tested with a Raspberry Pi 4 and Google Coral tensor accelerator), while a Navio2 [57] provides an Inertial Measurement Unit (IMU) and I/O for RC radios, servos and motor controllers. Input is provided by a Raspberry Pi camera with 130 degree field of view and IR filter (to improve daytime performance), and optionally a 360-degree planar YDLIDAR X4 [58] to measure radial distances. The platform receives human input from a Logitech F710 dual analog USB joystick.

The HobbyKing platform is four-wheel drive and has a brushless motor capable of over 20 kph. The Pi is vibrationally-isolated on an acrylic plate, reducing mechanical noise and providing crash protection. The camera is mounted atop the same acrylic plate and protected by an aluminum enclosure to minimize damage during collisions. The camera is mounted to a 3D printed bracket, the angle of which was set to provide an appropriate field of view for line detection. A LiDAR, if used, is mounted to this same plate using standoffs to raise the height above the camera enclosure. The vehicle platform is shown in Figure 14.

The Raspberry Pi runs Raspbian Stretch with realtime kernel provided by Emlid. At boot, the OS launches the Ardupilot [59] service, mavros [60], and the joy node. If LiDAR is used, the rplidar node is loaded. The user launches one of two Python nodes via SSH: a teleop node, which uses the joystick to control the car and logs telemetry to an onboard SD card at 10 Hz, or a model node, which uses one or more camera images and a pretrained model to command the steering servo to follow line markers using a pretrained neural network. In this mode, the user manually controls the throttle using the F710. Motor and servo commands take the form of a pulse width command ranging from 1000–2000 μ S, published to the `/mavros/rc/override` topic.

Figure 14. This 1/10th scale buggy features a Raspberry Pi 3B+, Navio2 interface board, Logitech F710 USB dongle, and a Raspberry Pi camera. Not pictured: LiDAR.

When teleop mode is started, IMU and controller data are captured by ROS subscribers and written to .CSV file, along with the 160×120 RGB .JPG image captured from *picamera* at the same time step. An overview of ROS architecture, including nodes and topics, appears in Figure 15.

Figure 15. A series of ROS nodes communicate with the ROS Core in order to exchange telemetry, sensor data, and control information.

5.2. Modular Training Environment

We designed a test track using reconfigurable “monomers” to create repeatable environments. We created track elements using low-cost $\frac{1}{2}$ EVA foam tiles. Components included tight turns, squared and rounded turns, sweeping turns, straightaways, and lane changes. Tiles are shown in Figure 16.

Straightaways and rounded (fixed-width) corners provide the simplest features for classification; wide, sweeping and right-angle corners provide challenging markings.

The reconfigurable track is quick to setup and modular compared with placing tape on the ground. It also helps to standardize visual indicators, similar to how lane dividers have fixed dimensions depending on local laws. Inexpensive track components can be stored easily, making this approach suitable to budget-constrained organizations. Sample track layouts appear in Figure 17.

5.3. Integration with Gamified Digital Simulator

Simulation affords researchers low-cost, high-speed data collection across environments. We use the GDS to parallelize and crowdsource data collection without the space, cost, or setup requirements associated with capturing data from conventional vehicles. We aim for a physical vehicle to “learn” to drive visually from GDS inputs.

In Section 4, we describe the creation of an in-game proxy for the physical platform. The virtual car mirrors the real vehicle’s physics, and driven using the same F710 joystick. The simulator roughly matches the friction coefficient, speed, and steering sensitivity between the vehicles, with numeric calibration where data were readily quantifiable (e.g., field of view for the camera).

The virtual car saves images to disk in the same format as the physical vehicle to maximize data interoperability, and exports throttle position, brake position, and steering angle to a CSV.

(a) This monomer, made up of two half-tiles, is a high-speed and narrow “lane change”.

(b) This monomer, made up of two half-tiles and two full tiles, is a high-speed and very wide “lane change” piece. It is difficult to navigate as only one lane marker is visible at a time (or none, if the car is perfectly centered).

(c) This monomer, made up of one tile, is a very sharp 90-degree turn. When approached head-on, there are only two small angled pieces of tape to differentiate left from right turns (the vertical and horizontal lines dominate the field of view).

(d) This monomer is a sweeping turn made up of one tile, and is fairly easy for the vehicle to follow. However, the turn radius is almost exactly the maximum allowed by the car’s steering geometry.

(e) This monomer is a single-tile straightaway, slightly wider than the width of the buggy.

Figure 16. These monomers are made of 24 square EVA foam interlocking gym tiles and combine to form a range of track configurations with varying complexity.

(a) This outdoor track layout is similar to the original rectangular surface used in the coin-collecting game.

(b) This track mimics the track geometry used in the simulator with invisible colliders.

Figure 17. We tested multiple physical tracks both matching the layouts in-game and of entirely new designs.

6. Data Collection, Algorithm Implementation and Preliminary Results

For our purposes, the development of a line following algorithm functioning in both simulation and on the physical platform serves as a representative “minimum viable test case,” with successful performance in both the simulated and real environment reflective of the suitability of crowd-generated data for training certain self-driving algorithms, as well as of the relevance and tight coupling of both the hardware and software elements designed.

To prove the viability of the simulator as a training tool, we designed an experiment to collect line-following data in the simulated environment for training a behavior-cloning, lane-following Convolutional Neural Network (CNN). This approach is not designed for State of the Art performance; rather, it is to demonstrate the tool’s applicability to generating suitable training data and resulting trained models for operation on the test platform without explicit domain adaptation.

We generated training data both from “wisdom of the crowd” (multiple human drivers) and from “optimal AI” (steering and velocity based on logical rules and perfect situational information). We then attempted to repurpose the resulting model, without retraining, to the physical domain.

We first collected data in the simulated environment by manually driving laps in 10-min batches. As described in Section 5.3, the simulated camera images, steering angles, and throttle positions were written to file at 30 Hz. In addition to relying on “wisdom of the crowd” and implicit game mechanics enforcing effective data generation, samples with or near zero velocity, negative throttle (braking/reverse), or steering outside the control limits (indicating a collision) were ignored to prevent the algorithm from learning “near crash” situations. Such data can later be used for specialized training.

Rather than using image augmentation, we instead used the in-game “optimal AI” (AI with complete, noiseless sensor measurements and well characterized rules as described in Section 4.10) to generate additional ground-truth data from the simulator and ultimately used only symmetric augmentation (image and steering angle mirroring to ensure left/right turn balance).

When training CNNs on a fixed-image budget of 50 K or 100 K images, neither the human-only training approach, nor the AI-only approach individually resulted in models supporting in-game or real-world AI capable of completing laps. With larger numbers of images from each individual training source, an effective neural network might be trained. Instead, blending the training images 50%/50% from human and AI sources with the same image budget (50 and 100 K) yielded models capable of completing real-world laps (occasionally and repeatably, respectively). Our results indicate that the joint-training approach converges more quickly than either source alone and we posit that the combination of repeatability (assisted exploration via in-game AI) and variability (human operation) yields a more effective training set.

Final training data were approximately half human-driven and half controlled by AI with noiseless information. We collected 224,293 images, almost 450,000 images after symmetry augmentation. Data capture occurred rapidly; we were able to capture all necessary data. To the point of being faster than physical testing on a full-scale vehicle, it took less time to capture, train, and validate the model than a typical review cycle for experiments involving human subjects—approximately one week. Each image was then post-processed to extract informative features.

We developed an image preprocessing pipeline in OpenCV [61] similar to that proposed in the Udacity “Intro to Self Driving” Nano Degree Program capable of:

1. Correcting the image for camera lens properties
2. Conducting a perspective transform to convert to a top-down view
3. Conversion from RGB to HSL color space
4. Gaussian blurring the image
5. Masking the image to particular ranges of white and yellow
6. Greyscaling the image
7. Conducting Canny edge detection
8. Masking the image to a polygonal region of interest
9. Fitting lines using a Hough transform
10. Filtering out lines with slopes outside a particular range
11. Grouping lines by slope (left or right lines)
12. Fitting a best-fit line to the left and/or right side using linear regression
13. Creating an image of the best-fit lines
14. Blending the (best-fit) line(s) with the edge image, greyscale image, or RGB image

Using a sample CNN to estimate the significance of each preprocessing step, we conducted a grid search and identified a subset as being performance-critical. Ultimately, preprocessing comprised:

1. Conversion from RGB to HSL color space
2. HSL masking to allow only white and yellow regions to pass through into a binary (black/white) image
3. Gaussian blurring
4. Masking the image to a region including only the road surface and none of the vehicle or environs

This pipeline improved predictor robustness for varied lighting conditions without increasing per-frame processing time compared to raw RGB images. Example input and output from the real and simulated camera appear in Figure 18.

We additionally tested camera calibration and perspective transformation, but found these operations to add little performance relative to their computational complexity. The predicted steering angles and control loop update sufficiently quickly that over- or under-steering is easily addressed, and a maximum steering slew rate prevented the vehicle from changing steering direction abruptly, minimizing oscillation.

We trained 25+ CNN variants in Keras [47] using simulated data. For each model, we recorded outsample mean-squared error (MSE). We saved the best model and stopped training when the validation loss had not decreased more than 0.1 across the previous 200 epochs. In practice, this was approximately 400 epochs. In all cases, testing and validation loss decreased alongside each other for the entirety of training, indicating the models did not overfit.

(a) This is a sample RGB input image from the simulated environment.

(b) This image shows the preprocessed camera input (B&W) for the simulated environment. The lines stand out relative to the rest of the image, improving classifier robustness.

(c) This is a sample RGB input image from the Raspberry Pi camera on the physical vehicle.

(d) This image shows the preprocessed camera input (B&W) for the real environment. The extracted features are similar between the simulated and physical world, though the real-world has more noise and the white floor is retained as a potential “line feature”.

Figure 18. Preprocessing images is an efficient process that improves model transferrability between the simulated and physical world.

From these results, we selected two CNNs with the best outsample performance: one using a single input image, and one using a three-image sequence (current and the two preceding frames) to provide time history and context. The trained convolution layers were 2D for single-images and 3D for image sequences, which provided the sequence model with temporal context and improved velocity independence. The representative models chosen to demonstrate the simulator’s ability to generate data suitable for effective sim2real domain transfer, are shown in Figure 19a,b.

A comparison of the predictive performance of the single-image and multi-image model for the (simulated) validation set appears in Figure 20. These plots compare the predicted steering angle to the ground-truth steering angle, with a 1:1 slope indicating perfect fit.

(a) This 2D Convolutional Neural Network is the single-image model, taking a greyscale 160×120 image as input and outputting a single (float) angle.

(b) This 3D Convolutional Neural Network is the single-image model, taking three greyscale 160×120 images as input and outputting a single (float) angle.

Figure 19. The two models used to predict steering angle from greyscale images are both convolutional neural networks (CNNs)-2D for the single-image case, and 3D for the image-sequence case. Both models minimize MSE and use the Adam optimizer.

(a) This figure shows the predicted versus ground-truth steering angles for the single-image model, with a reasonably strong diagonal component despite few outlying values.

(b) This figure shows the predicted versus ground-truth steering angles for the multiple-image model, with a strong diagonal clustering.

Figure 20. Comparing the model performance for the single- and multi-image predictor, using in-game images captured at approximately 30 frames per second.

7. Model Testing and Cross-Domain Transferrability

We tested the trained model in both virtual and physical environments to establish qualitative performance metrics related to domain adaptation.

7.1. Model Validation (Simulated)

We first tested the model in the simulator, using Keras to process output images from the virtual camera. The neural network monitored the image output directory, running each new frame through a pretrained model to predict the steering angle. This steering angle and a constant throttle value were converted to simulated joystick values and written to an input file monitored by the simulator. The simulator updated the vehicle control with these inputs at 30 Hz, so the delay between the image creation, processing, and input was 0.06 s or better. This process is described in Section 4.9.

The simulated vehicle was able to reliably navigate along straightaways and fixed-radius turns. It also correctly identified the directionality of tight and sweeping corners with high accuracy, though it struggled with hard-right-angle turns, suggesting the model identifies turns by looking for curved segments rather than by corner geometry. For some initial starting positions, the simulated vehicle could complete >20 laps without incident. For other seeds values, the vehicle would interact with the invisible colliders and “ping-pong” against the walls (directional trends were correct, but narrow lanes left little room for error). In these cases, human intervention unstuck the vehicle and the buggy would resume self-driving.

We next transferred the most-effective model to the physical vehicle without explicit transfer learning.

7.2. Model Transferrability to Physical Platform

We ported the pretrained model to the physical vehicle, changing only the HSL lightness range for OpenCV’s white mask and changed the polygonal mask region to block out the physical buggy’s suspension, and scaled the predicted output angle from degrees to microsecond servo pulses. There was no camera calibration and no perspective transformation.

As with the simulated vehicle, the buggy was able to follow straight lines and sweeping corners using the unaltered single- and multi-image models, with the vehicle repeatedly completing several laps. It was not necessary to retrain the model. Both the real car and simulator control loops operate at low loop rates (8–30 Hz) and speeds (≈ 10 kph), so the classifiers’ predicted steering angles cause each vehicle to behave as though being operated with a “bang-bang” controller (“left-right”) rather than a nuanced PID controller (see [video](#) (accessed on 2 October 2020) running a representative image sequence model at low speeds). Though line-following appears “jerky,” small disparities between the calibration and sensitivity of the virtual and physical vehicle’s steering response minimally impact lane keeping as framerates (enabled with enhanced compute) increase.

We also qualitatively evaluated the single- and multi-image models relative to their performance in the simulated environment. In practice, the model relying on the single image worked most robustly within the simulated environment. This is because the images for training were captured at a constant 30 Hz, but the vehicle speed varied throughout these frames. Because the 3D convolution considers multiple frames at fixed time intervals, there is significant velocity dependence. The desktop was able to both process and run the trained model at a consistent 30 Hz, including preprocessing, classification, polling for override events from the joystick, and writing the output file, making the single-image model perform well and making the impact of reduced angular accuracy relative to the image-sequence model insignificant as the time-delta between control inputs was only 0.03 s.

In the physical world, where the buggy speed and frame capture and processing are slower, the vehicle’s velocity variation is a smaller percentage of the mean velocity and computational complexity matters more. As a result, the image sequence provides better performance as it anticipates upcoming turns without the complication of high inter-frame velocity variation. The physical vehicle performed laps consistently with the image-sequence model, though it still struggled with the same right-angle turns as the simulator.

Other models (e.g., LSTMs) may offer improved performance over those demonstrated; an ablation study could validate this. However, those chosen models demonstrate effectively the desired goal of model transferrability and the function of the end-to-end simulator and physical platform toolchain.

These results show successful model transferrability from the simulated to physical domain without retraining. The gamified driving simulator and low-cost physical platform provide an effective end-to-end solution for crowdsourced data collection, algorithm training, and model validation suitable for resource-sensitive research and development environments.

8. Conclusions, Discussion, and Future Work

Our toolchain uniquely combines simulation, gamification, and extensibility with a low-cost physical test platform. This combination supports semi-supervised, crowdsourced data collection, rapid algorithm development cycles, and inexpensive model validation relative to contemporary solutions.

There are opportunities for future improvement. For example, adding a calibration pattern would allow us to evaluate the impact of camera calibration on model performance. We plan to include simulated LiDAR to improve the simulator's utility for collision avoidance, and to create a track-builder utility or procedural track generator. Incorporating multiplayer, simulated traffic, and/or unpredictable events ("moose crossing" or passing cyclists [62]) would help train more complex scenarios. Alternatively, an extension of the simulator may be used to train adversarial networks for self-driving to support automated defensive driving techniques [63], or high-performance vehicles [64].

Because the simulator is based on a multi-platform game engine, broader distribution and the creation of improved scoring mechanisms and game modes will provide incentive for players to contribute informative supervised data, supporting rapid behavior cloning for long-tail events. These same robust scoring metrics would allow us to rank the highest-performing drivers' training data more heavily than lower-scoring drivers when training the neural network. Some of these can be visible to the user (a "disqualification" notice), while other score metrics may be invisible (a hidden collider object could disable image and CSV capture and deduct from the user's score when the vehicle leaves a "safe" region). Other changes to game mechanics might support the capture of necessary "edge case" unsuitable to generate by other means—particularly with typical research labs' economic or computational constraints. These adaptations might, for example, contribute to the capture of data from incident avoidance scenarios, or operating in inclement weather, or during particular vehicle failures (such as a tire blowout). Multiplayer driving, with AI or human operated other vehicles, would generate other sorts of edge case data to enhance data capture via naturalistic ablation. This work will require developing a network backend for data storage and retrieval from diverse devices. Changing game mechanics in some respects is simpler and more comprehensively addresses evolving data capture needs than seeking to explicitly codify desirable and undesirable data capture.

There may also be opportunities to integrate the simulator and physical vehicles into an IoT framework [65] or with AR/VR tools [66], such that physical vehicles inform the simulation in realtime and vice-versa.

Finally, crowdsourcing only works if tools are widely available, and we plan to refine and release the compiled simulator, test platform details, and resulting datasets to the public.

Author Contributions: Conceptualization—G.P., J.E.S. Resources, Data Curation, and Administration—J.E.S. Methodology—G.P., J.E.S.; Software—G.P., J.E.S., Y.S.; Investigation—G.P., J.E.S.; Writing—Original Draft—G.P. and J.E.S.; Writing—Review and Editing—G.P., J.E.S., K.P., Y.S. Visualization—G.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: We thank the NVIDIA Corporation for providing the Titan Xp GPU used in this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

- O’Kane, S. How Tesla and Waymo Are Tackling a Major Problem for Self-Driving Cars: Data. Available online: <https://www.theverge.com/transportation/2018/4/19/17204044/tesla-waymo-self-driving-car-data-simulation> (accessed on 1 June 2019).
- Stewart, J. Tesla’s Autopilot Now Changes Lanes—And You’re Gonna Help It Out. Available online: <https://www.wired.com/story/tesla-navigate-on-autopilot/> (accessed on 1 June 2019).
- Kassens-Noor, E.; Neal, Z.; Siegel, J.; Decaminada, T. Choosing Morals or Ethics: A Possible Determinant to Embracing Autonomous Vehicles? Poster presented at Transportation Research Board Annual Meeting. In Proceedings of the Transportation Research Board Annual Meeting, Online, 12 February 2021.
- Tangen, J.M.; Kent, K.M.; Searston, R.A. Collective intelligence in fingerprint analysis. *Cogn. Res. Princ. Implic.* **2020**, *5*, 23. [CrossRef]
- Hu, J.; Zhang, Y.; Rakheja, S. Adaptive Trajectory Tracking for Car-like Vehicles with Input Constraints. *IEEE Trans. Ind. Electron.* **2021**. [CrossRef]
- Balaji, B.; Mallya, S.; Genc, S.; Gupta, S.; Dirac, L.; Khare, V.; Roy, G.; Sun, T.; Tao, Y.; Townsend, B.; et al. Deepracer: Autonomous racing platform for experimentation with sim2real reinforcement learning. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; pp. 2746–2754.
- Siegel, J.E.; Coda, U. Surveying Off-Board and Extra-Vehicular Monitoring and Progress Towards Pervasive Diagnostics. *arXiv* **2021**, arXiv:2007.03759.
- Madrigal, A.C. Inside Waymo’s Secret World for Training Self-Driving Cars. Available online: <https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/> (accessed on 1 June 2019).
- Shafaei, A.; Little, J.J.; Schmidt, M. Play and Learn: Using Video Games to Train Computer Vision Models. *arXiv* **2016**, arXiv:1608.01745.
- Johnson-Roberson, M.; Barto, C.; Mehta, R.; Sridhar, S.N.; Vasudevan, R. Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks? *arXiv* **2016**, arXiv:1610.01983.
- Hawkins, A.J. It’s Elon Musk vs. Everyone Else in the Race for Fully Driverless Cars. Available online: <https://www.theverge.com/2019/4/24/18512580/elon-musk-tesla-driverless-cars-lidar-simulation-waymo> (accessed on 1 June 2019).
- Teoh, E.R.; Kidd, D.G. Rage against the machine? Google’s self-driving cars versus human drivers. *J. Saf. Res.* **2017**, *63*, 57–60. [CrossRef] [PubMed]
- Rockstar Games. Grand Theft Auto. Available online: <https://www.rockstargames.com/V/> (accessed on 1 June 2019).
- Martinez, M.; Sitawarin, C.; Finch, K.; Meincke, L.; Yablonski, A.; Kornhauser, A.L. Beyond Grand Theft Auto V for Training, Testing and Enhancing Deep Learning in Self Driving Cars. *arXiv* **2017**, arXiv:1712.01397.
- Franke, C. Autonomous Driving with a Simulation Trained Convolutional Neural Network. Master’s Thesis, University of the Pacific, Stockton, CA, USA, 2017.
- Fridman, L.; Jenik, B.; Terwilliger, J. DeepTraffic: Driving Fast through Dense Traffic with Deep Reinforcement Learning. *arXiv* **2018**, arXiv:1801.02805.
- VDrift. VDrift. Available online: <http://www.vdrift.net> (accessed on 1 June 2019).
- Haltakov, V.; Unger, C.; Ilic, S. Framework for Generation of Synthetic Ground Truth Data for Driver Assistance Applications. In *Pattern Recognition*; Weickert, J., Hein, M., Schiele, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 323–332.
- Kramer, T. SdSandbox. Available online: <https://github.com/tawnkramer/sdsandbox/tree/donkey> (accessed on 1 June 2019).
- Yu, F. Train Donkey Car in Unity Simulator with Reinforcement Learning. Available online: <https://flyyufelix.github.io/2018/09/11/donkey-rl-simulation.html> (accessed on 1 June 2019).
- Roscoe, W.; Kramer, T. Donkey Simulator. Available online: <https://docs.donkeycar.com/guide/simulator/> (accessed on 1 June 2019).
- Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An Open Urban Driving Simulator. In Proceedings of the 1st Annual Conference on Robot Learning, Mountain View, CA, USA, 13–15 November 2017; pp. 1–16.
- Marín, J.; Vázquez, D.; Gerónimo, D.; López, A.M. Learning appearance in virtual scenarios for pedestrian detection. In Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, USA, 13–18 June 2010; pp. 137–144. [CrossRef]
- Hattori, H.; Boddeti, V.N.; Kitani, K.; Kanade, T. Learning scene-specific pedestrian detectors without real data. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3819–3827. [CrossRef]
- Filipowicz, A.; Liu, J.; Kornhauser, A.L. Learning to Recognize Distance to Stop Signs Using the Virtual World of Grand Theft Auto 5. In Proceedings of the Transportation Research Board 96th Annual Meeting, Washington, DC, USA, 8–12 January 2017.

26. Togelius, J.; Lucas, S.M. Evolving robust and specialized car racing skills. In Proceedings of the 2006 IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 1187–1194. [\[CrossRef\]](#)
27. Deng, J.; Dong, W.; Socher, R.; Li, L.; Li, K.; Li, F.-F. ImageNet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255. [\[CrossRef\]](#)
28. Liu, Z.; Luo, P.; Wang, X.; Tang, X. Deep Learning Face Attributes in the Wild. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 3730–3738. [\[CrossRef\]](#)
29. Zhou, T.; Krähenbühl, P.; Aubry, M.; Huang, Q.; Efros, A.A. Learning Dense Correspondence via 3D-Guided Cycle Consistency. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 117–126. [\[CrossRef\]](#)
30. Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 23–30. [\[CrossRef\]](#)
31. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 1137–1149. [\[CrossRef\]](#) [\[PubMed\]](#)
32. Sundermeyer, M.; Marton, Z.C.; Durner, M.; Brucker, M.; Triebel, R. Implicit 3d orientation learning for 6D object detection from RGB images. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 699–715.
33. Kehl, W.; Manhardt, F.; Tombari, F.; Illic, S.; Navab, N. SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 1521–1529.
34. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. In *Computer Vision—ECCV 2014*; Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T., Eds.; Springer: Cham, Switzerland, 2014; pp. 740–755.
35. Liu, X.; Qi, C.R.; Guibas, L.J. FlowNet3D: Learning Scene Flow in 3D Point Clouds. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019.
36. Mayer, N.; Ilg, E.; Hausser, P.; Fischer, P.; Cremers, D.; Dosovitskiy, A.; Brox, T. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 4040–4048.
37. Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. Vision meets robotics: The KITTI dataset. *Int. J. Robot. Res.* **2013**, *32*, 1231–1237. [\[CrossRef\]](#)
38. Sun, Y.; Liu, Z.; Wang, Y.; Sarma, S.E. Im2Avatar: Colorful 3D Reconstruction from a Single Image. *arXiv* **2018**, arXiv:1804.06375.
39. Chang, A.X.; Funkhouser, T.A.; Guibas, L.J.; Hanrahan, P.; Huang, Q.; Li, Z.; Savarese, S.; Savva, M.; Song, S.; Su, H.; et al. ShapeNet: An Information-Rich 3D Model Repository. *arXiv* **2015**, arXiv:1512.03012.
40. Xiang, Y.; Mottaghi, R.; Savarese, S. Beyond PASCAL: A benchmark for 3D object detection in the wild. In Proceedings of the IEEE Winter Conference on Applications of Computer Vision, Steamboat Springs, CO, USA, 24–26 March 2014; pp. 75–82. [\[CrossRef\]](#)
41. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*; Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q., Eds.; Curran Associates, Inc.: Nice, France, 2014; pp. 2672–2680.
42. Shrivastava, A.; Pfister, T.; Tuzel, O.; Susskind, J.; Wang, W.; Webb, R. Learning from Simulated and Unsupervised Images through Adversarial Training. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 2242–2251. [\[CrossRef\]](#)
43. Bousmalis, K.; Silberman, N.; Dohan, D.; Erhan, D.; Krishnan, D. Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 95–104. [\[CrossRef\]](#)
44. Bousmalis, K.; Irpan, A.; Wohlhart, P.; Bai, Y.; Kelcey, M.; Kalakrishnan, M.; Downs, L.; Ibarz, J.; Pastor, P.; Konolige, K.; et al. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018; pp. 4243–4250. [\[CrossRef\]](#)
45. Rosique, F.; Navarro Lorente, P.; Fernandez, C.; Padilla, A. A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research. *Sensors* **2019**, *19*, 648. [\[CrossRef\]](#) [\[PubMed\]](#)
46. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: <https://www.tensorflow.org/> (accessed on 1 June 2019).
47. Keras. 2015. Available online: <https://github.com/fchollet/keras> (accessed on 1 June 2019).
48. Von Ahn, L. Games with a Purpose. *Computer* **2006**, *39*, 92–94. [\[CrossRef\]](#)
49. Walther-Franks, B.; Smeddinck, J.; Szmidt, P.; Haidu, A.; Beetz, M.; Malaka, R. Robots, Pancakes, and Computer Games: Designing Serious Games for Robot Imitation Learning. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems CHI '15, Seoul, Korea, 18–23 April 2015; Association for Computing Machinery: New York, NY, USA, 2015; pp. 3623–3632. [\[CrossRef\]](#)

50. Lazzaro, N. The 4 Keys 2 Fun. Available online: <http://www.nicolelazzaro.com/the4-keys-to-fun/> (accessed on 1 June 2019).
51. Lazzaro, N. Why we Play Games: Four Keys to More Emotion without Story. In Proceedings of the Game Developers Conference, San Jose, CA, USA, 22–26 March 2004.
52. PyGame Developers. PyGame. Available online: <http://pygame.org/> (accessed on 1 June 2019).
53. Robotis. PLATFORM-TurtleBot 3. Available online: <http://www.robotis.us/turtlebot-3/> (accessed on 1 June 2019).
54. DonkeyCar. Donkey® Car-Home. Available online: <https://www.donkeycar.com/> (accessed on 1 June 2019).
55. O’Kelly, M.; Sukhil, V.; Abbas, H.; Harkins, J.; Kao, C.; Pant, Y.V.; Mangharam, R.; Agarwal, D.; Behl, M.; Burgio, P.; et al. F1/10: An Open-Source Autonomous Cyber-Physical Platform. *arXiv* **2019**, arXiv:1901.08567.
56. Goldfain, B.; Drews, P.; You, C.; Barulic, M.; Velev, O.; Tsiotras, P.; Rehg, J.M. Autorally: An open platform for aggressive autonomous driving. *IEEE Control. Syst. Mag.* **2019**, *39*, 26–55. [CrossRef]
57. Emlid. Navio2 | Emlid. Available online: <https://emlid.com/navio/> (accessed on 1 June 2019).
58. Technology, Y. YD LIDAR X4. Available online: <http://ydlidar.com/product/X4> (accessed on 1 June 2019).
59. Ardupilot. ArduPilot Open Source Autopilot. Available online: <http://www.ardupilot.org/> (accessed on 1 June 2019).
60. mavros-ROS Wiki. Available online: <http://wiki.ros.org/mavros> (accessed on 1 June 2019).
61. Bradski, G. The OpenCV Library. *Dr. Dobbs’ J. Softw. Tools* **2000**, *120*, 122–125.
62. Barnett, J.; Gizinski, N.; Mondragon-Parra, E.; Siegel, J.E.; Morris, D.; Gates, T.; Kassens-Noor, E.; Savolainen, P. Automated Vehicles Sharing the Road: Surveying Detection and Localization of Pedalcyclists. *IEEE Trans. Intell. Veh.* **2020**. [CrossRef]
63. Gupta, P.; Coleman, D.; Siegel, J.E. Towards Safer Self-Driving Through Great PAIN (Physically Adversarial Intelligent Networks). *arXiv* **2020**, arXiv:cs.LG/2003.10662.
64. Siegel, J.; Morris, D., Robotics, Automation, and the Future of Sports. In *21st Century Sports: How Technologies Will Change Sports in the Digital Age*; Schmidt, S.L., Ed.; Springer: Cham, Switzerland, 2020; pp. 53–72. [CrossRef]
65. Wilhelm, E.; Siegel, J.; Mayer, S.; Sadamori, L.; Dsouza, S.; Chau, C.K.; Sarma, S. Cloudthink: A scalable secure platform for mirroring transportation systems in the cloud. *Transport* **2015**, *30*, 320–329. [CrossRef]
66. Pappas, G.; Siegel, J.; Politopoulos, K. VirtualCar: Virtual Mirroring of IoT-Enabled Avacars in AR, VR and Desktop Applications. In *ICAT-EGVE 2018-International Conference on Artificial Reality and Telexistence and Eurographics Symposium on Virtual Environments-Posters and Demos*; Huang, T., Otsuki, M., Servièrès, M., Dey, A., Sugiura, Y., Banakou, D., Michael-Grigoriou, D., Eds.; The Eurographics Association: Lower Saxony, Germany, 2018. [CrossRef]