UE4Sim: A Photo-Realistic Simulator for Computer Vision Applications

Matthias Mueller, Vincent Casser, Jean Lahoud, Neil Smith, Bernard Ghanem
Visual Computing Center, KAUST, Thuwal, Saudi Arabia
{matthias.mueller.2, vincent.casser, jean.lahoud, neil.smith, bernard.ghanem}@kaust.edu.sa

Abstract

We present a photo-realistic training and evaluation simulator (UE4Sim) with extensive applications across various fields of computer vision. Built on top of the Unreal Engine, the simulator integrates full featured physics based cars, unmanned aerial vehicles (UAVs), and animated human actors in diverse urban and suburban 3D environments. We demonstrate the versatility of the simulator with two case studies: autonomous UAV-based tracking of moving objects and autonomous driving using supervised learning. The simulator fully integrates both several state-of-the-art tracking algorithms with a benchmark evaluation tool and a deep neural network (DNN) architecture for training vehicles to drive autonomously. It generates synthetic photo-realistic datasets with automatic ground truth annotations to easily extend existing real-world datasets and provides extensive synthetic data variety through its ability to reconfigure synthetic worlds on the fly using an automatic world generation tool.

1. Introduction

The photo-realism of modern game engines provides a new avenue for developing and evaluating methods for diverse sets of computer vision (CV) problems, which will ultimately operate in the real-world. In particular, game engines such as Unity, UNIGINE, CRYENGINE, and Unreal Engine 4 (UE4) have begun to open-source their engines to allow low-level redesign of core functionality in their native programming languages. This full control over the gaming engine allows researchers to develop novel applications and lifelike physics based simulations, while benefiting from the free generation of photo-realistic synthetic visual data. Since these modern game engines run in real-time, they provide an end-to-end solution for training with synthetic data, conducting controlled experiments, and real-time benchmarking. As they approach not only photo-realism but lifelike physics simulation, the gap between simulated and real-world applications is expected to substantially decrease. In this paper, we present UE4Sim, a full featured, customizable, physics-based simulator built within the Unreal Engine 4. The simulator directly provides accurate car and UAV physics, as well as, both the latest state-of-the-art tracking algorithms with a benchmark evaluation tool and a TensorFlow-based deep learning interface.

UE4Sim allows access to both visual data captured from cameras mounted in the simulated environment and semantic information that can be used for learning-based CV applications. For example, in addition to RGB images, the simulator provides numerous capabilities, such as depth, segmentation, and ground truth labelling, which can enable a wide variety of applications as shown in Fig. 1. More details on the simulator capabilities and applications are presented in Sec. 3. Although recent work by [47] has shown the advantages of using pre-built simulated worlds (*e.g.* GTA V's Los Angeles city), this approach is not easily reconfigurable. While these worlds are highly detailed, they are not amenable to user customization, which limits the potential variety needed for large-scale data generation and extensive evaluation in diverse scenarios. To address this drawback and unlike other simulators used for CV purposes, we divide a large variety of high-poly Physically-Based Rendering (PBR) textured assets into building blocks that can be placed and configured within a simple GUI and then procedurally generated within UE4Sim at runtime. Moreover, dynamic agents (*e.g.* pedestrians and cars) can be included to generate more dynamic scenarios. Of course, the open source nature of the implementation gives the user the freedom to modify/prune/enrich this set of assets. A similar strategy can be taken for indoor scenes as well. As such, this process can generate a very rich variety of city and suburban scenes, thus, bolstering the generation of diverse datasets for deep neural network (DNN) training, as a step towards preventing the

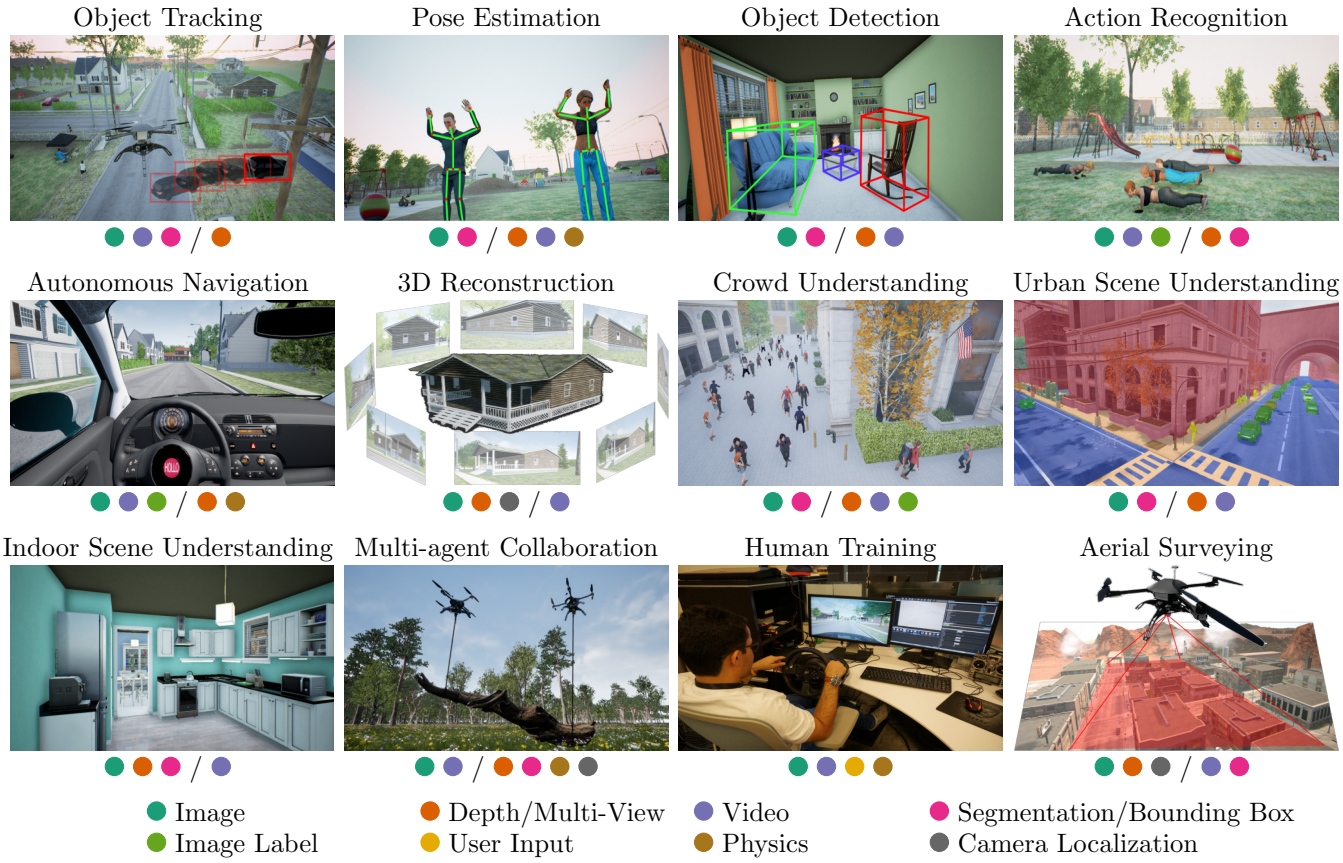


Figure 1: An overview of some common applications in computer vision in which our simulator can be used for generating synthetic data and performing (real-time) evaluation. For each application, we show which type of data is usually a necessity (*e.g.* segmentation masks/bounding box annotations for learning object detection) and which can be used optionally.

over-fitting of DNN methods and fostering better generalization properties. To advocate the generality of UE4Sim, we adopt two popular use cases from the CV literature: real-time tracking evaluation from a UAV and autonomous car driving.

Empowering UAVs with automated CV capabilities (*e.g.* tracking, object/activity recognition, mapping, *etc.*) is becoming a very important research direction in the field and is rapidly accelerating with the increasing availability of low-cost, commercially available UAVs. In fact, aerial tracking has enabled many new applications in computer vision (beyond those related to surveillance), including search and rescue, wild-life monitoring, crowd monitoring/management, navigation/localization, obstacle/object avoidance, and videography of extreme sports. Aerial tracking can be applied to a diverse set of objects (*e.g.* humans, animals, cars, boats), many of which cannot be physically or persistently tracked from the ground. In particular, real-world aerial tracking scenarios pose new challenges to the tracking problem, exposing areas for further research. In UE4Sim, one can directly feed video frames captured from a camera onboard a UAV to CV trackers and retrieve their tracking results to update UAV flight. Any tracker (*e.g.* written in MATLAB, Python, or C++) can be tested within the simulator across a diverse set of photo-realistic simulated scenarios allowing new quantitative methods for evaluating tracker performance. In fact, this paper extends our previous work [33] by evaluating trackers on a new more extensive synthetic dataset.

Inspired by recent work on self-driving cars by [4] and the synthetic KITTI dataset by [12], we implement a TensorFlow-based DNN interface with UE4Sim. In our self-driving application, both UE4Sim and the DNN run at real-time speeds in parallel, allowing fully interactive evaluation to be performed. The DNN is trained in a supervised manner by exploiting the extensive and diverse synthetic visual data that can be captured and the free accompanying labelling (*i.e.* way-

point coordinates). With some required acceleration measures, our simulator can also be extended to facilitate reinforcement learning methods for this task, although this remains outside the scope of this paper.

Contributions. The contributions of our work are four-fold: **(1)** An end-to-end physics-based, fully customizable, open-source simulator environment for the CV community working on autonomous navigation, tracking, and a wide variety of other applications; **(2)** a customizable synthetic world generation system; **(3)** a novel approach for tracker evaluation with a high-fidelity real-time visual tracking simulator; and **(4)** a novel, robust deep learning based approach for autonomous driving that is flexible and does not require manually collected training data.

2. Related Work

2.1. Learning from Simulation

A broad range of work has recently exploited physics based simulators for learning purposes, namely in animation and motion planning ([18; 15; 27; 24; 51; 16; 14]), scene understanding ([3; 38]), pedestrian detection ([29]), and identification of 2D/3D objects ([17; 31; 39]). For example, in [18], a physics-based computer game environment (Unity) is used to teach a simulated bird to fly. Moreover, hardware-in-the-loop (HIL) simulation has also been used in robotics to develop and evaluate controllers and for visual servoing studies (*e.g.* JMAVSim, [44; 52] and RotorS by [11]). The visual rendering in these simulators is often primitive and relies on off-the-shelf simulators (*e.g.* Realflight, Flightgear or XPlane). They do not support advanced shading and post-processing techniques, are limited in terms of available assets and textures, and do not support motion capture (MOCAP) or key-frame type animation to simulate natural movement of actors or vehicles.

Recent work (*e.g.* [12; 47; 9]) show the advantages of exploiting the photo-realism of modern game engines to generate training datasets and pixel-accurate segmentation masks. Since one of the applications presented in this work is UAV tracking, we base our own UE4Sim simulator on our recent work ([33]), which provides full hardware and software in-the-loop UAV simulation built on top of the open source Unreal Engine 4.

2.2. UAV Tracking

A review of related work indicates that there is still a limited availability of annotated datasets specific to UAVs, in which trackers can be rigorously evaluated for precision and robustness in airborne scenarios. Existing annotated video datasets include very few aerial sequences ([53]). Surveillance datasets such as PETS or CAVIAR focus on static surveillance and are outdated. VIVID [6] is the only publicly available dedicated aerial dataset, but it is outdated and has many limitations due to its small size (9 sequences), very similar and low-resolution sequences (only vehicles as targets), sparse annotation (only every 10th frame), and focus on higher altitude, less dynamic fixed-wing UAVs. There are several recent benchmarks that were created to address specific deficiencies of older benchmarks and introduce new evaluation approaches [25; 26; 49], but they do not introduce videos with many tracking nuisances addressed in this paper and common to aerial scenarios.

Despite the lack of benchmarks that adequately address aerial tracking, the development of tracking algorithms for UAVs has become very popular in recent years. The majority of object tracking methods employed on UAVs rely on feature point detection/tracking ([46; 37]) or color-centric object tracking ([19]). Only a few works in the literature ([40]) exploit more accurate trackers that commonly appear in generic tracking benchmarks such as MIL in [2; 10], TLD in [40], and STRUCK in [28; 32]. There are also more specialized trackers tailored to address specific problems and unique camera systems such as in wide aerial video ([41; 45]), thermal and IR video ([13; 43]), and RGB-D video ([36]). In UE4Sim, the aforementioned trackers, as well as, any other tracker, can be integrated into the simulator for real-time aerial tracking evaluation, thus, standardizing the way trackers are compared in a diverse and dynamic setting. In this way, state-of-the-art trackers can be extensively tested in a life-like scenario before they are deployed in the real-world.

2.3. Autonomous Driving

Work by [4; 5; 50; 42; 35; 1; 20; 48] show that with sufficient training data and augmented camera views autonomous driving and flight can be learned by a DNN. The driving case study for UE4Sim is primarily inspired by [5] and other work ([30; 27; 21; 22]), which uses TORCS (The Open Racing Car Simulator by [54]) to train a DNN to drive at casual speeds through a course and properly pass or follow other vehicles in its lane. The vehicle controls are predicted in [5] as a discrete set of outputs: turn-left, turn-right, throttle, and brake. The primary limitation of TORCS for DNN development is that the environment in which all training is conducted is a race track with the only diversity being the track layout. This bounded environment does not include the most common scenarios relevant to autonomous driving, namely urban, suburban, and rural environments, which afford complexities that are not

present in a simple racing track (*e.g.* pedestrians, intersections, cross-walks, 2-lane on-coming traffic, *etc.*).

The work of [47] proposed an approach to extract synthetic visual data directly from the Grand Theft Auto V (GTA V) computer game. Of particular interest is the high-quality, photo-realistic urban environment in which rich driving data can be extracted, along with the logging of user input for supervised (SL) or reinforcement learning (RL). However, a primary limitation of this approach is that the virtual world, although generally dynamic and highly photo-realistic, cannot be interactively controlled or customized. This limits its flexibility in regards to data augmentation, evaluation, and repeatability, thus, ultimately affecting the generalization capabilities of DNNs trained in this setup. One insight we highlight in this paper is that without adding additional shifted views, the purely visual driving approach overfits and fails. Since synthetic data from GTA V is limited to a single camera view, there is no possible way to augment the data or increase the possible variety of views. Second, repeatability is not possible either, since one cannot start an evaluation in the exact spot everytime, control the randomized path of cars and people in the world, or programmatically control or reconfigure anything else in the game.

Virtual KITTI by [12] provides a Unity based simulation environment, in which real-world video sequences can be used as input to create virtual and realistic proxies of the real-world. They demonstrate that the gap is small in transferring between DNN learning on synthetic videos and their real-world counterparts, thus, emphasizing the crucial need for photo-realistic simulations. They generate 5 cloned worlds and 7 variations of each to create a large dataset of 35 video sequences with about 17,000 frames. The synthetic videos are automatically annotated and serve as ground truth for RGB tracking, depth, optical flow, and scene segmentation. This work is primarily focused on generating video sequences and does not explore actual in-game mechanics, such as the physics of driving the vehicles, movement of actors within the world, dynamic scene interactions, and real-time in-world evaluation. Control predictions using DNNs is not addressed in their work, since the focus is on the evaluation of trained DNN methods on the video sequences extracted from the engine. Our work addresses these unexplored areas, building upon a fundamental conclusion of [12], which states that proxy virtual worlds do have high transferability to the real-world (specifically for DNN based methods) notably in the case of autonomous driving.

Although [4] collected datasets primarily from the real-world in their DNN autonomous driving work, their approach to create additional synthetic images and evaluate within a simulation is very relevant to the work presented here. The simulated and on-road results of their work demonstrate advances in how a DNN can learn end-to-end the control process of a self-driving car directly from raw input video data. However, the flexibility regarding augmentation of data collected in the real-world is strongly constrained, so much so, that they had to rely on artificial view interpolations to get a limited number of additional perspectives. UE4Sim does not share this disadvantage, since it can generate any amount of data needed in the simulator, as well as, evaluate how much and what type of view augmentation is needed for the best performing DNN. Furthermore, our DNN approach for the UE4Sim driving case study does not require human training data, and it possesses a number of additional advantages by design, such as flexibility regarding vehicle controllers and easy support for lane changing, obstacle avoidance, and guided driving compared to an end-to-end approach.

3. Simulator Overview

Simulator capabilities. UE4Sim is built on top of Epic Game's Unreal Engine 4 and expands upon our previous work in [33], which primarily focused on UAV simulation. As recognized by others ([12; 47]), modern game engine architecture allows real-time rendering of not only RGB images, but can also with minor effort be re-tasked to output pixel-level segmentation, bounding boxes, class labels, and depth. Multiple cameras can be setup within a scene, attached to actors, and programmatically moved at each rendering frame. This capability allows additional synthetic data to be generated, such as simultaneous multi-view rendering, stereoscopy, structure-from-motion, and view augmentation. UE4 also has an advanced physics engine allowing the design and measurement of complex vehicle movement. Not only does the physics allow realistic simulation of moving objects but it can also be coupled with additional physics measurements at each frame. Finally, the broad support of flight joysticks, racing wheels, game consoles, and RGB-D sensors allows human control and input, including motion-capture, to be synchronized with the visual and physics rendered environment.

Simulated computer vision applications. In order to demonstrate these capabilities, we set up UE4Sim to generate sample synthetic data for twelve primary computer vision topics. In Fig. 1, we present a screenshot from each of the following applications: **(1)** object tracking; **(2)** pose estimation; **(3)** object detection (2D/3D); **(4)** action recognition; **(5)** autonomous navigation; **(6)** 3D reconstruction; **(7)** crowd understand-

ing; **(8)** urban scene understanding; **(9)** indoor scene understanding; **(10)** multi-agent collaboration; **(11)** human training; and **(12)** aerial surveying. In this paper, we present the full implementation and experiments on two of these CV applications: object tracking and autonomous navigation. We are releasing the full UE4Sim implementation of these applications, as well as, the general interface between UE4Sim and third party software, so as to facilitate its use in the community. In general, we believe that our simulator can provide computer vision researchers a rich environment for training and evaluating their methods across a diverse set of important applications.

Unique contributions of UE4Sim to UE4. Significant modifications to UE4 are made in order to enable the capabilities and applications addressed above. UE4 is re-tasked as a simulator and synthetic vision generator by the creation of new blueprints (a UE4 visual scripting language) and at a lower level bespoke C++ classes. UE4 is fully open source allowing us to exploit the full API code base to accomplish specific vision tasks that may never have been intended by UE4 creators. Unique contributions of UE4Sim include: a full Python, C++, and Matlab Socket Interface (TCP/UDP), physics-based waypoint navigation for cars and UAVs, PID controllers, flight and tracking controllers for UAVs, multi-object logging and replay system, synthetic visual data augmentation system, and an outdoor world generator with external drag-drop graphical user interface.

UE4 provides a marketplace in which users can contribute and sell assets to the community. We purchased a broad selection of assets and modified them to work specifically with UE4Sim. For example, the variety of cars in our simulated environment come from a set of purchased asset packs. On the other hand, the UAV used for tracking is based on a Solidworks model designed and produced by the authors to replicate a real-world UAV used to compete in a UAV challenge (refer to Fig. 2 for a rendering of this UAV).

Figure 2: The tracking UAV rendered inside UE4Sim. The UAV is equipped with gimbaled landing gear allowing for both a stabilized camera and articulated legs for gripping of objects in multi-agent simulation tasks.

4. Tracking

4.1. Overview

One case study application of the UE4Sim simulator is the ability to generate datasets with free automatic groundtruth (see Fig. 3) and to evaluate state-of-the-art CV tracking algorithms "in-the-loop" under close to real-world conditions (see Fig. 4). In this section we demonstrate both of these capabilities. Specifically, we select five diverse state-of-the-art trackers for evaluation. These are SAMF [23], SRDCF [7], MEEM [55], C-COT [8] and MOSSE_{CA} [34]. We then perform an offline evaluation analogous to the popular tracking benchmark by [53], as is common in object tracking, but on automatically annotated sequences generated within the simulator. However, we go beyond and additionally perform an online evaluation where trackers are directly integrated with the simulator thereby controlling the UAV "on-the-fly". Preliminary results of this application were presented in [33].

The simulator provides a test bed in which vision-based trackers can be tested on realistic high-fidelity renderings, following physics-based moving targets, and evaluated using precise ground truth annotation. Here, tracking is conducted from a UAV with the target being a moving car. As compared to the initial work in [33], the UAV has been replaced with a new quad-copter design allowing more flight capabilities (*e.g.* perched landing, grabbing, package delivery, *etc.*), and improved physics simulation using substepping has been integrated. In addition, the environment and assets are now automatically spawned at game time according to a 2D map created in our new city generator. The interface with CV trackers has been redesigned and optimized allowing fast transfer of visual data and feedback through various means of communication (*e.g.* TCP, UDP or RAM disk). This allows easy integration of trackers written in a variety of programming languages. We have modified the chosen state-of-the-art trackers to seamlessly communicate with the simulator. Trackers that run in MATLAB can directly be evaluated within the online tracking benchmark.

4.1.1 UAV Physics Simulation and Control

Within UE4, the UAV is represented as a quadcopter with attached camera gimbal and gripper. A low-level flight controller maintains the UAV position and altitude within a physics based environment. Movement of the copter is updated per frame by UE4's physics simu-

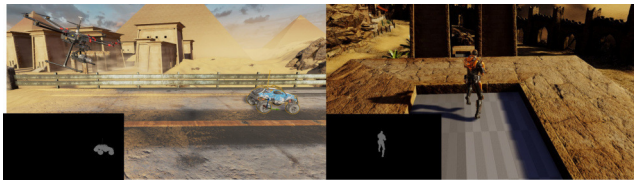


Figure 3: Two synthetic images in a desert scene generated from a virtual aerial camera within UE4Sim accompanied by their object-level segmentation masks.

Figure 4: An image of our UAV model during online tracking evaluation.

lator that accounts for Newtonian gravitational forces, input mass, size of the UAV, and linear/angular damping. Rotating the normal of the thrust vector along the central x- and y-axis of the copter and varying thrust enables the copter to move within the environment mimicking real-world flight. Similar to hardware-in-the-loop (HIL) approaches, the UAV control in UE4 utilizes several tuned PID controllers (written in C++ and accessed as a UE4 Blueprint function) and the calculations are done in substeps to decouple them from the frame rate.

Since we want to model a UAV in position hold, movement in the x and y directions are simulated by updating the required roll and pitch with a PID controller. The PID controller uses the difference between current and desired velocity in the x and y directions as error to be minimized. Similar to real-world tuning we experimentally adjust the controller weights within the simulator until we achieve a smooth response. Altitude of the UAV is maintained by an additional PID controller that adjusts thrust based on desired error between current altitude and desired altitude. Through this system, we are able to accurately simulate real-world flight of multi-rotors and control them by either a joystick or external input. The actual position of the UAV is kept unknown to the controller and trackers.

4.1.2 Extracting and Logging Flight Data

Attached to the UAV is a camera set at a 60 degree angle and located below the frame of the copter. At every frame, the UAV camera's viewport is stored in two textures (full rendered frame and custom depth mask), which can be accessed and retrieved quickly in MATLAB enabling the real-time evaluation of tracking within UE4Sim. Finally, at each frame, we log the current bounding box, position, orientation, and velocity of the tracked target and the UAV for use in the evaluation of the tracker.

4.1.3 MATLAB/C++ Integration

The trajectory of the vehicle is replayed from a log file to ensure equal conditions for all trackers. In our experiments, trackers are setup to read frames from a RAM disk. Alternatively, trackers can communicate through TCP or UDP with the simulator. The frames are output at 320×180 pixels in order to reduce latency. A script runs in MATLAB to initialize the current tracker with the initial bounding box sent by the simulator. The MATLAB script then continues to read subsequent frames and passes them to the current tracker. The output of the tracker, after processing a frame, is a bounding box, which is read by UE4 at every frame and used to calculate error between the center of the camera frame and the bounding box center. Trackers always get the most recent frame, so they drop/miss intermediate frames if their runtime is slower than the rate at which frames are being acquired. This inherently penalizes slow trackers just like in a real-world setting.

4.1.4 Visual Servoing

The onboard flight control simulator updates the UAV position to bring the tracked object back to the center of the camera field-of-view. This is done in real-time by calculating the error from the tracker's bounding box and its integration with two PID controllers. Since the camera system is mounted on a gimbal and the camera angle and altitude are held constant, only the vertical and horizontal offsets need to be calculated in the current video frame to properly reorient the UAV. The translational error in the camera frame is obtained by finding the difference between the current target's bounding box center and the center of the video frame. A fully-tuned PID controller for both the x and y dimensions receives this offset vector and calculates the proportional response of the copter movement to recenter the tracked object. The visual servoing technique employed in the simulator is robust and, with top performing trackers, it is able to follow targets across large

and diverse environments.

4.1.5 Qualitative Tracker Performance Evaluation

The introduction of automatic world generation allows us to fully control the environment and isolate specific tracking attributes, carry out multiple controlled experiments, and generate very diverse annotated datasets on-the-fly. Unlike real-world scenarios where the UAV and target location are not exactly known (*e.g.* error of 5-10m due to inaccuracies of GPS), we can quantitatively compare position, orientation, and velocity of the UAV at each time-step to understand the impact of the tracker on flight dynamics.

The simulator also enables new approaches for online performance measurement (see Fig. 5). For evaluation, we propose several approaches to measure tracker performance that can only be accomplished using our UE4Sim simulator: (1) the impact of a dynamic frame rate (different camera frame rates can be simulated and trackers are fed frames at most at the rate of computation), (2) trajectory error between the tracked target and the UAV, (3) trajectory error between a UAV controlled by ground-truth and a UAV controlled by a tracking algorithm, (4) long-term tracking within a controlled environment where attribute influence can be varied and clearly measured, and (5) recorded logs for each tracker are replayed and the path of each tracker is drawn on the 2D world map or animated in the 3D world.

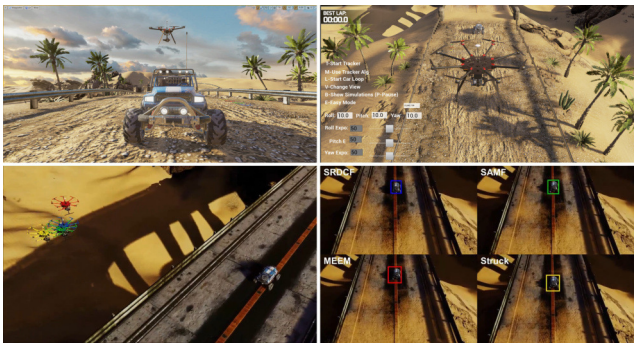


Figure 5: *Top:* Third person view of one environment in the simulator. *Bottom:* Four UAVs are controlled by different trackers indicated by the different colors.

4.2. Offline Evaluation

In order to demonstrate the dataset generation capabilities, we automatically generate a synthetic dataset with images captured from an UAV while following a car on the ground. The UAV uses the ground truth bounding box as input to the PID controller. We capture a total of 5 sequences from two maps with the car moving at 3 different speed levels (low: $4\,\mathrm{m\,s^{-1}}$, medium: $6\,\mathrm{m\,s^{-1}}$, high: $8\,\mathrm{m\,s^{-1}}$). The shortest sequence is 3.300 frames and the longest sequence is 12.700 frames in length. In order to benchmark tracking algorithms, we follow the classical evaluation strategy of the popular online tracking benchmark by [53]. We evaluate the tracking performance using two measures: precision and success. Precision is measured as the distance between the centers of a tracker bounding box (bb_tr) and the corresponding ground truth bounding box (bb_gt). The precision plot shows the percentage of tracker bounding boxes within a given threshold distance in pixels of the ground truth. To rank trackers according to precision, we use the area under the curve (AUC) measure, which is also used in [53]. Success is measured as the intersection over union of pixels in box bb_tr and those in bb_gt. The success plot shows the percentage of tracker bounding boxes, whose overlap score is larger than a given threshold. Similar to precision, we rank trackers according to success using the area under the curve (AUC) measure. We only perform a one-pass evaluation (OPE) in this paper.

The results in Fig. 6 show that MEEM performs best in terms of both precision and success while running at over 30fps. However, note that evaluation was performed on a powerful workstation and at a low image resolution. C-COT has comparable performance but runs at a significantly lower speed (less than 1fps). Surprisingly, $\mathrm{MOSSE_{CA}}$ which only used very simple features (only gray-scale pixel intensity) performs remarkably well, while running at over 200fps. We attribute this to its novel incorporation of context. Also note that at an error threshold of about 25 pixels which might still be acceptable for many applications, it is actually on par with MEEM and C-COT. SRDCF achieves similar performance as $\mathrm{MOSSE_{CA}}$ and is about twenty times slower. SAMF performs significantly worse than all other trackers in this evaluation.

Fig. 7 shows some qualitative results of the offline experiments. The images in the first row show part of a sequence in one setup, where the car is moving at high speed. SAMF already fails very early when the car drives quickly around a corner. The images in the second row show another critical moment of the same sequence, where the car goes out of view for a short period of time. Except for SAMF which already lost the target earlier, all trackers are able to track the target until the very end. Note how $\mathrm{MOSSE_{CA}}$ has drifted and is only tracking a small corner of the car by the end. The third row shows a sequence of

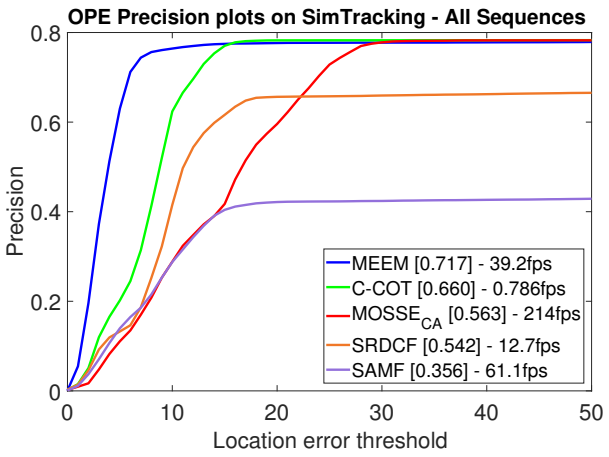


Figure 6: Offline evaluation: average overall performance in terms of precision and success

images from a much more difficult setup, where the car is moving at medium speed. There is heavy occlusion by a tree causing all trackers besides MOSSE_{CA} and C-COT to lose the car. The images in the fourth row show that MOSSE_{CA} has a much better lock on the car after this occlusion than C-COT. However, when the car gets fully occluded for several frames, no tracker is able to re-detect the car.

4.3. Online Evaluation

In this evaluation, the tracking algorithms communicate with the simulator. They are initialized with the first frame captured from the UAV and the corresponding ground truth bounding box. They then receive subsequent frames as input and produce a bounding box prediction as output, which is sent back to the simulator and serves as input to the PID controller of the

Figure 7: Offline evaluation: qualitative results

UAV for the purpose of navigation. Depending on the speed of the tracking algorithm, frames are dropped so that only the most recent frame is evaluated much like in a real system. This evaluation emphasizes the effect of a tracker's computational efficiency on its online performance and provides insight on how suitable it would be for real-world scenarios.

We first optimize the UAV visual servoing using the ground truth (GT) tracker, which has access to the exact position of the object from the simulator. Despite the absolute accuracy of the GT tracker, the flight mechanics of the UAV limit its ability to always keep the target centered, since it must compensate for gravity, air resistance, and inertia. After evaluating the performance of the UAV with GT, each tracker is run multiple times within the simulator with the same starting initialization bounding box.

Fig. 8 shows the trajectories of the tracking algorithms on each evaluated map. SAMF is only able to complete the easy map1 when the car is moving at low speed and fails consistently otherwise. The higher the speed, the earlier the failure occurs. SRDCF, MEEM and MOSSE_{CA} are all able to complete map1 at all speed levels. However, on map2, SRDCF fails quite early at both speeds. MEEM is able to complete about 75% of map2 at medium speed, but fails early at high speed. MOSSE_{CA} performs best and is the only tracker to complete map2 at medium speed. At high speed, it is able to complete about 75% of map2, again outperforming all other trackers by a margin.

C-COT, which won the VOT16 challenge, and is currently considered the best tracking algorithm, fails to impress in this evaluation. Just to initialize and process the first frame takes about 15 seconds by which time the target is long gone. To ensure fair comparison we keep the car stationary for more than 15 seconds to

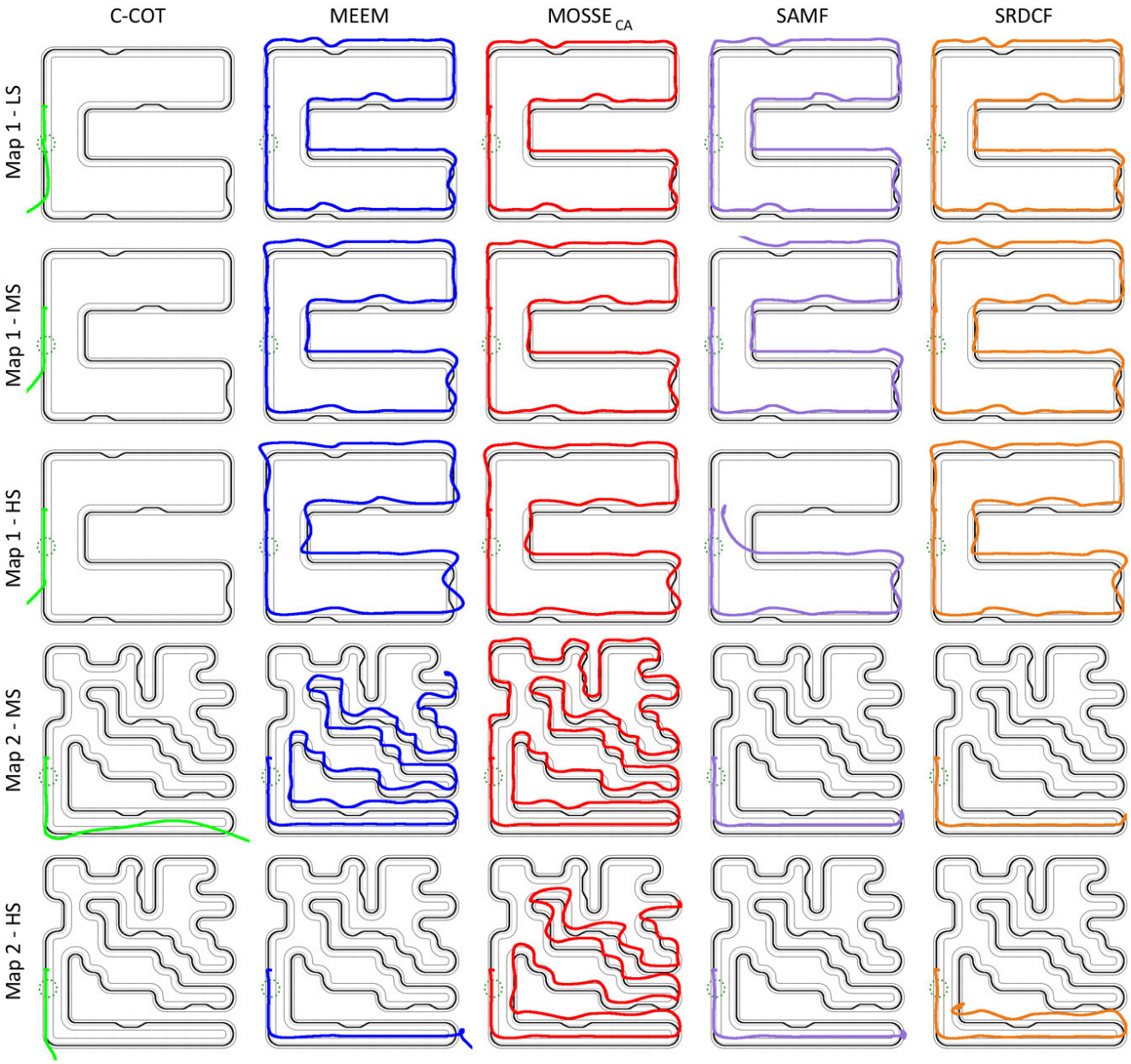


Figure 8: Qualitative results: trajectory of 5 different trackers in 5 experiments on two different maps. LS, MS, and HS denote low, medium, and high speed characterizing the tracked object (car).

provide plenty of time for initialization. However, after the car starts moving the target is lost very quickly in both maps and at all speeds, since C-COT only runs at less than 1fps resulting in very abrupt UAV motions and unstable behavior.

4.4. Discussion

MOSSE_{CA} by [34] achieves similar accuracy as the top state-of-the-art trackers, while running an order of magnitude faster. It uses a much simpler algorithm and features. This allows for deployment on a real system with limited computational resources. Further it allows processing images with larger resolution and hence more details which is especially important in the case of UAV tracking where objects are often very low resolution as showed by [33]. Lastly it permits controlling the UAV at a much faster rate if frames can be captured at higher frame rates. This also has the side effect to simplify the tracking problem since the target moves less between frames.

5. Autonomous Driving

5.1. Overview

In the second case study application, we present a novel deep learning based approach towards autonomous driving in which we divide the driving task into two sub-tasks: pathway estimation and vehicle control. We train a deep neural network (DNN) to predict waypoints ahead of the car and build an algorithmic controller on top for steering and throttle control. Compared to learning the vehicle controls end-to-end, our approach has several advantages: our approach only requires auto-generated training data (waypoints), whereas a human driver would be required to generate extensive data for a supervised end-to-end approach. Additionally, our waypoint approach is more flexible, since it generalizes across different cars, even beyond the one used in training. As such, the car can be tuned or even replaced with a different vehicle without retraining the network. Finally, tasks such as lane changing (Sec. 5.5.3), visual obstacle avoidance (Sec. 5.5.4) or guided driving (Sec. 5.5.5) become quite straight-forward and easy to implement with our approach. We found that augmenting our data with respect to viewing direction is crucial for achieving high performance in these tasks. Our waypoint-based method automatically assigns real ground truth data to augmented view images as well, whereas the vehicle control outputs would have to be modified in a non-trivial way, which might require manual annotation.

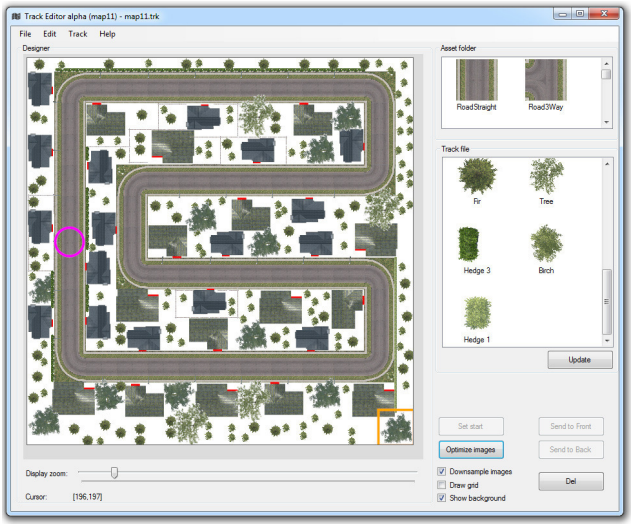
To generate virtual driving environments, we develop an external software tool, where maps can be designed from a 2D overhead view, and directly imported into our simulator. From there, we automatically generate synthetic training data for our waypoint DNN. We also use the environments to evaluate our driving approach in an online fashion. This is made possible by developing an interface that enables our simulator to communicate seamlessly with the deep learning framework (*i.e.* TensorFlow) and thus enables our model to control the car in real-time.

5.2. Data Acquisition

5.2.1 Generating Virtual Driving Environments

To automatically generate virtual driving environments, we develop an editor that can be used to build anything from small neighborhoods up to entire cities from an overhead view (see Fig. 9b). Users can simply manipulate standardized blocks of arbitrary sizes that represent objects such as road parts, trees or houses. They can also choose to generate the road network randomly. This process can easily generate a very diverse set of training and testing environments. The editor is fully compatible with our simulator, and the generated environments can be loaded directly within UE4Sim. Fig. 9a shows images taken from our simulator while driving within an urban environment constructed using our editor in Fig. 9b.

(a) Images captured from our simulator in driving mode, top: third-person view, bottom: first-person view.



(b) Our editor to create virtual driving environments. As shown in the preview, a variety of objects such as road blocks, houses or trees can be arranged on a grid.

Figure 9: Images showing two views of a car spawned in a map, which is constructed from an overhead view using our city editor (b)

5.2.2 Generating Synthetic Image Data

Since we are training a DNN to predict the course of the road, it is not necessary to collect human input (*i.e.* steering angle or use of accelerator/break). Thus, to synthetically generate image data, we automatically move the car through the road system, making sure it is placed on the right lane and properly oriented, *i.e.* at a tangent to the pathway. We render one image at every fixed distance that we move the car. However, we find that using this data alone is not sufficient to obtain a model that is able to make reasonable predictions once the car gets off the traffic lane (see Sec. 5.5.1). This observation is typical for sequential decision making processes that make use of imitation or reinforcement learning. Therefore, we augment this original data by introducing two sets of parameters: x-offsets that define how far we translate the car to the left or right on the viewing axis normal, and yaw-offsets that define angles we use to rotate the car around the normal to the ground. For each original image, we investigate using fixed sets of these offsets combined exhaustively, as well as, randomly sampling views from predefined ranges of the two offset parameters.

5.2.3 Generating Ground Truth Data

We describe the course of the road by a fixed number of waypoints that have an equal spacing. For each non-augmented view that is rendered, we choose 4 waypoints with a distance of 2 meters between each pair, so that we predict in a range of 2 to 8 meters. We then encode these 4 waypoints relative to the car position and orientation by projecting them onto the viewing axis. Fig. 10 illustrates the encoding method. We define a vertical offset that is measured as the distance between the car position and the projected point along the viewing axis (green bracket), and a horizontal offset that is defined as the distance between the original and projected point along the viewing axis normal (green line segment). For each augmentation of an original view, we use the same four waypoints and the same encoding strategy to represent the augmented view. As such, each view (original or augmented) will be described with a rendered image and labeled with 8 *ground truth* offsets using the aforementioned waypoint encoding.

We use a total of 16 driving environments (maps) for training the network (with a 12-4 split in training and validation) and an additional 4 for testing. The total road length is approximately 22.741 m. To study the effect of context on driving performance, we setup each map in two surroundings: a sandy desert (with road only) and an urban setting (with a variety of trees, parks, houses, street lanterns, and other objects). The details of our training and test set are summarized in

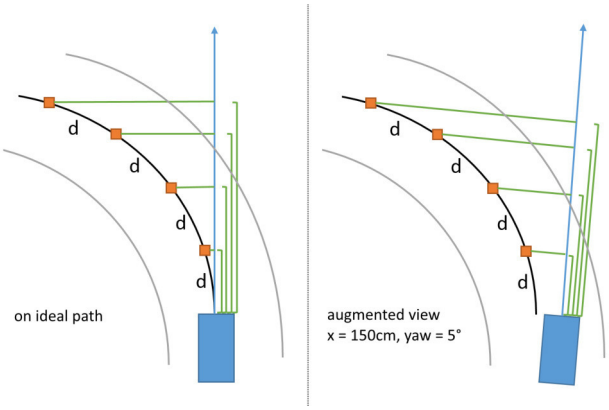


Figure 10: An illustration of our waypoint encoding. The car (blue, viewing direction denoted by blue arrow) is about to enter a left turn. 4 waypoints (orange) are chosen such that there is a constant distance d between them on the ideal path arc. Each waypoint is then encoded by a set of horizontal (green line segment) and vertical (green bracket) offsets.

	Training	**Validation**	**Testing**
Maps	12	4	4
Road length	13,678 m	4,086 m	4,977 m
Stepsize straights	80 cm	200 cm	-
Stepsize turns	20 cm	200 cm	-
X-offset range	$[-4\,\text{m}, 4\,\text{m}]$	$[-4\,\text{m}, 4\,\text{m}]$	-
Yaw-offset range	$[-30°, 30°]$	$[-30°, 30°]$	-
Random views	1	3	-
Total images	66,816	57,100	-

Figure 11: A description of our datasets and default sampling settings. We have two versions of each dataset, one in a desert and one in an urban setting, sharing the same road network.

Table 11.

5.3. DNN-Training

We choose the structure of our waypoint prediction network by running extensive experiments using a variety of architectures. We optimized for a small architecture that achieves high performance on the task in real-time. The structure of our best performing network is shown in Fig. 12. Notice that we optionally include one additional input (goal) that bypasses the convolutional layers. This is used to encode the desired direction at intersections for guided driving (see Sec. 5.5.5). The network is able to run at over 500 frames per second (fps) on an Nvidia Titan Xp when using a batch size of one (faster otherwise). We also expect it to be real-time capable on slower and/or embedded GPUs. We train our networking using a standard L2-

loss and the Adam optimization algorithm, setting the base learning rate to 5e−5. We also use early stopping when the validation error does not decrease anymore.

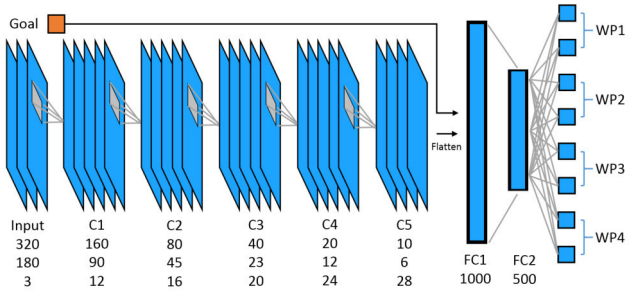


Figure 12: Our network architecture. We use a total of 7 layers, two of which are fully connected to predict two offsets for each waypoint from a single 320×180 resolution RGB-image. Optionally, we include one more value (goal) in the fully connected layer that encodes the direction the car should follow at intersections.

5.4. Vehicle Controller and Scoring

We use a simple approach to obtain steering angle and throttle from the waypoint predictions. Although there exist more sophisticated methods, we found this approach to work well in practice despite its simplicity. Based on the vertical and horizontal offset v and h of the first waypoint, we set the steering angle to $\theta = \arctan(\frac{h}{v})$. The throttle controller uses the horizontal offset of the last waypoint and sets the throttle based on its absolute value. For small values, throttle is high as there is a straight ahead. As the car moves closer to a turn, the absolute value increases and we lower throttle to slow down the car. Since the predictions of our network are very accurate, the trajectory that the car takes is smooth even without explicitly enforcing temporal smoothness in training or during evaluation. Of course, our approach can be easily used with different kinds of controllers that take factors like vehicle physics into account.

While there is no dedicated measurement for the controller, its performance is reflected by our measurements on the testing set. There, we run the whole system under real conditions in the simulator, whereby the car position is recorded and penalized for deviations from the ideal pathway. We use the Euclidean distance as the penalty term and average it over the course of all tracks. For reference, the width of the car is 2 m and the lane width is 4 m each, so that the car stays exactly at the edge of the right lane on both sides if $d = 1$ m. We denote the range within these bounds as the critical region. Besides measuring the average deviation, we also create cumulative histograms that denote what percentage of time the car stayed within a certain range. We use a bin size of 5 cm for the histogram.

5.5. Evaluation

5.5.1 Investigating the Impact of Augmentation.

We run a variety of experiments to investigate the impact of augmentation on our network's performance. Our results indicate that while using viewpoint augmentation is crucial, adding only a few additional views for each original frame is sufficient to achieve good performance. Without any augmentation, the system fails on every test track. Slight deviations from the lane center accumulate over time and without any augmentation, the DNN does not learn the ability to recover from the drift. We investigate two strategies for choosing the camera parameters in augmented frames: using a set of fixed offsets at every original frame, and performing random sampling. As described in Table 11, we obtain random views within an x-range of $[-4\,\text{m}, 4\,\text{m}]$ and a yaw-range of $[-30°, 30°]$, providing comprehensive coverage of the road. While adding more views did not lower performance, it did not increase it either. Therefore, we choose a random sampling model that uses just one augmented view. For fixed views, we test different sets of offsets that are subsequently combined exhaustively. We find that the best configuration only utilizes rotational (yaw) offsets of $[-30°, 30°]$. Results on the test set for both augmentation strategies are shown in Table 13.

	Random views	**Fixed views**
% in $[-25\,\text{cm}, 25\,\text{cm}]$	0.9550	0.8944
% in $[-50\,\text{cm}, 50\,\text{cm}]$	0.9966	0.9680
% in $[-1\,\text{m}, 1\,\text{m}]$	1.0000	0.9954
Avg deviation [cm]	7.1205	14.6129

Figure 13: Comparison of our best performing networks trained with random sampling and fixed view augmentation, respectively. While the fixed view model still achieves very good results, it is not on par with the random sampling one.

We find that networks trained on fixed offsets perform worse than the ones trained on randomly augmented data. However, using the best fixed offset configuration (only yaw-offsets with $[-30°, 30°]$), we still achieve very reasonable results. This is also the setting used in several related work, including [50] and probably [4] (two rotational offsets only are used in both papers, but the exact angles are not given in the latter). Our model is also able to navigate the car within the critical range of $[-1\,\text{m}, 1\,\text{m}]$. However, while it does

outperform all human drivers, its average deviation is more than twice as high as our random view model. Our random view model stays within a very close range of $[-25\,\text{cm}, 25\,\text{cm}]$ over 95% of the time, while it almost never leaves the range of $[-50\,\text{cm}, 50\,\text{cm}]$, compared respectively to 89% and 97% for the fixed view model. With an average deviation of just 7.12 cm or 14.61 cm, both models drive much more accurately than our best performing human test subject at 30.17 cm.

Despite training the fixed view model on 100% more synthetic views and 50% more data in total, it is outperformed by our random view model. We make the same observation for models trained on even more fixed views. Since random sampling is not feasible in the real-world, this shows another advantage of using a simulator to train a model for the given driving task.

5.5.2 Comparison to Human Performance

We compare the performance of our system to the driving capabilities of humans. For this, we connect a ThrustMaster Steering Wheel and Pedal Set and integrate it into our simulator. We then let three humans drive on the training maps (desert) as long as they wish. After this, we let them complete the testing maps (desert) and record results of their first and only try. We create cumulative histograms of their performance as described in Sec. 5.4. We compare these results to those of our best performing network in both desert and urban environments depicted by histograms in Fig. 14. Clearly, all three human subjects achieve similar performance. In approximately 93-97% of cases the human controlled car stays entirely within the lane edges ($\pm 1\,\text{m}$). While subjects 1 and 2 left the lane in the remaining instances, they never exceeded a distance of 140 cm. This is not the case for the third test subject, who goes completely off track in a turn after losing control of the car.

In comparison, our approach is clearly superior and yields much better results. Both histograms saturate very quickly and reach 100% at about 60 cm distance. Thus, DNN driving is much more accurate, and there is absolutely no instance where our network-driven approach navigates the car close to the lane edges. The results also suggest that our network is able to generalize to not only the given unseen test tracks, but also unseen environments not used in training, since the results on the highly altered urban environment are close to those on the desert environment.

5.5.3 Changing Lanes

Changing lanes is a fundamental capability a self-driving car needs to provide. It is essential in tasks such as navigation in multi-lane scenarios or executing

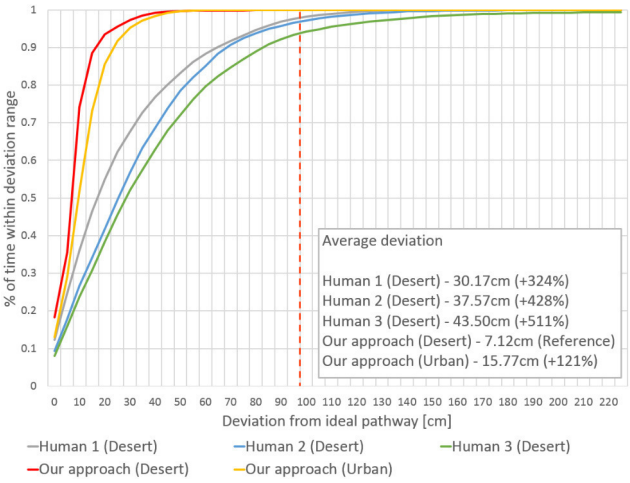
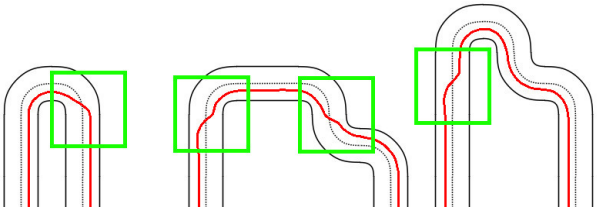


Figure 14: Cumulative histogram showing the deviation between ideal pathway and driven trajectory. The histogram shows the percent of time (y-axis) the car is located within the respective range (x-axis). We compare the performance of three human drivers to our system on the test tracks. Our system stays significantly closer to the ideal pathway than any human driver, and entirely avoids the critical zone (right of the dotted red line, which denotes the lane edges).

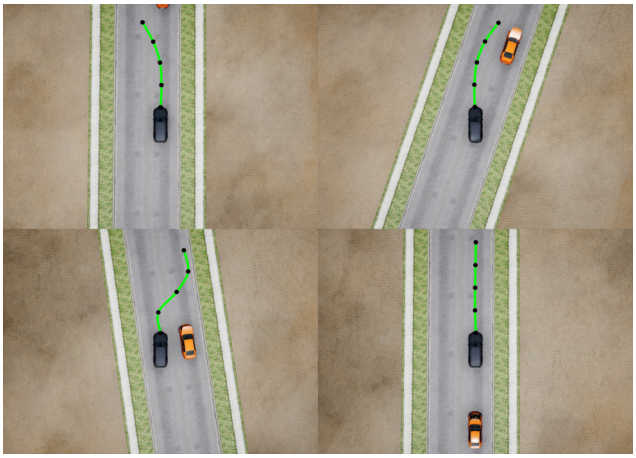
passing maneuvers. Our network that is trained to predict waypoints on the right lane already possesses the required capabilities to predict those on the left lane. To use them, we just flip the image before processing it in the network, and then flip the predicted horizontal offsets of each waypoint. In enabling and disabling this additional processing, the car seamlessly changes lanes accordingly. Fig. 15a shows some qualitative results of our procedure. The driven trajectory is shown in red from an overhead view. Notice how the system handles a variety of cases: lane changes on straights, in ordinary turns and also S-bends. In practice, an external controller is used to trigger a lane change for this purpose. Our procedure can easily be triggered by an external controller that relies on other sensory data, as is the case in many cars on the market today, which can determine if the desired lane is occupied. Within our simulator, it is also possible to simulate sensors, opening up interesting possibilities for more comprehensive future work in simulated environments.

5.5.4 Learning to Avoid Obstacles

We extend our driving approach to include obstacle avoidance. For this task, we propose two approaches. One approach is to just use sensors that trigger the lane changing procedure as necessary, as discussed in Sec. 5.5.3. The other (more flexible) approach is to



(a) Qualitative results of our lane change procedure when called in different situations. Parts of a road map are shown from a top-down view with the driven trajectory marked in red. The lane change was initiated within the green regions.



(b) Qualitative results for visual obstacle avoidance in our desert environment. We visualized the waypoint predictions in green. The image sequence shows how the car is approaching the obstacle, performing the lane change, planning to change back to the right lane and then driving normally behind the obstacle.

Figure 15: Qualitative results for lane changing and obstacle avoidance.

perceive the obstacle visually and model the waypoint predictions accordingly. In this section, we present the latter, since the former has already been described.

We extend our city map editor to support obstacles that can be placed in the world as any other block element. When the ground truth (waypoints) for a map are exported, they are moved to the left lane as the car comes closer to the obstacle. After the obstacle, the waypoints are manipulated to lead back to the right lane. We extend the editor to spawn different obstacles randomly on straights and thus provide a highly diverse set of avoidance situations. After rendering these images and generating the corresponding waypoints, we follow the standard procedure for training used in obstacle-free driving described before. Fig. 15b shows qualitative results in the form of an image sequence with waypoint predictions visualized in green. As it comes close to the obstacle, the car already predicts the waypoints to lead towards the other lane. It then changes back to the right lane after passing the obstacle and predicts the usual course of the road.

5.5.5 Guided Driving

We denote the task of controlling the car in situations of ambiguity (*i.e.* intersections) as guided driving. This is especially useful for tasks such as GPS-based navigation, as an external controller can easily generate the appropriate sequence of directions to reach the target location. To implement this capability, we make use of the goal input in our network architecture (see Fig. 12) that feeds directly into the multilayer perceptron. We encode the intent to go left as -1, go straight as 0 and go right as $+1$. We design five large maps with a variety of intersections devoted only for the guided driving task. When generating the ground truth files, we randomly decide which direction to take at intersections and make sure to fully exploit the road network of each map. We set the ground truth value of the goal input according to the direction taken if anything between the car and the furthest waypoint lies on an intersection. In other cases, we randomize the goal input as to make the network robust to changes of its value at non-intersections. We use the same configuration for training and the same random-sampling augmentation strategy. At test time, we change the goal input dynamically in the simulator. Fig. 16 shows an example at an intersection where we parked the car and just changed the value of the goal input. The respective waypoint predictions for left, straight, and right are shown in red, blue and green, respectively. We find that learned guided driving is highly accurate in practice. In fact, in a one-hour driving test, we did not notice any mistakes.

5.6. Discussion

We present a novel and modular deep learning based approach towards autonomous driving. By using the deep network for pathway estimation only (thus decoupling it from the underlying car controls), we show that tasks such as lane change, obstacle avoidance, and guided driving become straightforward and very simple to implement. Furthermore, changes regarding the vehicle or its behaviour can be applied easily on the controller side without changing the learned network. Our approach even works without any need for human-generated or hand-crafted training data (although manually collected data can be included if available), thus, avoiding the high cost and tedious nature of manually collecting training data. We demonstrate the effectiveness of our approach by measuring the performance on different diversely arranged environments

Figure 16: An example showing results of our guided driving method. The car is located at an intersection. The image in the top right corner shows the situation from above with a visualization of the waypoint predictions when the goal input is set to -1 (red/left), 0 (blue/straight) and 1 (green/right), respectively.

and maps, showing that it can outperform the capabilities of human drivers by far.

6. Conclusions and Future Work

In this paper, we present an end-to-end high fidelity simulator that can be used for an extensive set of applications ranging across various fields of computer vision and graphics. We demonstrate the versatility of the simulator for evaluating vision problems by studying two major applications within simulated environments, namely vision-based tracking and autonomous driving. To the best of our knowledge, this simulator is the first to provide, on both fronts, a complete real-time synthetic benchmark and evaluation system for the vision community. The simulator goes beyond providing just synthetic data, but comprises a full suite of tools to evaluate and explore new environmental conditions and difficult vision tasks that are not easily controlled or replicated in the real-world.

In the future, we aim to further investigate the transference of capabilities learned in synthetic worlds to the real-world. Moreover, since our developed simulator and its seamless interface to deep learning platforms are generic in nature and open-source, we expect that this combination will open up unique opportunities for the community to develop better networks and tracking algorithms, expand its reach to other fields of autonomous navigation, and to benefit other interesting AI tasks (*e.g.* those using reinforcement learning).

Acknowledgments. This work was supported by the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research through the VCC funding.

References

[1] O. Andersson, M. Wzorek, and P. Doherty. Deep learning quadcopter control via risk-aware active learning. In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI), 2017, San Francisco, February 4-9.*, 2017. Accepted.

[2] B. Babenko, M.-H. Yang, and S. Belongie. Visual Tracking with Online Multiple Instance Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1619–1632, Dec 2010.

[3] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013.

[4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[5] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 2722–2730, Washington, DC, USA, 2015. IEEE Computer Society.

[6] R. Collins, X. Zhou, and S. K. Teh. An open source tracking testbed and evaluation web site. In *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS 2005), January 2005*, January 2005.

[7] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg. Learning spatially regularized correlation filters for visual tracking. In *The IEEE International Conference on Computer Vision (ICCV)*, Dec 2015.

[8] M. Danelljan, A. Robinson, F. Shahbaz Khan, and M. Felsberg. *Beyond Correlation Filters: Learning Continuous Convolution Operators for Visual Tracking*, pages 472–488. Springer International Publishing, Cham, 2016.

[9] C. De Souza, A. Gaidon, Y. Cabon, and A. Lopez Pena. Procedural generation of videos to train deep action recognition networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[10] C. Fu, A. Carrio, M. Olivares-Mendez, R. Suarez-Fernandez, and P. Campoy. Robust real-time vision-based aircraft tracking from unmanned aerial vehicles. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 5441–5446, May 2014.

[11] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. *RotorS—A modular gazebo MAV simulator framework*, volume 625 of *Studies in Computational Intelligence*, pages 595–625. Springer, Cham, 2016.

[12] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2016.

[13] A. Gaszczak, T. P. Breckon, and J. Han. Real-time people and vehicle detection from UAV imagery. In

J. Röning, D. P. Casasent, and E. L. Hall, editors, *IST/SPIE Electronic Imaging*, volume 7878, pages 78780B–1–13. International Society for Optics and Photonics, January 2011.

[14] S. Ha and C. K. Liu. Iterative training of dynamic skills inspired by human coaching techniques. *ACM Trans. Graph.*, 34(1):1:1–1:11, Dec. 2014.

[15] P. Hamalainen, S. Eriksson, E. Tanskanen, V. Kyrki, and J. Lehtinen. Online motion synthesis using sequential monte carlo. *ACM Trans. Graph.*, 33(4):51:1–51:12, July 2014.

[16] P. Hamalainen, J. Rajamaki, and C. K. Liu. Online control of simulated humanoids using particle belief propagation. *ACM Trans. Graph.*, 34(4):81:1–81:13, July 2015.

[17] M. Hejrati and D. Ramanan. Analysis by synthesis: 3D object recognition by object reconstruction. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2449–2456, June 2014.

[18] E. Ju, J. Won, J. Lee, B. Choi, J. Noh, and M. G. Choi. Data-driven control of flapping flight. *ACM Trans. Graph.*, 32(5):151:1–151:12, Oct. 2013.

[19] A. Kendall, N. Salvapantula, and K. Stol. On-board object tracking control of a quadcopter with monocular vision. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 404–411, May 2014.

[20] D. K. Kim and T. Chen. Deep neural network for real-time autonomous indoor navigation. *CoRR*, abs/1511.04668, 2015.

[21] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1061–1068, New York, NY, USA, 2013. ACM.

[22] J. Koutník, J. Schmidhuber, and F. Gomez. *Online Evolution of Deep Convolutional Network for Vision-Based Reinforcement Learning*, pages 260–269. Springer International Publishing, Cham, 2014.

[23] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, L. Čehovin, G. Nebehay, T. Vojíř, G. Fernandez, A. Lukežič, A. Dimitriev, et al. The visual object tracking vot2014 challenge results. In *Computer Vision-ECCV 2014 Workshops*, pages 191–217. Springer, 2014.

[24] A. Lerer, S. Gross, and R. Fergus. Learning Physical Intuition of Block Towers by Example, 2016. arXiv:1603.01312v1.

[25] A. Li, M. Lin, Y. Wu, M.-H. Yang, and S. Yan. NUS-PRO: A new visual tracking challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):335–349, Feb 2016.

[26] P. Liang, E. Blasch, and H. Ling. Encoding color information for visual tracking: Algorithms and benchmark. *IEEE Transactions on Image Processing*, 24(12):5630–5644, 2015.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ICLR*, abs/1509.02971, 2016.

[28] H. Lim and S. N. Sinha. Monocular localization of a moving person onboard a quadrotor mav. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2182–2189, May 2015.

[29] J. Marín, D. Vázquez, D. Gerónimo, and A. M. López. Learning appearance in virtual scenarios for pedestrian detection. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 137–144, June 2010.

[30] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[31] Y. Movshovitz-Attias, Y. Sheikh, V. Naresh Boddeti, and Z. Wei. 3D pose-by-detection of vehicles via discriminatively reduced ensembles of correlation filters. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[32] M. Mueller, G. Sharma, N. Smith, and B. Ghanem. Persistent aerial tracking system for UAVs. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference*, October 2016.

[33] M. Mueller, N. Smith, and B. Ghanem. *A Benchmark and Simulator for UAV Tracking*, pages 445–461. Springer International Publishing, Cham, 2016.

[34] M. Mueller, N. Smith, and B. Ghanem. Context-aware correlation filter tracking. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[35] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun. Off-road obstacle avoidance through end-to-end learning. In Y. Weiss, P. B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 739–746. MIT Press, 2006.

[36] T. Naseer, J. Sturm, and D. Cremers. Followme: Person following and gesture recognition with a quadrocopter. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 624–630, Nov 2013.

[37] A. Nussberger, H. Grabner, and L. Van Gool. Aerial object tracking from an airborne platform. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 1284–1293, May 2014.

[38] J. Papon and M. Schoeler. Semantic pose using deep networks trained on synthetic RGB-D. *CoRR*, abs/1508.00835, 2015.

[39] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3D geometry to deformable part models. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3362–3369, June 2012.

[40] J. Pestana, J. Sanchez-Lopez, P. Campoy, and S. Saripalli. Vision based GPS-denied object tracking and following for unmanned aerial vehicles. In *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, pages 1–6, Oct 2013.

[41] T. Pollard and M. Antone. Detecting and tracking all moving objects in wide-area aerial video. In *Computer Vision and Pattern Recognition Workshops*

(CVPRW), 2012 IEEE Computer Society Conference on, pages 15–22, June 2012.

[42] D. A. Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. Morgan-Kaufmann, 1989.

[43] J. Portmann, S. Lynen, M. Chli, and R. Siegwart. People detection and tracking from aerial thermal views. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1794–1800, May 2014.

[44] Y. A. Prabowo, B. R. Trilaksono, and F. R. Triputra. Hardware in-the-loop simulation for visual servoing of fixed wing UAV. In *Electrical Engineering and Informatics (ICEEI), 2015 International Conference on*, pages 247–252, Aug 2015.

[45] J. Prokaj and G. Medioni. Persistent tracking for wide area aerial surveillance. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1186–1193, June 2014.

[46] A. Qadir, J. Neubert, W. Semke, and R. Schultz. *On-Board Visual Tracking With Unmanned Aircraft System (UAS)*, chapter On-Board Visual Tracking With Unmanned Aircraft System (UAS). Infotech@Aerospace Conferences. American Institute of Aeronautics and Astronautics, Mar 2011. 0.

[47] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. *Playing for Data: Ground Truth from Computer Games*, pages 102–118. Springer International Publishing, Cham, 2016.

[48] U. Shah, R. Khawad, and K. M. Krishna. Deepfly: Towards complete autonomous navigation of MAVs with monocular camera. In *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*, ICVGIP '16, pages 59:1–59:8, New York, NY, USA, 2016. ACM.

[49] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1442–1468, July 2014.

[50] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield. Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness. *ArXiv e-prints*, May 2017.

[51] J. Tan, Y. Gu, C. K. Liu, and G. Turk. Learning bicycle stunts. *ACM Trans. Graph.*, 33(4):50:1–50:12, July 2014.

[52] B. R. Trilaksono, R. Triadhitama, W. Adiprawita, A. Wibowo, and A. Sreenatha. Hardware-in-the-loop simulation for visual target tracking of octorotor UAV. *Aircraft Engineering and Aerospace Technology*, 83(6):407–419, 2011.

[53] Y. Wu, J. Lim, and M.-H. Yang. Online Object Tracking: A Benchmark. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2411–2418. IEEE, June 2013.

[54] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié, C. Guionneau, and R. Coulom. TORCS, the open racing car simulator. `http://www.torcs.org`, 2014.

[55] J. Zhang, S. Ma, and S. Sclaroff. MEEM: robust tracking via multiple experts using entropy minimization. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2014.