replicAnt - generating annotated images of animals in complex environments with Unreal Engine

Fabian Plum[1*], René Bulla[2], Hendrik Beck[1], Natalie Imirzian[1], David Labonte[1*]

[1]Department of Bioengineering, Imperial College London, London, United Kingdom.
[2]The Pocket Dimension, Muenchen, Germany.

*Corresponding author(s). E-mail(s): fabian.plum18@imperial.ac.uk; d.labonte@imperial.ac.uk;
Contributing authors: renebullatv@gmail.com; h.beck20@imperial.ac.uk; n.imirzian@imperial.ac.uk;

Abstract

Deep learning-based computer vision methods are transforming animal behavioural research. Transfer learning has enabled work in non-model species, but still requires hand-annotation of example footage, and is only performant in well-defined conditions. To overcome these limitations, we created *replicAnt*, a configurable pipeline implemented in Unreal Engine 5 and Python, designed to generate large and variable training datasets on consumer-grade hardware instead. *replicAnt* places 3D animal models into complex, procedurally generated environments, from which automatically annotated images can be exported. We demonstrate that synthetic data generated with *replicAnt* can significantly reduce the hand-annotation required to achieve benchmark performance in common applications such as animal detection, tracking, pose-estimation, and semantic segmentation; and that it increases the subject-specificity and domain-invariance of the trained networks, so conferring robustness. In some applications, *replicAnt* may even remove the need for hand-annotation altogether. It thus represents a significant step towards porting deep learning-based computer vision tools to the field.

Keywords: ethology, deep learning, detection, tracking, pose-estimation, semantic segmentation

1 Introduction

Enabled by the continued reduction in cost of computational hardware and breakthroughs in deep neural network architectures and training paradigms, Data-driven deep learning approaches now represent the state-of-the-art in almost all computer vision applications[1, 2]. This success has been achieved in discriminative applications such as classification[3], detection[4], pose-estimation[5], and semantic segmentation[6], as much as in generative applications, as demonstrated by recent advancements in diffusion networks which can create stylised and near photo-realistic images from text prompts[7, 8]. Both discriminative and generative approaches have in common that they primarily involve supervised learning, which, to an extent, resembles high dimensional interpolation: achieving generalisation is practically synonymous with ensuring that inputs at training time reasonably resemble those encountered at inference time. As an illustrative example, successful detection requires that instances of the target class are identified regardless of image context and subject appearance[4]; the ideal detector is subject-specific, but domain-invariant. Large, curated and annotated datasets — such as those provided by ImageNet[9], COCO[10], or CiFAR[11] — are indispensable in this process, as they provide a basis for learnable real-world principles, and complex testing grounds.

A prime area of application for the emerging machine learning toolset is animal behavioural research[12–18], where it promises to reduce time costs, increase statistical power, and minimise potential for human bias; machine learning may altogether revolutionise what is possible in ethology[13, 18], and its intersection with neuroscience[16, 19, 20], morphology[21], locomotion[13, 18, 22], and conservation[23]. Despite the divergence in the questions tackled, all applications in these research areas have in common the need for annotated training data. Unfortunately, datasets of a size and quality required to achieve robust domain-invariant inference are rarely available, and – apart from a few model species such as mice or *Drosophila* – the effort required to curate them often outweighs the immediate benefit of the enabled automation. Transfer-learning – i. e., pre-training (parts of) a network on a separate, much larger, dataset, and refining the network on a small number of hand-annotated images – is a strategy which has been implemented with great success in markerless animal pose-estimation[16, 17, 20]. However, the price paid for the substantial reduction in the necessary amount of hand-annotation is that the resulting networks are typically only performant under stereotyped conditions, and frequently require extensive input pre-processing. Even minute deviations from the refinement data – for example in form of partial occlusion or changes in specimen appearance, lighting, background, perspective, or camera type – can result in a substantial drop in network performance. As a result, transfer-learning strategies have remained limited to well-controlled laboratory conditions, and are typically unsuitable for footage from the gold standard of behavioural studies – field experiments. Some of these generalisation issues can be addressed with data augmentation, i. e. the application of image perturbations with the aim to alter image appearance while retaining its meaning and label[4, 24, 25]. For example, by changing the rotation, scale, hue, and resolution of an image, its contents would still remain identifiable. However, even extensive augmentation pales in

its efficacy in comparison to using larger and more varied datasets instead[4, 24, 25].

In robotic[26–28], human[29–32], and automated driving[33–35] applications, annotated datasets comprising billions of images can now be produced "synthetically", i. e. through simulation with a computer. By placing 3D models in simulated environments, variable and annotated datasets can be generated at scale, and at a fraction of the cost and time required for hand-annotation of real images[31, 32, 34, 36]. The use of synthetic data is particularly attractive where annotated real datasets are practically absent or only of insufficient size, as is the case for almost all non-human animal studies[22, 37–39]. However, for all its conceptual attractiveness, using synthetic data is not without problems: the simulation must bridge the "simulation-reality gap", i. e. the synthetic image must be comparable in appearance to real images; as before, the key challenge is that the training data must represent a superset of the inputs received at inference time. As an illustrative example, Arent et al. [22] modelled Indian stick insects as a rigid body consisting of lines to improve the performance of a DeepLabCut[20] pose estimator. Such simplified geometric approaches can improve performance, but remain restricted to stereotyped recording settings, simple animal morphology, and a single output data type. Comprehensive and generalisable approaches which utilise more realistic animal representations, handle large digital animal populations, can create highly variable environments, and provide options for complex annotation, remain absent.

Here, we address this gap and present a synthetic dataset generator, *replicAnt*, implemented in Unreal Engine 5, a 3D computer graphics game engine, and Python. *replicAnt* can be used to simulate the appearance of animals in complex, procedurally generated environments with all but a few clicks of a mouse. Leveraging recent advancements in photogrammetry, real-time ray tracing, and high-resolution mesh handling, *replicAnt* runs on consumer-grade computational hardware, automatically produces rich image annotations, and can simulate virtually any recording conditions, including variations in camera model and perspective; individual number, size, pose, and colouration; scene lighting; image resolution and magnification; and environment appearance. We demonstrate the versatility and utility of *replicAnt* by using the synthetic data it generates to train deep neural networks for automatic inference in four common animal applications: (1) detection - localising animals in an image; (2) tracking—retaining the identity of animal detections across continuous frames; (3) markerless pose-estimation—extracting the coordinates of user-defined body landmarks; and (4) semantic and instance segmentation—determining which areas of an image correspond to an animal on a pixel level.

2 Results

2.1 *replicAnt*

replicAnt uses 3D models of animals to produce a user-defined number of annotated images. It is designed to generate large and variable datasets involving hundreds of animals with minimal user effort; due to the rich and automated annotation, a

single synthetic dataset can then be used to train a variety of deep neural networks. *replicAnt* requires: 3D models of the study organism(s); the installation of a pre-configured yet editable Unreal Engine project; and custom-written data parsers, used to translate the generated data into formats compatible with the deep learning-based computer vision system(s) of choice (see Fig. 1).

replicAnt is agnostic to the origin of the subject 3D model(s) used as input. Throughout this work, we use high resolution 3D models produced with the open-source photogrammetry platform *scAnt* (Fig. 1 a) [40]; but we also demonstrate that simpler hand-sculpted models can suffice for some applications. In general, the higher the 3D model fidelity, the higher the application flexibility.

Depending on their origin, 3D models may need to be cleaned, and—if the randomised pose variation feature of *replicAnt* is to be used—virtual bones and joints need to be assigned, and their range of movement defined (see Fig. 1 c, Methods 4.1.3 and the *replicAnt* GitHub https://github.com/evo-biomech/replicAnt). In this paper, we focus on insects, first because of personal predilection, and second because an exoskeleton avoids the need to simulate the complex soft tissue deformation associated with postural changes in animals with endoskeletons. However, powerful approaches to create photo-realistic models of vertebrates exist[41–43], and *replicAnt* is not limited to arthropod models (or even just animals, for what it is worth). The cleaned and rigged model is imported into a pre-configured Unreal Engine 5 project, where a simplified collision mesh is computed to enable interactions with objects inside the simulated world (Fig. 1 d).

Next, a customisable digital population is generated by simulating multiple instances of the original subject model. Variation between subject instances is achieved through simple appearance modifications, such as changes in brightness, contrast, hue, saturation, and scale. The range of these modulations can be adjusted through a simple user interface, and custom modifications can be added. Subjects are later sampled at random from this population, and placed into procedurally generated environments, from which annotated images are extracted.

Each scene is generated in a hierarchical process, structured into five customisable levels to maximise computational efficiency; changes in lower-level hierarchical elements influence higher level elements (Fig. 1 e). At the lowest level of scene hierarchy sits a ground plane with random topology. At the second level, this ground plane is populated with 3D assets; polygon meshes of objects such as plants, rocks, and common household items, all of random size. Assets are drawn from a curated library, and placed by a configurable number of asset scatterers. At the third level, the ground and each asset are assigned Physically Based Rendering materials, generated by blending randomly generated patterns with a curated texture library. Large material maps, or decals, are generated and wrapped around the ground plane and all assets to achieve a cohesive scene appearance. At the fourth level, a configurable number of subjects from the model population are placed at random locations, and their pose

is adjusted via inverse kinematics, such that they can interact with the surrounding meshes. At the fifth and highest hierarchical level, scene lighting is introduced in form of a configurable number of coloured light sources and High Dynamic Range Images (HDRIs), and a virtual camera with randomisable extrinsics, intrinsics and post processing parameters is placed; the scene generation is now complete (Fig. 1 f).

Using the virtual camera, "image passes" are exported from each scene iteration. Each pass encodes different information (Fig. 1 g-j), for example the optical image render itself, or depth information (for details, see Methods). User-defined passes can be added as required. Each image pass set is supplemented by a data file which contains configurable annotations, for example subject bounding boxes, 2D and 3D key point coordinates, class labels, or camera intrinsics and extrinsics. The combination of image passes and data files constitute synthetic data which can be used to train deep neural networks for various computer vision tasks(Fig. 1 k-n).

The entire process, including the generation of a user-specified number of scene iterations, image pass rendering, and data file writing, is fully automated, but leaves open plenty of opportunity to introduce variation with minimal effort. The pre-configured Unreal project, detailed documentation, and additional resources are available from https://github.com/evo-biomech/replicAnt.

2.2 Applications

In order to demonstrate that the synthetic data generated by *replicAnt* is of sufficient quality to power applications in animal behavioural research, we used it to train various popular deep learning networks for animal detection, tracking, pose-estimation, and semantic and instance segmentation. The performance of these networks was then evaluated on dedicated example datasets. Unless stated otherwise, all synthetic data used for training was generated using *replicAnt*'s default settings (see Methods 4.4 and GitHub for details). We will now show that *replicAnt* does not only significantly improve the trained networks' ability to generalise to unseen conditions but, in some cases, removes the need for hand-annotation altogether, and in others, it may present the only option to generate datasets large enough to train robust and performant networks in reasonable time.

2.2.1 Detection

A digital population of *Atta vollenweideri* leafcutter ants (Forel 1893), comprising 100 simulated individuals, was created using 3D models of a minor, media, and major worker, all generated with *scAnt*[40] (Fig. 2 b, see Methods 4.6 for details). This population formed the basis for two synthetic datasets, each encompassing 10,000 annotated images with a resolution of 1024×1024 px: one used all three 3D models ("group"), and one using only the largest model ("single"). Furthermore, to investigate the influence of synthetic dataset size on inference performance, networks were trained on 1% ("small"), 10% ("medium"), and 100% ("large") of the "group"

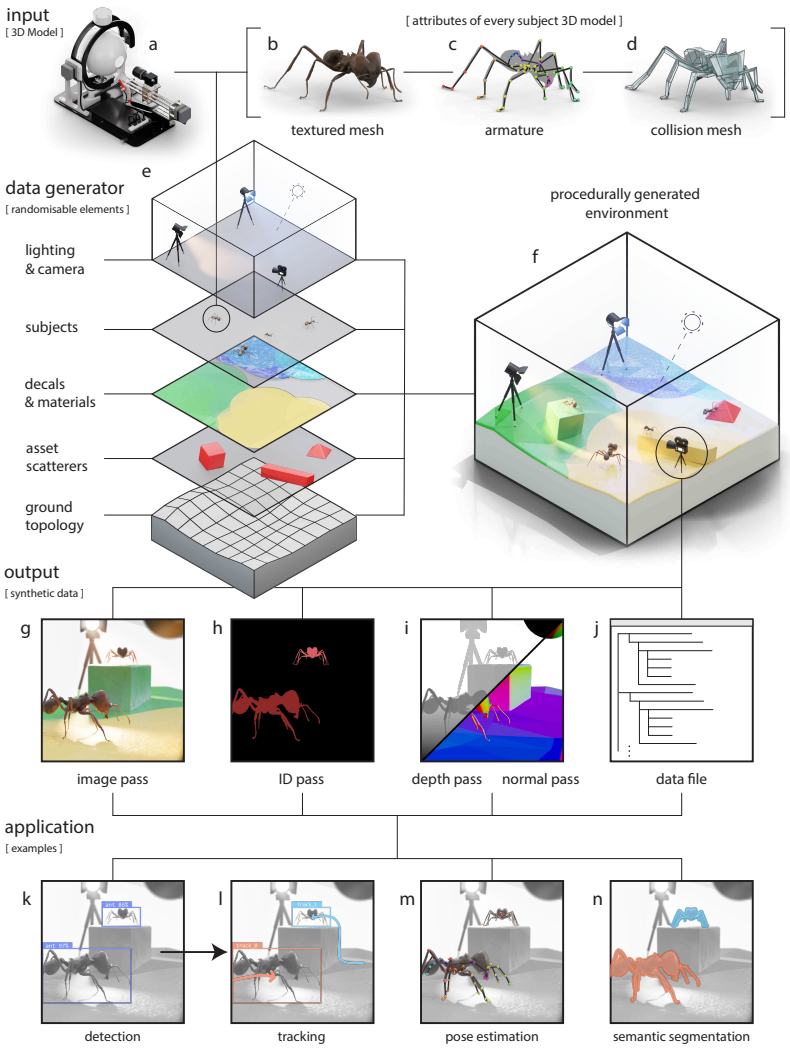

Fig. 1 ***replicAnt* is a toolbox designed to procedurally generate and automatically annotate image samples from 3D animal models. The combination of images and annotations constitutes "synthetic data" which can be used in a wide range of deep learning-based computer vision applications.** (a) *replicAnt* requires digital 3D subject models; all but one subject model used in this work were generated with the open-source photogrammetry platform *scAnt*[40]. Each model comprises (b) a textured mesh, (c) an armature, defined by virtual bones and joints, to provide control over animal pose, and (d) a low-polygonal collision mesh to enable interaction of the model with objects in its environment. (e) 3D models are placed within environments procedurally generated with a pre-configured yet customisable project in Unreal Engine 5. (f) Every scene consists of the same core elements, configurable via dedicated randomisation routines to maximise variability in the generated data. 3D assets are scattered on a ground of varying topology; layered materials, decals, and light sources introduce further sources of variability across scene iterations (see examples in Figures 2,3,4,5,6). From each scene, we generate (g) image, (h) ID, (i) depth, and normal passes, accompanied by (j) a human-readable data file which contains annotations and key information on image content (see methods for details). Synthetic datasets generated with *replicAnt* can then be parsed to train networks for a wide range of computer vision applications in animal behavioural research, including (k) detection, (l) tracking, (m) 2D and 3D pose-estimation, (n) and semantic segmentation.

dataset. Dataset generation took about ten hours each for the full "group" and "single" datasets on a consumer-grade laptop (6 Core 4 GHz CPU, 16 GB RAM, RTX 2070 Super).

The generated synthetic datasets were then used to train a commonly used object detector, YOLOv4[4], subsequently tested on laboratory recordings of a crowded foraging trail (Fig. 2 d). Foraging trails of *Atta* ants present an ideal example for complex detection tasks as individuals vary in size, trails are highly cluttered, and partial as well as full occlusions occur frequently. In order to introduce variation in scene appearance, akin to what may be expected in field conditions, scene lighting, exposure time, camera magnification and foraging trail background were altered systemically, yielding five different recording scenarios (Fig. 2 e). From these videos, 1000 frames with between 36 to 103 individuals were hand-annotated using BlenderMotionExport[44]. For comparison, we also trained detectors using 5000 of these hand-annotated images, using image combinations from the different recording scenarios. Five-fold cross validation, with 80/20 splits between training and validation data, was used for all training (see methods for details on test data and training schedules).

In general, detectors performed best on within-domain data, where they achieved close to perfect performance (Fig. 2 g). The notable exception to this rule were close-up recordings, where the detector trained on synthetic data outperformed the within-domain network. However, the performance of detectors trained with real data dropped notably when they were used for inference on unseen recordings, despite the similarity in perspective (Fig. 2 g-h). In sharp contrast, the detectors trained with synthetic data retained a robust and consistent performance throughout (Fig. 2 g). To quantify this difference, the Average Precision (AP) was averaged, so yielding a mean Average Precision across all unseen test cases (mAP, see equation 2). The detectors trained exclusively with synthetic data achieved an mAP of 0.913 ± 0.0079, both higher and less variable than all detectors trained with any of the five real subdatasets (Fig. 2 g). For comparison, the best real data detector was trained on noisy images, and achieved a mAP of 0.878 ± 0.0258. Networks trained exclusively on synthetic data converged more slowly and exhibited an overall higher loss during training compared to any set of real training images, indicating a higher level of complexity of the generated images (Fig. 2 f). These results indicate that the large volume and variability of synthetic data substantially increases robustness of detections, and suggests that supplementing training datasets with synthetically generated samples may be a suitable strategy to significantly reduce the hand-annotation to achieve benchmark performance and to allow networks to better generalise to unseen conditions. To test this idea, detectors were trained on "mixed" datasets, containing both real and synthetically generated images (see methods for details). Networks trained with a 10,000/100 synthetic/real split ("sb1") achieved an mAP of 0.9501 ± 0.014, close to the benchmark performance (Fig. 2 g). In order to confirm that synthetic data enables networks to recognise ants specifically and not just objects with similar appearance, we tested detectors trained with 3D models of desert termites (*Gnathamitermes* sp., see below), which resulted in a negligible mAP of 0.007 ± 0.005 (Fig. 2 g).

Next, we sought to demonstrate that high model fidelity is not required for detection tasks which typically involve low magnification recordings. Instead, even simpler hand-sculptured models can be used to train performant networks, powered by the large volume and variability of training images that can be generated with *replicAnt*. To this end, we procured a test dataset of 1000 consecutive frames of 49 freely moving desert termites, *Gnathamitermes* sp., recorded in the field and hand-annotated using BlenderMotionExport[44] to provide a simple benchmark (Fig. 3 and methods for details). Two 3D models, one of a worker and one of a soldier, were hand-sculpted from reference images using Blender v3.1. A YOLOv4[4] network, trained on a dataset of 10,000 synthetically generated images with a resolution of 1024×1024 px (Fig. 3 a-c. SI A4 for details), achieved an AP of 0.956 ± 0.001 on the annotated recordings, and produced accurate detections in qualitative test cases (Fig. 3 e-f).

2.2.2 Multi-animal tracking

Sufficiently precise detectors can in principle be used to build simple, yet robust and performant trackers. To facilitate the use of *replicAnt*-trained detectors in tracking applications, we introduce *OmniTrax*[45], an open-source Blender add-on. *OmniTrax* allows users to conduct interactive detection-based buffer-and-recover tracking using imported YOLO detector networks[4, 46], and multi-animal pose-estimation, using DeepLabCut[see below 20]; it also provides extensive annotation options. Tracking is achieved by linking YOLO detections across frames via Kalman-Filtering and the Hungarian method for track association [47]. To assess the performance of this simple tracking architecture, we imported the best performing detection networks trained exclusively on synthetic data into *OmniTrax*, and tracked laboratory and field recordings of *A. vollenweideri* leafcutter ants and *Gnathamitermes* sp. desert termites (Fig. 2. See above and methods for details). The ant detector tracked between 61 and 103 *A. vollenweideri* ants over 1000 frames at 30 fps, equivalent to real time inference on a consumer-grade laptop (6 core CPU, 16 GB Ram, RTX 2070); the desert termite detector tracked 49 individuals across 1000 frames. Default tracker settings were used for both test cases.

The ant tracker achieved Multiple Object Tracking Accuracy (MOTA) scores of 0.901, 0.945, 0.859, 0.821 in the "base", "dark", "bright", "noisy" cases respectively (see equation 3, [48] and Fig. 4) . Most ID switches were caused by track fragmentation, which can be avoided by refining tracker settings, or through simple manual corrections within *OmniTrax*. The desert termite tracker achieved a MOTA of 0.96. Only two true ID switches occurred; the remaining errors reflect partially fragmented tracks, and a single out-of-focus animal which was not registered (Fig. 4). Thus, and despite its structural simplicity compared to other recent approaches[16, 49–51], the detection-based tracker powered by *replicAnt* and implemented in *OmniTrax* can track a large number of animals in crowded and open scenes—without the need to hand-annotate a single image.

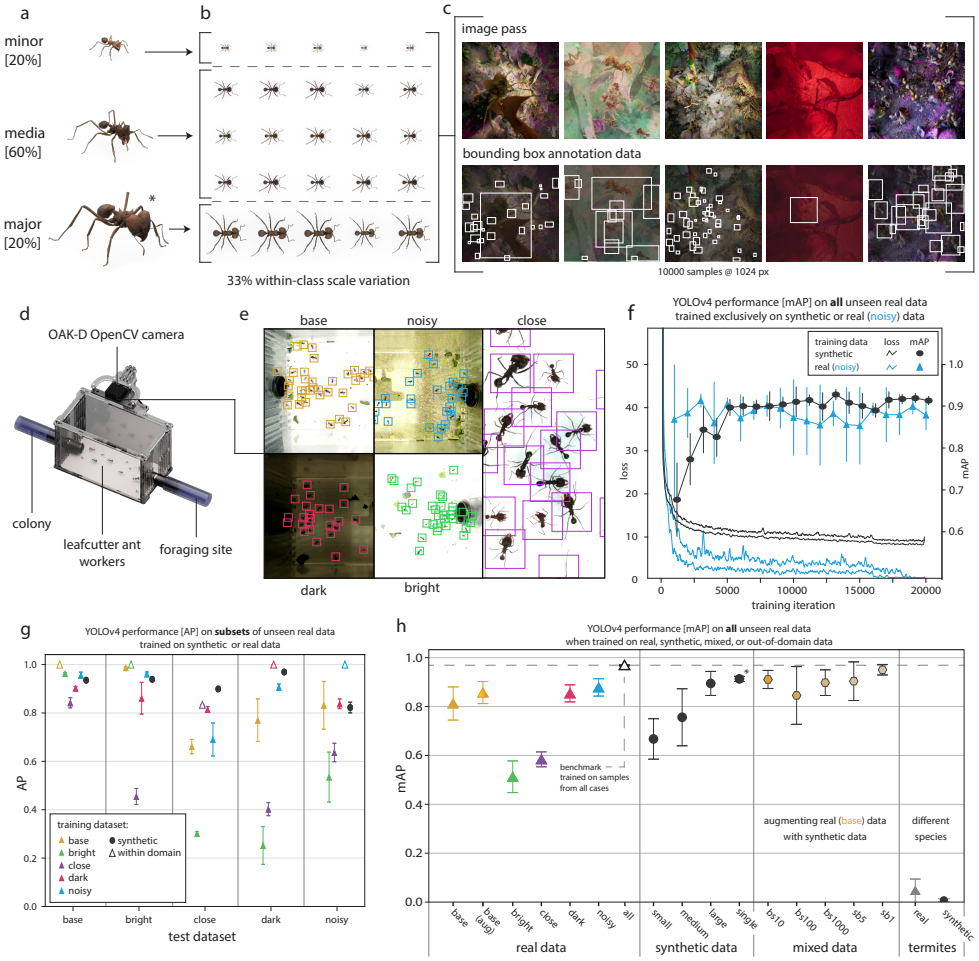

Fig. 2 Performance of YOLOv4 detectors trained with real, synthetic and "mixed" data. (a) 3D models of leafcutter ant workers, created with *scAnt*[40], form the basis of (b) a digital population comprising 100 individuals which differ in scale, hue, contrast, and saturation. *replicAnt* was then used to produce synthetic datasets with 10,000 annotated samples from this population. (c) Examples of image render passes (top row) and bounding box annotations (bottom row). (d) Test data were obtained with a laboratory setup consisting of an OAK-D OpenCV camera, which recorded foraging trails of *Atta* leafcutter ants from a top-down perspective. (e) Recording conditions were varied to produce five sub-datasets of varying difficulty (see methods for details). (f) YOLOv4[4] networks converged more slowly when trained on synthetic data, compared to when trained on any set of real data while ultimately yielding higher mAP scores indicating a more complex training task. (g) Networks trained on real images from any sub-dataset perform poorly on out-of-domain recordings, as indicated by the low average precision (AP) of the detections (solid triangles). In notable contrast, networks trained solely on synthetic data achieved a strong detection performance throughout, likely because they have been exposed to considerably larger variation at training time (black circles). (h) The superior performance of networks trained on synthetic data is most apparent when the mean Average Precision (mAP) is compared directly: The highest mAP of 0.951 ± 0.014 was achieved by a network trained on a mix of synthetic and real data (case sb1, 10,000 synthetic and 100 real images; see methods and SI for a full breakdown of each dataset). Remarkably, the second highest and most consistent mAP was achieved by a network trained solely on synthetic data (0.913 ± 0.001, marked with an asterisk).

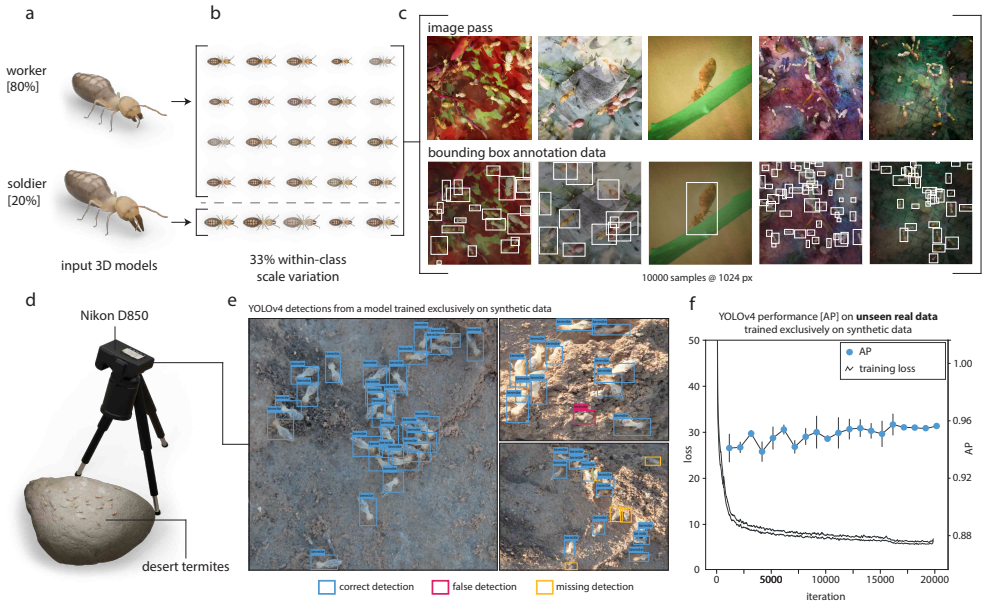

Fig. 3 Even low fidelity 3D models can be used to train performant networks for low-magnification applications such as animal detection. (a) 3D models of a worker and a soldier desert termite (*Gnathamitermes* sp.) were sculpted and textured from reference images in Blender v3.1. (b) A digital population comprising 80 workers and 20 soldiers, each with randomised scale, hue, contrast, and saturation, was generated from these models (see SI A4), and used within *replicAnt* to generate a synthetic dataset with 10,000 annotated images. (c) Examples of image render passes (top row), and bounding box annotations (bottom row). (d) Test data was recorded in the field, using a Nikon D850 and a Nikkor 18-105 mm lens. (e) Example frames demonstrate the high precision of a YOLOv4 detector trained exclusively on synthetic data; only a small number of occluded or blurry individuals were missed, and few false positives were produced (confidence threshold of 0.65, and non-maximum suppression of 0.45). (f) The YOLOv4[4] network converged after about 20000 iterations, and achieved an Average Precision (AP) of 0.956 $\pm$ 0.001, retrieved from returned detections on 1000 hand-annotated frames of 49 freely moving termites (see methods).

2.2.3 Pose-estimation

Animal pose-estimation typically leverages transfer-learning. Although excellent performance is possible in controlled settings, the characteristically small training datasets usually fail to provide the variability required for generalisation to unseen recording settings. As a result, performance is extremely sensitive to changes in scene or specimen appearance[15–17]. The key problem is that maximising performance through overfitting of perspective-dependent latent features leads to domain-dependence. Our aim is to overcome this limitation by leveraging the large and variable synthetic datasets produced by *replicAnt* to embed an improved subject understanding into the pose-estimation networks. In other words, we ultimately seek to train a single generalist network, rather than several scene- and perspective-dependent specialists, as is currently best practice[16, 17, 20].

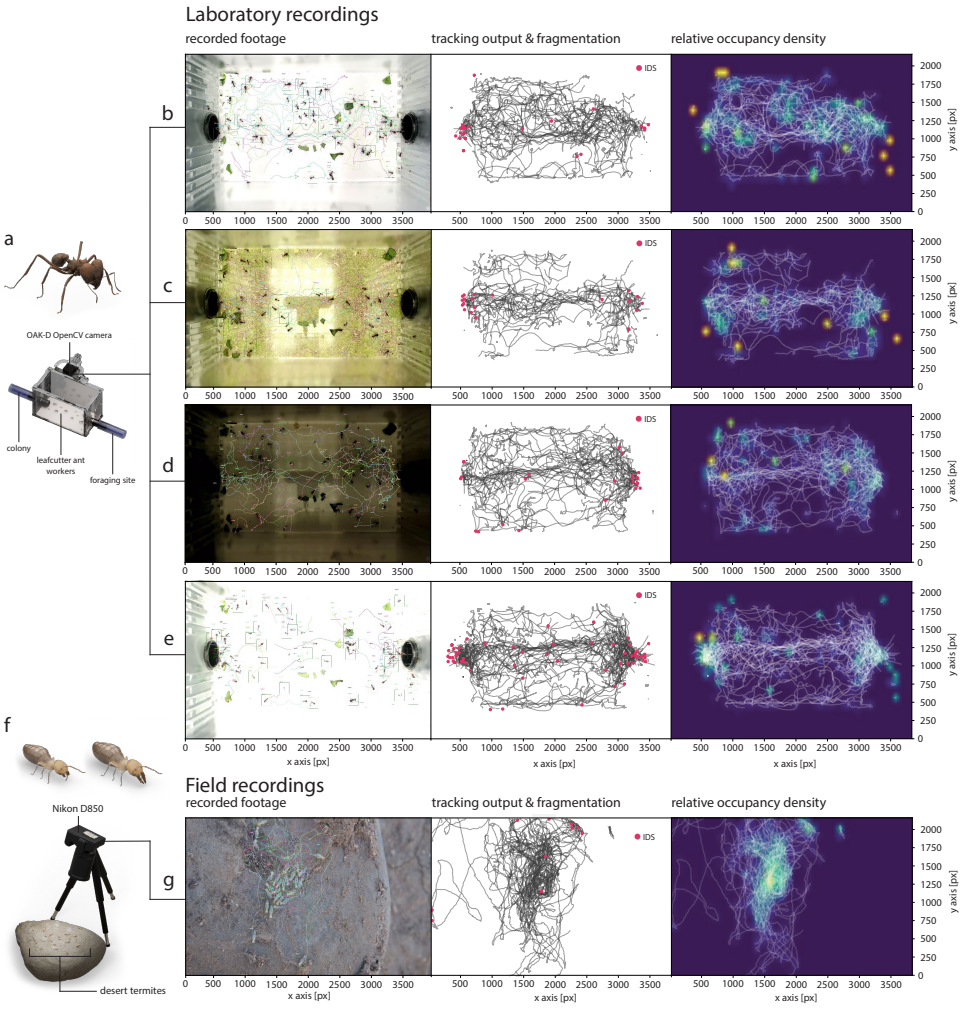

Fig. 4 Detectors trained exclusively on synthetic data can be used for multi-animal buffer-and-recover tracking. (a) A YOLOv4 network[4] was trained on synthetic images generated from a single *Atta vollenweideri* specimen, and then used within *OmniTrax*[45] to automatically track individual ants in (b) well-exposed, (c) noisy, (d) dark and (e) bright (over-exposed) footage (left column). Detections are provided by the YOLOv4 network[4], and linked across frames through a simple buffer-and-recover approach, using a 2D Kalman-Filter implementation and the Hungarian method for track association cost assignment[47]. (b-e) Tracking performance was evaluated by tracking between 61-103 individuals, which could freely enter and exit the recording area, over 1000 frames, with up to 62 animals present simultaneously. ID switches (IDS) are marked in red and mostly occur at the entrances to the recording site. They typically result from track fragmentation due to prolonged occlusion, and can thus be easily excluded from further analysis (middle column). Tracking performance deteriorates with overexposed images (e), fuelled by a combination of motion blur and drastic changes in appearance due to exposure clipping. Relative occupancy density maps visualise cluttered areas, path preferences, and static individuals (right column). (f-g) A YOLOv4 network[4], trained exclusively on synthetic images of desert termites (Gnathamitermes sp.) was used to track 49 desert termites across 1000 frames from field recordings. Only two true identity switches occurred (IDS, entrance at the top right corner); all other IDS are the result of fragmented tracks.

To move towards this aim, we used a 3D model of a sunny stick insect (*Sungaya inexpectata* Zompro 1996, first instar) to generate 10 sub-datasets of a one digital sunny stick insect with different randomisation seeds, characterised by 70% scale variation, and hue, brightness, contrast, and saturation shifts producing 1,000 samples each. These datasets were combined into one single-animal synthetic dataset encompassing 10,000 images at a resolution of 1500×1500 px; the automated annotations included the location of 46 key points distributed along the body (Fig. 5 a-c, See methods and SI for further details). Dataset generation took about three hours on a consumer-grade laptop (6 Core 4 GHz CPU, 16 GB RAM, RTX 2070 Super).

The synthetic dataset was then used to train a DeepLabCut[20] (DLC) pose estimator with a ResNet101 backbone. We emphasize that other excellent markerless pose estimators, such as SLEAP[16] or DeepPoseKit[17], exist, and merely use DLC by way of example. The best tool will likely depend on the specific use-case. To test pose-estimation performance, two datasets of walking sunny stick insects were curated as test cases (Fig. 5).

One dataset, denoted "platform", represents a typical controlled case, where lighting and image background are constant, and only camera perspective varies: *S. inexpectata* were recorded walking across an evenly lit, tiltable platform, at 55 fps with five synchronised machine vision cameras (Fig. 5 d-e). From these recordings, all 49 key points were hand-annotated in each of 805 frames (see SI for split details). The second dataset, denoted "handheld", consists of 200 hand-annotated frames from ten handheld videos (20 frames per video), recorded with a cell phone at 25 fps (Fig. 5 h). This dataset represents an uncontrolled case with variable recording conditions, and includes motion blur, perspective and magnification changes, out-of-focus frames, and frequent partial occlusions.

A DLC network trained on frames from all camera perspectives achieved a benchmark mean relative error on platform data of 10.9% across all camera views (Fig. 5 g, see equation 4). In remarkable contrast, networks trained exclusively on data from a single camera perspective produced a mean relative error of up to 86.4% for frames from unseen perspectives. This poor performance partially reflects domain-sensitivity, characteristic of many transfer learning approaches: the networks fail to generalise, because they have only been exposed to a small inference-specific dataset with limited variation.

A DLC network trained exclusively with synthetic data seemingly competes with the benchmark performance out-of-the-box: it achieved a mean relative error of 5.89% across all platform camera views (Fig. 5 g). However, this low error is deceiving, as the network assigned low confidence scores to more than 50% of key points, which are thus excluded from the error estimate. However, the network provides an excellent starting point for refinement: the provision of a mere ten hand-annotated frames per camera orientation suffices to estimate key points with a mean relative error of 8.14%—better than benchmark performance.

The limited ability of networks trained on real data to generalise becomes even more apparent when they are put to work on recordings of the same species, recorded under different conditions. The pose-estimation network trained on the full platform dataset achieved a mean relative error of 77.18% on the handheld dataset (Fig. 5 g). Key points were frequently placed more than two body lengths away from the specimen, demonstrating that key point detection strongly relies on recording-specific latent features; the volume and variability of the supplied training data was insufficient to embed a general specimen-specific understanding. In sharp contrast, the network trained solely on synthetic data achieved a mean relative error of 6.25% on handheld recordings—more than an order of magnitude smaller. Refinement with five randomly sampled frames from each video resulted in a mean relative error of 5.03%, close to the benchmark performance of 3.55%, achieved by a network trained on the full handheld dataset (Fig. 5 g & i).

On the basis of the above, we conclude that the large sample size and variability afforded by synthetic data can meaningfully increase the domain-invariance and robustness of pose-estimation networks, and thus substantially reduce the required user effort: better or near-benchmark performance was achieved with 4-fold and 16-fold fewer hand-annotated samples in the handheld and the platform case, respectively (Fig. 5 e-i).

2.2.4 Semantic segmentation

We have demonstrated that synthetic data generated by *replicAnt* can substantially reduce the hand-annotation required to power accurate detection, tracking and pose-estimation, or even render it obsolete. Next, we show that it can enable inference in applications for which hand-annotation is so onerous that it is unlikely to be performed at the required scale for all but the most common objects: semantic segmentation, a computer vision task involving pixel-level classification (Fig. 6).

replicAnt was used to generate a digital population of 20 leaf-footed bugs (*Leptoglossus zonatus*, Dallas, 1852), based on a 3D model of an adult specimen produced with *scAnt*[40] (Fig. 6. a-b). Two synthetic datasets were generated, each encompassing 10,000 images with 1024×1024 px resolution (Fig. 6. c): one dataset used *replicAnt*'s default parameters; for the other, *replicAnt*'s asset library was supplemented with various plant models from the Quixel Asset library (Epic Games, Inc.), in order to simulate the image content of typical field macro-photographs. Data generation took between 6-10 hours on a consumer-grade laptop (6 Core 4 GHz CPU, 16 GB RAM, RTX 2070 Super) for the default and plant case, respectively.

Images from both datasets were combined to form a single dataset, used to train Mask-R-CNN[6], UperNet + SWIN transformer[52], and PSPNet[53] networks (see methods and SI for details). Producing large validation datasets is infeasible for highly specific semantic segmentation tasks—the very reason why synthetic data is so helpful in these applications. To provide an indicative performance metric, we extracted network masking accuracy for a small number of hand-annotated image

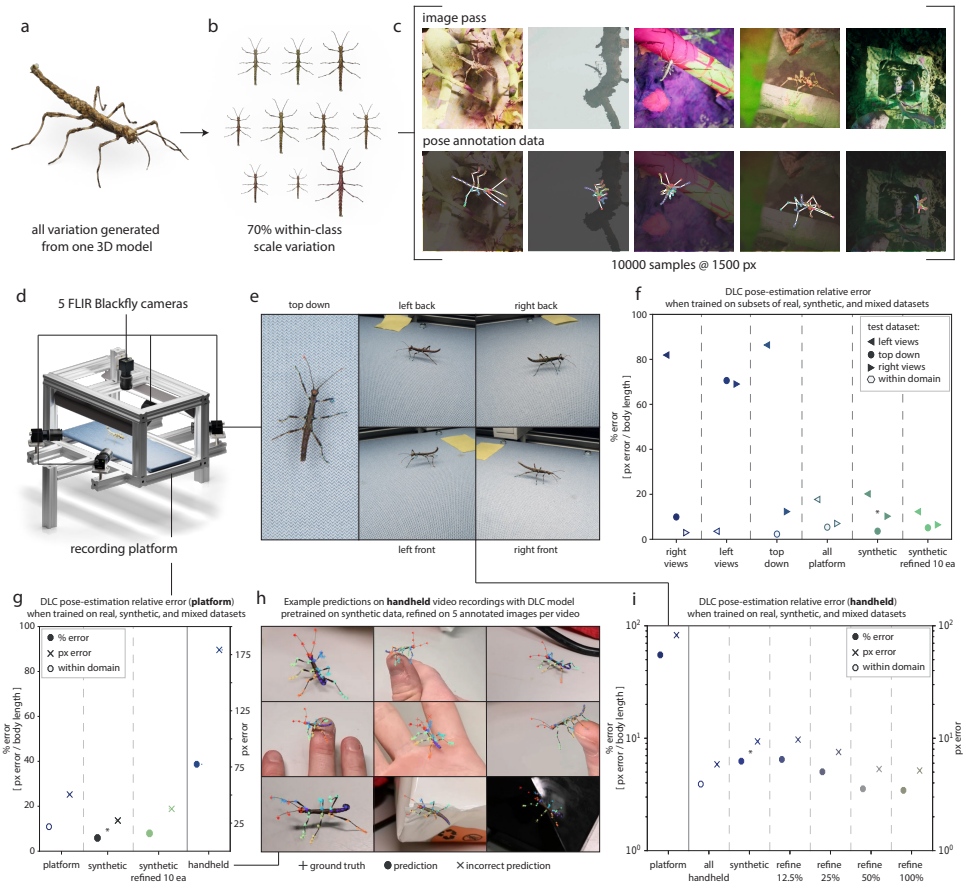

Fig. 5 Performance of DeepLabCut markerless pose-estimators[20] trained on real, synthetic and mixed datasets. (a-b) A digital population of 10 individuals was generated from a single 3D model of a *Sungaya inexpectata* stick insect, created with the open-source photogrammetry platform *scAnt*[40]. Population subjects differed in scale, hue, contrast, and saturation, and formed the basis for a synthetic dataset with 10,000 images at 1500×1500 px resolution. (c) Examples of image render passes and key point annotation; the locations of the 46 key points are provided in the SI. A DeepLabCut[20] network with a ResNet101 backbone was then pre-trained on the synthetic dataset for 800k iterations, and fine-tuned on splits of real samples for a further 800k iterations (see methods and SI for details). Two hand-annotated datasets serve as test cases: (d-e) five synchronised machine vision cameras mounted to a tiltable platform recorded walking stick insects in controlled conditions, so that only camera perspective varied; (h) handheld cell phone recordings of walking stick insects across the laboratory, which include partial occlusions, and variations in background and lighting. (f-g) Networks pre-trained on synthetic data and refined with just 10 samples per camera perspective achieved a mean relative error of 8.14%, and a mean pixel error of 37.69 px across orientations—lower than the benchmark performance achieved by the network trained on the full dataset of 805 total example images (mean relative error of 10.90%, mean pixel error of 50.57 px). The asterisk, *, indicates networks for which $\leq 50\%$ of the inferred key points were above the confidence threshold of 0.6. (i) The performance of pose-estimation networks trained with real data deteriorated drastically when they were put to work on recordings of the same species obtained under different conditions, demonstrating their reliance on specimen-independent features. This domain-specificity is most evident in the handheld recordings, where camera orientation, lighting, and background change continuously: networks trained on real platform data performed poorly, but networks trained solely on synthetic data approach benchmark performance with the addition of just five hand annotated example images (i, refine 25%), indicating a stronger specimen-specific understanding (see example videos in SI).

examples via the Average Class-wise Recall (ACR. See equation 5).

All trained networks were able to identify the majority of specimen pixels, and segmented few background pixels (Fig. 6 d-e). Remarkably, PSPNet, the oldest tested architecture, produced the most accurate segmentations at both high and low magnification, and even in the presence of partially occluded or out-of-focus bodyparts: it achieved an ACR of 94.03% (Fig. 6 d-e). Overall mask quality was lowest for Mask-R-CNN, which struggled with fine appendages at higher magnification and in images with higher background noise (ACR of 82.3%). This problem may in part be specific to the particular implementation of Mask-R-CNN, which was trained using lower resolution segmentation polygons instead of a per-pixel segmentation mask encoding (see methods for details). Mask-R-CNN does however additionally produce instance segmentations, useful where images contain more than one individual. To utilise this feature, a Mask-R-CNN network was trained on 10,000 synthetically generated images of leafcutter ants (Fig. 2 a-c and Fig. 6 f; see methods for training details), and used to run inference on photographs of foraging *Atta*. The produced masks were of high quality, and contained few false positives (Fig. 6 g).

3 Discussion

Deep learning-based computer vision methods promise to fundamentally alter what is possible in animal behavioural research[12–18]. A key remaining bottleneck is the "data-hunger" of supervised learning techniques: annotated datasets of the size and variability required to achieve robust, domain-invariant performance are rarely available, and in any case time-intensive to produce[36, 54]. One strategy to overcome this limitation is to produce annotated data synthetically, using sufficiently realistic computer simulations[31, 32, 34, 36]. In order to facilitate this process, we developed *replicAnt*: a synthetic data generator built in Unreal Engine 5 and Python. *replicAnt* is designed to run on consumer grade hardware, and can generate around 1000 annotated images per hour. We provide extensive documentation, parser scripts for popular deep learning frameworks, pre-trained networks for all listed applications, benchmark datasets, additional software to aid automated detection-based buffer-and-recover tracking and 2D pose-estimation[45], and a growing library of ready-to-use 3D animal models.

The utility of *replicAnt* was demonstrated by using it to train deep neural networks for stereotypical tasks in animal behavioural research. In multi-animal detection, tracking, and semantic segmentation, networks trained exclusively with synthetic data achieved a performance sufficient to remove the need for hand-annotation altogether. In markerless pose-estimation, pre-training networks on synthetic data increased the subject-specific understanding of the networks, so enabling a reduction of the amount of hand-annotation required to achieve benchmark performance by more than one order of magnitude (Fig 5 g). The resulting reduction in time costs enables broad comparative studies – of key biological importance, but currently absent from the literature[55]. We hope that open sharing of 3D models, test data and trained

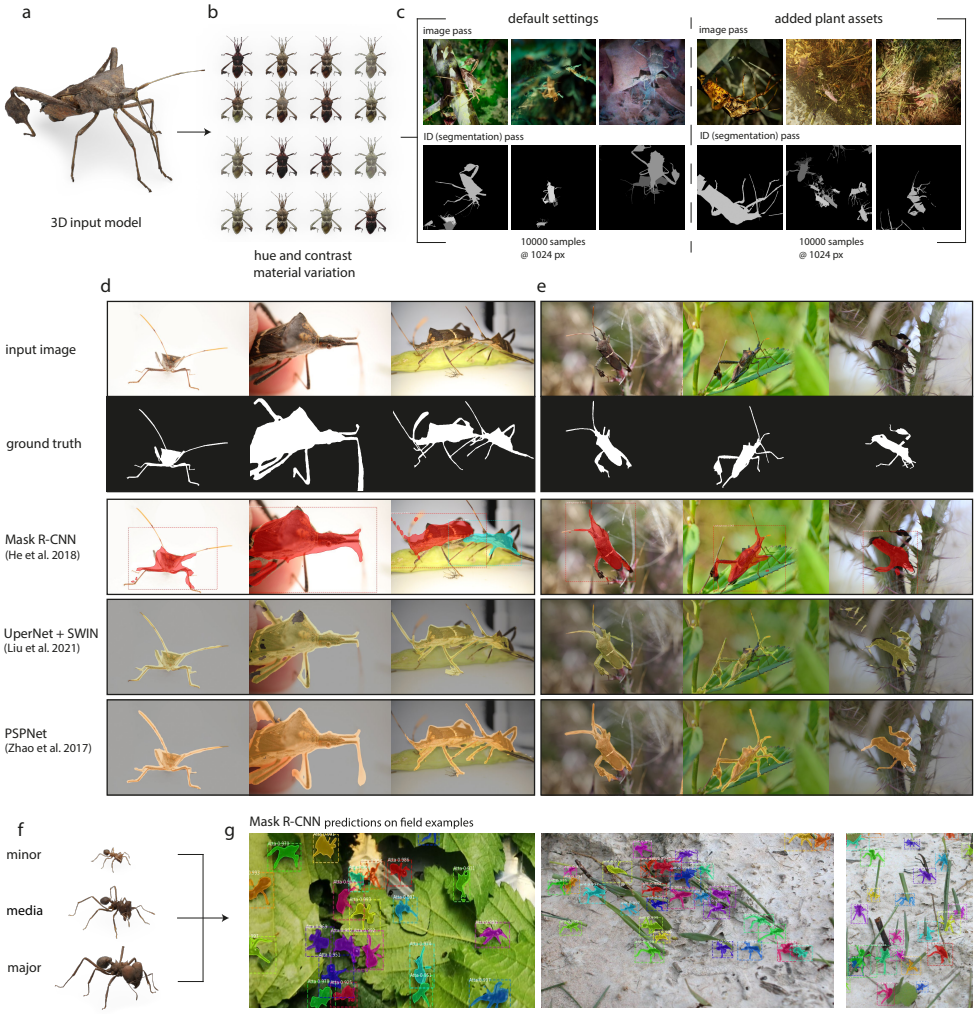

Fig. 6 Performance of semantic- and instance segmentation networks trained exclusively on synthetic data. (a-b) A population of *Leptoglossus zonatus* leaf-footed bugs was simulated from a 3D model of an adult, produced with the open photogrammetry platform *scAnt*[40]. (c) This population was used to generate two datasets, each encompassing 10,000 images: one with *replicAnt*'s default settings; the other with additional scatterers which placed 3D assets of plants to simulate the image content of typical macrophotographs. Images show examples of the image (top row) and ID (segmentation) passes (bottom row), respectively. (d-e) Three deep convolutional semantic segmentation networks—Mask-R-CNN[6], UperNet + SWIN Transformer[52], and PSPNet[53]—were trained on the synthetic data, and their performance was assessed on a small number of hand-annotated (d) laboratory and (e) field macro-photographs (see SI for details). The highest Average Class-wise Recall (ACR) of 94.03% was achieved by the PSPNet architecture with a ResNet101 back-bone. The UperNet + SWIN Transformer network achieved an intermediate overall performance, reduced by fragmented masks and false positives. Segmentations produced by the Mask-R-CNN network had the lowest ACR of 82.3%, but as Mask-R-CNN additionally performs instance segmentation, it is attractive for images that contain multiple individuals. (f-g) To illustrate this feature, a Mask-R-CNN network was trained on 10,000 synthetic images of *Atta vollenweideri* ant workers (Fig. 2), and used to segment (g) crowded laboratory (left) and field photographs (right) of foraging *Atta* leaf-cutter ants.

networks will decrease the need for case-specific refinement, and eventually lead to powerful "generalist" networks.

Ample of opportunity for expansion of *replicAnt* exists. For example, the combination of depth passes and camera extrinsics and intrinsics can in principle be used to train networks to infer 3D locations directly from a single 2D image[28, 32, 56]; informing posture variation within *replicAnt* with 3D kinematics data from live animals may yield networks which can infer the location of occluded key points with reasonable accuracy; and further annotation, for example on minute species differences or body size, can readily be appended. *replicAnt* also provides a fertile test-ground to further our fundamental understanding of supervised learning. As an illustrative example, *replicAnt* can in principle generate arbitrarily large datasets. Not all images are created equal, however, and the control over environmental variability, combined with the ability for "mixed training", provides an excellent opportunity to probe which image elements are most effective for accelerated network fitting. Ultimately, it is our hope that *replicAnt* represents a significant step towards porting machine-learning based computer vision tools to the field.

4 Methods

The generation of synthetic datasets with *replicAnt* can be subdivided into three steps: (i) 3D Subject Model creation and preparation; (ii) set-up of the generator in Unreal Engine; and (iii) parsing generator outputs into common machine learning data formats to train deep neural networks of choice. In the following sections, we first outline the general structure of the data generation process; application specific details are provided at the end. Detailed documentation and interactive Jupyter notebooks for data parsing and benchmarking are available on GitHub (https://github.com/evo-biomech/replicAnt).

4.1 3D Subject Models

In principle, 3D models used in *replicAnt* can come from any source. However, model fidelity is a primary determinant of the simulation-reality-gap, and thus influences the performance that can be achieved. Practically, the required model fidelity depends on the desired application. For example, in applications which typically involve low resolution footage, such as multiple animal tracking (Fig 2 – 4), lower fidelity 3D models may suffice; networks used for high-resolution semantic segmentation (Fig 6) or pose-estimation (Fig 5), in turn, require models with higher fidelity. We used the open-source photogrammetry platform *scAnt*[44] to produce high fidelity models, and Blender v3.1 to sculpt lower fidelity models. Irrespective of model origin, the model mesh may need to be cleaned, retopologised, and rigged prior to import in the generator. For all models used in this study, this process was completed using Blender v2.92 & 3.1.

4.1.1 Clean-up

All unconnected vertices and floating artefacts of the model mesh were deleted, and surfaces cleaned, using Blender's native editing and sculpting tools. Holes were closed by collapsing surrounding vertices to a single point, and/or rebuilding the surrounding topology. The re-connected mesh regions were made seamless by projecting adjacent texture information onto the collapsed or newly created area, respectively; any overlapping vertices and self-intersecting faces were removed (for further details, see ref. [44]).

4.1.2 Retopologising

To accelerate the data generation process and to allow for larger simulated digital populations, we decreased the mesh resolution of each model to between 1,000 to 10,000 vertices, using Blender's native decimate modifier. The number of vertices was chosen such that the overall shape was preserved, but fine topology information, such as hairs and other surface detail were removed; this information was captured by albedo and normal maps instead, generated for the retopologised meshes through texture baking from the original high-resolution input[44].

4.1.3 Rigging

In order to enable posture variation, models were rigged – each model was assigned a set of rigid segments referred to as bones, with individual bones connected through joints. The collection of bones and joints defines the model's armature. In principle, users can assign an arbitrary number of virtual bones and joints, each with a specific range of motion. We provide a base armature template that was used throughout this work; it can readily be adapted to animal-specific needs (see Appendix Fig D1). The segment deformation associated with joint movement was restricted to proximal parts of each mesh segment using weight painting, an appropriate simplification due to the effectively rigid arthropod exoskeleton.

4.1.4 Model porting to Unreal Engine

Curated models, including materials, textures and the armature, were transferred from Blender to Unreal engine 5 via the send-to-unreal Add-on (https://github.com/EpicGames/BlenderTools). Rigged meshes can also be imported directly into Unreal Engine 5, but may then require additional manual editing. Regardless of model origin, process shader, scaling, collisions properties, and animation blueprints need to be assigned to the subject model after import. Examples of configured 3D subject models as well as detailed documentation on the model preparation process are available on GitHub (https://github.com/evo-biomech/replicAnt).

4.2 *replicAnt*

The core of *replicAnt* is a pre-configured Unreal Engine 5 project which includes: (i) example subject models; (ii) project configurations, referred to as levels within Unreal Engine 5; (iii) an asset library – a curated collection of 3D meshes which may

populate the generated scenes; (iv) a set of basic materials – image maps that are blended with procedurally generated textures to define the appearance of scattered assets and the ground; and (v) High Dynamic Range panorama images (HDRIs), used as the environment background and to provide additional scene lighting.

These elements form the backbone for the dataset generation process, which can be controlled through a simple user interface; more advanced configuration is possible through editable blueprints, a visual and structured node-based alternative to the use of text-based source code. User-control is facilitated through four main components: (1) The User Interface editor widget (UI); (2) the content browser; (3) the outliner; and (4) the viewport.

The UI is divided into four tabs: General – to define the randomisation schedule of all generator elements, to control the balance between generation speed and output variability, to configure the desired output types and location, and to set the number of unique output samples; Subjects – to select 3D models for inclusion in the digital population, and to specify subject population size and appearance randomisation; Environment – to select HDRIs and to specify the ground mesh resolution; and Debug – to control various stalling periods related to stability, and the randomisation seed. The seed value controls the randomisation routines of all elements and so enables repeatability. For example, identical scenes can be populated with different subjects, so producing datasets of different animals in equal environments.

The Content Browser is akin to a file explorer, and provides access to the full project data structure, all assets and blueprints. Additional content, such as textures or 3D models, can be imported, and key individual components of the data generator can be modified (for additional documentation refer to https://docs.unrealengine.com/5.0/). The Outliner displays and provides access to all elements included in the current project configuration, for example the generated subject population, the scene camera, the ground plane, and lights. Users can add, edit, or remove randomisable elements such as asset scatterers, light blocking elements, and decal generators (for details, refer to the respective sections below).

Finally, the viewport provides a 3D preview of the current project state. By selecting "change preview environment" from within the Environment tab of the UI, or by selecting "Test" at the bottom of any of the UI tabs, users can directly assess how the current settings translate into the randomised scene generation process, through either randomising the lighting setup or executing a complete randomisation iteration respectively. The viewport also provides a window into active dataset generation processes: it displays snapshots of output passes, and indicates both the time elapsed and an estimated time to completion.

Dataset generation then proceeds through simulation of scenes, defined by a series of randomisable elements as detailed below. In order to generate datasets with maximal variability in minimal time, the frequency with which scene elements are updated is

tied to the time it takes to execute each update. To rationally guide this process, we defined a hierarchy from computationally most to least demanding randomisation components: (i) ground plane, (ii) asset, (iii) subject placement and pose, (iv) material, lighting, camera intrinsics and extrinsics, and image post processing. Elements lower in this hierarchy influence elements upstream, and the frequency with which each element is updated can be user-controlled (Fig 1). We now briefly explain the basic scene generation at each of these hierarchical element levels.

4.2.1 Environment

The environment, which we define as all scene elements aside from subjects, is procedurally build from a number of hierarchically linked modules of surface and material generators, asset scatterers, and a dynamic lighting system. In the first step, a ground plane is generated and tessellated to create a parameterisable surface of up to 5120000 triangles. All assets are later placed on this ground, initially created as a 2D surface plane. Height variation is then encoded through a three-channel RGB map. Each channel controls different aspects of the terrain generation process, a design choice which enables independent variation of terrain topology and terrain material. The RGB map is procedurally generated through a set of sub-modules which introduce mesh noise of variable granularity and geometric pattern. In order to maximise variation in terrain topology and material, the generated noise is blended with a curated library of monochromatic displacement maps. The red channel encodes the displacement map for the tessellated ground plane. High values (≥ 0.5) indicate a positive, and low values (≤ 0.5) a negative shift in height relative to the mean height, respectively; a value of 0.5 thus corresponds to mean ground plane height. The green channel encodes additional noise to blend different materials. The blue channel provides an opportunity for additional user-defined variation; it does not affect default randomisation routines.

4.2.2 Asset scatterers

In order to increase the variability of the environment, 3D assets – texture-less photogrammetry models of common objects – are placed on the terrain by a user-defined number of asset scatterers (all models used in this study are from Quixel AB, Epic Games. For a full list, see SI). Each scatterer is assigned a number of assets, one of which is randomly selected per iteration (see Methods 4.2). Each scatterer is also assigned a material at random; assets spawned by the same scatterer thus share the same material, so that no separate texture information is required for each asset mesh. As a result, the project file size remains small, even though the numbers of meshes in the asset library is large. A number of scatterer presets are provided. The number of scatterers and their individual configuration can be set in the Outliner, which provides further control over key parameters of each scatterer: (i) the set of assets which may be scattered; (ii) the asset size range (all assets are by default of equal size, that is their largest length in X,Y,Z coordinate space is normalised); and (iii) the asset number range, defining the minimum and maximum number of instances of the drawn asset the scatterer may spawn per iteration, respectively.

4.2.3 Subject placement and posing

In each randomisation iteration, the generator places subjects drawn from a user-defined "population" at a randomised coordinate. Each subject is subjected to pose variation, which is randomised at two levels: mesh-interacting and mesh-independent posing. Key in this process is an animation blueprint (ABP_InsectBase), which specifies the list of body parts subject to mesh-interacting and mesh-independent posing, respectively, along with the joint types and the joint range of motion. Different subjects within the a population can be assigned individual animation blueprints by generating "child instances" of the original "parent" blueprint.

During mesh-interacting posing, subjects are first scattered and then rotated around their geometric centre, once they are in proximity to the generated ground plane. If a subject intersects with the ground plane or a scattered asset or their randomised location lies below the ground plane, the process is repeated until valid locations are determined for every subject. Each subject is moved along its down-vector until its mesh intersects with any scattered asset, previously spawned subject, or the ground plane. If a subject does not intersect with any mesh along its trace, a new centre-rotation is proposed, and the process repeated until a valid rotation and resulting placement location are found. Possible end-point locations are determined via ray casts; the Unreal Engine 5 internal Full body Inverse Kinematics solver (IKS) is then used to determine permissible joint angles for all body parts assigned in the respective animation blueprint. By default, mesh-interaction posing is performed for all leg segments, i. e. subjects generally stand on other meshes (see SI Fig D1). By restricting the number of solver iterations, however, some legs can also be left intentionally "mid-air". In such instances, a random point is placed within reach of the most distal part of the IK chain of the respective leg, and the IKS is used to solve for an appropriate joint configuration.

Mesh-independent posing, in turn, controls the pose of other key "bones", such as the head, antennae, or mandibles; the respective joints are each assigned a random angle within the permissible range.

4.2.4 Materials, Lighting, camera and post-processing

Materials for the terrain, decal and asset layers are generated by independent material generators. Each generator combines randomly generated patterns with a curated library of image textures. Further textures can be added to the respective content directories. The selection of textures, pattern generation, and blending are controlled via the seed value defined in the debug tab of the UI controls.

The environment is then lit using a series of lighting elements: colour-filtered High Dynamic Range Images (HDRIs); a main directional light; randomly placed coloured spotlights, which cast multiple sharp and diffuse shadows, either from scattered assets onto the subjects, or from the subjects themselves onto the environment; light blocking

occlusion planes; and volumetric fog, which introduces diffuse lighting and reduced visibility. The array holding the HDRIs can be appended with external images, and the volumetric fog density can be adjusted within the environment tab of the UI. For example, users may wish to match specific lighting conditions of an experimental setup, or simply increase lighting variation further. The number and colour of spawned spotlights is user-definable (see documentation on GitHub for details).

The combination of ground topology, assets, material and decal layers, subjects and lighting fully defines a specific populated environment. In a last step, a simulated camera is randomly spawned, and oriented such that it points towards a randomly selected subject; this random orientation offset ensures that the subject location relative to the image centre varies across images. Using this camera, an annotated sample image of the randomisation iteration can be captured; each unique camera perspective on an environment constitutes a scene. In addition to camera location and orientation (extrinsics), the generator can also randomise camera intrinsics, such as the simulated sensor size, focal length, aperture, and exposure. The camera randomisation step therefore presents a prime opportunity for computationally inexpensive variation, as variable scenes can be extracted from a single populated environment. In a final step, an array of post-processing filters, providing control over colour temperature and tint, saturation and contrast, vignetting, and various types of grain, are applied. These filters only affect the image render pass (see Generator outputs 4.3). All camera and post-processing attributes are randomisable (see documentation on GitHub).

4.3 Generator outputs

The generator produces two key outputs: annotation files and image passes. The annotation files comprise a single batch file, containing information of relevance for the whole dataset, and one annotation file per scene, containing image specific annotations. Image passes decode key scene information (see below); custom pass types can be added, or configured passes modified (see GitHub documentation). The desired image pass types, image resolution, compression and output directory are specified in the general tab of the UI.

4.3.1 Annotation files

Each dataset is accompanied by a single human-readable batch file, which contains general information: the number of samples, the dataset name, the seed value, and the constant image pass dimensions. Additionally, it provides references to the IDs internally assigned to each subject within the population, as well as its class, which corresponds to the assigned subject model name, and relative scale. This information is also used by custom data parsers (see below).

For each iteration, the set of image passes is accompanied by a unique human-readable sample file. Sample files document information essential for 3D localisation and pose-estimation applications: the camera intrinsics and extrinsics, including the 3D camera location and rotation, diagonal field of view, and the full view projection matrix. Additionally, sample files provide annotations, such as the coordinates of

subject bounding boxes, all 2D key point locations in pixel space, and all 3D key point locations relative to the camera coordinate for every simulated subject. Further custom annotations can be added.

4.3.2 Image passes

By default, the generator is configured to write four image pass types per scene: image render, ID, depth, and normal passes. Each pass types encodes different key information.

The image render pass is an RGB colour rendering of the simulated camera view, including image noise, variation in exposure, and depth of field (see **??**). The level of compression can be fixed or randomised, and several image formats are available for selection in the general tab of the UI (JPG, PNG, EXR, BMP).

The ID pass encodes subject IDs by assigning all pixels a unique, subject-specific RGB colour value; all non-subject pixels are assigned a value of (0,0,0). Occlusion can thus be determined at pixel-level for the full subject mesh, and for individual key points at the following parser stage. Furthermore, the relative occupancy of the subject mesh within its 2D bounding box can be extracted. Information on occlusion, key point location and bounding box occupancy is essential to exclude fully occluded subjects or individual key points from neural network fitting where desired. In our illustrative pose-estimation examples (see below), occluded key points were excluded from training; however, users can toggle this option on and off in the respective Jupyter notebook parsers (see GitHub documentation). The ID pass also forms the basis for segmentation maps (Fig. 6. c, ID (segmentation) pass) used for run-on encoding, polygon encoding, or per-pixel encoding at the parser stage. ID pass images are always saved in uncompressed PNG format.

The depth pass is a monochromatic render which encodes the depth of each pixel in the scene, relative to the virtual camera plane. In combination with camera intrinsics/extrinsics and the ID pass, depth pass images can in principle be used to train networks for 3D semantic segmentation, size estimation, and depth inference. Depth pass images are always saved in uncompressed PNG format.

The normal pass encodes surface normals, required by some specialised pose-estimation applications[32, 56]. Normal passes can be produced both in camera view space and world space. We did not use normal passes in this work, but provide them to illustrate the modularity and extendibility of *replicAnt*. Normal pass images are always saved in uncompressed PNG format.

4.4 Data parsers

The combination of image passes and annotation files constitutes synthetic data which can be used to train a suite of machine-learning based computer vision models. In order

to facilitate this process, five groups of data parsers were implemented in form of documented Jupyter notebooks. These parsers translate the generated data into various data formats used by popular deep learning-based computer vision models: (i) YOLO compatible data[4, 46]; (ii) three DeepLabCut[20] and SLEAP[16] compatible data parsers; (iii) COCO[6, 10] formatted data, including image masks for detection, single and multi-animal pose-estimation, as well as semantic segmentation applications; (iv) MMSegmentation[57] compatible data for segmentation training and benchmarking; and (v) a custom 3D pose-estimation data format which includes camera intrinsics and extrinsics.

4.5 Applications

The synthetic data generated by *replicAnt* was used in conjunction with a set of data-driven machine learning tools, including multi-animal detection, tracking, pose-estimation, and semantic and instance segmentation. For each of these applications, we below describe (1) the curation of hand-annotated benchmark data; (2) the origin of 3D models and the pertinent details of the synthetic data generation process; (3) the network training procedure; and (4) the validation and performance characterisation. By necessity, these applications required to choose specific network architectures for each use case. In choosing specific network architectures over others, we do not wish to imply their superiority; all choices merely serve as illustrative examples, and other options may be better or worse, depending on the specific use case. Our aim is not to determine the best network architecture, but to test the broad applicability and quality of the synthetic data generated by *replicAnt*.

4.6 Detection

4.6.1 Benchmark data

Two video datasets were curated to quantify detection performance; one in laboratory and one in field conditions. The laboratory dataset consists of top-down recordings of foraging trails of *Atta vollenweideri* (Forel 1893) leaf-cutter ants. The colony was collected in Uruguay in 2014, and housed in a climate chamber at 25°C and 60% humidity. A recording box was built from clear acrylic, and placed between the colony nest and a box external to the climate chamber, which functioned as feeding site. Bramble leaves were placed in the feeding area prior to each recording session, and ants had access to the recording area at will. The recorded area was 104 mm wide and 200 mm long. An OAK-D camera (OpenCV AI Kit: OAK-D, Luxonis Holding Corporation) was positioned centrally 195 mm above the ground. While keeping the camera position constant, lighting, exposure, and background conditions were varied to create recordings with variable appearance: The "base" case is an evenly lit and well exposed scene with scattered leaf fragments on an otherwise plain white backdrop (Fig 2). A "bright" and "dark" case are characterised by systematic over- or under-exposure, respectively, which introduces motion blur, colour-clipped appendages, and extensive flickering and compression artefacts. In a separate well exposed recording, the clear acrylic backdrop was substituted with a printout of a highly textured forest ground to create a "noisy" case. Last, we decreased the camera distance to 100 mm at

constant focal distance, effectively doubling the magnification, and yielding a "close" case, distinguished by out-of-focus workers. All recordings were captured at 25 frames per second (fps).

The field datasets consists of video recordings of *Gnathamitermes* sp. desert termites, filmed close to the nest entrance in the desert of Maricopa County, Arizona, using a Nikon D850 and a Nikkor 18-105 mm lens on a tripod at camera distances between 20 cm to 40 cm. All video recordings were well exposed, and captured at 23.976 fps.

Each video was trimmed to the first 1000 frames, and contains between 36 and 103 individuals. In total, 5000 and 1000 frames were hand-annotated for the laboratory- (Fig 2 e) and field-dataset (Fig 3 e), respectively: each visible individual was assigned a constant size bounding box, with a centre coinciding approximately with the geometric centre of the thorax in top-down view. The size of the bounding boxes was chosen such that they were large enough to completely enclose the largest individuals, and was automatically adjusted near the image borders. A custom-written Blender Add-on aided hand-annotation: the Add-on is a semi-automated multi animal tracker, which leverages blender's internal contrast-based motion tracker, but also include track refinement options, and CSV export functionality[44, 45]. Comprehensive documentation of this tool and Jupyter notebooks for track visualisation and benchmarking is provided on the *replicAnt* and BlenderMotionExport GitHub[44].

4.6.2 Synthetic data generation

Two synthetic datasets, each with a population size of 100, were generated from 3D models of *Atta vollenweideri* leaf-cutter ants. All 3D models were created with the *scAnt* photogrammetry workflow[40]. A "group" population was based on three distinct 3D models of an ant minor (1.1 mg), a media (9.8 mg), and a major (50.1 mg). To approximately simulate the size distribution of *A. vollenweideri* colonies, these models make up 20%, 60%, and 20% of the simulated population, respectively. A 33% within class scale variation, with default hue, contrast, and brightness subject material variation, was used. (Fig 2). A "single" population was generated using the major model only, with 90% scale variation, but equal material variation settings.

A *Gnathamitermes* sp. synthetic dataset was generated from two hand-sculpted models; a worker and a soldier made up 80% and 20% of the simulated population of 100 individuals, respectively with default hue, contrast, and brightness subject material variation (Fig 3). Both 3D models were created in Blender v3.1, using reference photographs.

Each of the three synthetic datasets contains 10,000 images, rendered at a resolution of 1024 by 1024 px, using the default generator settings as documented in the Generator_example level file (see documentation on GitHub). To assess how the training dataset size affects performance, we trained networks on 100 ("small"), 1,000 ("medium"), and 10,000 ("large") subsets of the "group" dataset (see Appendix A2 for dataset sizes and splits). Generating 10,000 samples at the specified resolution

took approximately 10 hours per dataset on a consumer-grade laptop (6 Core 4 GHz CPU, 16 GB RAM, RTX 2070 Super).

Additionally, five datasets which contain both real and synthetic images were curated. These "mixed" datasets combine image samples from the synthetic "group" dataset with image samples from the real "base" case. The ratio between real and synthetic images across the five datasets varied between 10/1 to 1/100 (see SI A3 for dataset sizes and splits).

4.6.3 Network training

We used the AlexeyAB darknet implementation of YOLOv4[4] as a detector, because it balances inference quality and speed, and is in widespread use. Each network was trained for 20000 iterations to ensure convergence. The burn-in rate was set to 1000, the learning rate to 10-4, and trained weights were saved every 1000 iterations. As the benchmark data contain recordings with variable subject magnification, we selected a YOLOv4 variant with adjusted anchors, enabling both small and large detections relative to the image size; this variant performed best in preliminary trials.

All training was performed on a computational cluster with compute nodes providing 16 CPU cores, 64 GB of RAM, and a single NVIDIA RTX Quadro 6000 GPU. A comprehensive list of trained networks is provided in the SI.

4.6.4 Evaluation

In order to evaluate detection performance, we retrieved the Average Precision (AP) over 13 confidence thresholds, equally spaced between 0.2 to 0.8. A detection was considered correct if the Euclidean distance between its centre and the corresponding ground truth detection was within 5% of the image width; multiple detections of the same object were removed by non-maximum suppression at run-time. To provide a single measure of overall detection performance, we report the mean of the individual APs retrieved for each unseen case (mAP).

We chose a centre-based precision definition over the traditional intersection-over-union (IoU), because different methods were used to assign bounding boxes across hand- and computer annotated data: Synthetically generated bounding boxes represent the smallest rectangle which includes all projected 2D key points in the rendered images; hand-annotated bounding boxes, in turn, are fixed-area, square bounding boxes, as the custom-written centre tracking tool[44] does not report the shape of bounding boxes, and extracts bounding box centres for ease of use instead. Thus, in our application, the IoU is secondary to the proximity of bounding box centres.

The AP was computed as defined in the official scikit learn implementation[58]: it summarizes the precision-recall curve with a single weighted mean of the precision at each threshold; the increase in recall from the previous threshold acts as weight:

$$AP = \sum_n (R_n - R_{n-1}) P_n \tag{1}$$

$$mAP = \sum_m \frac{AP_m}{m} \tag{2}$$

Here, P_n and R_n are the precision and recall at the nth threshold. Using decreasing threshold values, the recall R_{n-1} at the first threshold is set to 0; when the threshold is maximal, no detections are returned, so that the precision $P_0(R_0)$ is equal to unity by definition. This calculation differs from computing the area under the precision-recall curve with the trapezoidal rule, which relies on linear interpolation, and can result in inflated performance estimation[58].

Five-fold cross validation was performed for all trained networks, with an 80/20 data split. Due to the similarity of neighbouring frames from videos recorded with a framerate high compared to the average individual movement speed, validation based on withheld frames alone can artificially inflate accuracy. To avoid this inflation, accuracy was instead assessed by computing the mAP of networks tested only on images outside the original recording domain.

4.7 Tracking

4.7.1 Benchmark data

To evaluate multi-animal tracking performance, we used the benchmark datasets curated for the detection experiments — these also provide individual identities across frames due to the annotation with the BlenderMotionExport Add-on[44]. For the laboratory dataset, we used the annotated "base", "dark", "bright", and "noisy" videos, which include between 63 and 103 ant workers. For the field dataset, we use the annotated recording of 49 *Gnathamitermes* sp..

4.7.2 Synthetic dataset generation and automated tracking

In principle, sufficiently precise detectors can be used to build simple yet robust and performant detection-based trackers. By retrieving detections for each processed frame, a simple buffer-and-recover tracker can be used to automatically associate detections of adjacent frames to produce coherent tracks and preserve individual identities across frames. The best performing detector models for laboratory and field recordings, trained exclusively on synthetically generated images of *A. vollenweideri* and *Gnathamitermes* sp., respectively, were used for detection-based tracking (see 4.6.2). To facilitate tracking, we introduce an open-source Blender Add-on, *OmniTrax*[45]. The best performing YOLO detector models were imported into *OmniTrax*. *OmniTrax* then uses the loaded network to retrieve input detections for each frame, and to assign detections across frames to specific individuals; tracks are produced automatically and without further user-intervention (for further information, refer to the *OmniTrax* implementation and ref.[45]). For all tracking, we used the default tracker and Kalman filter settings, a detector confidence threshold of 0.8, and non-maximum

suppression of overlapping detections of 0.45. Detections with bounding boxes smaller than 20 pixels were automatically excluded from track association.

4.7.3 Evaluation

To assess tracker performance, the tracks produced by the detector networks were compared to the hand-annotated ground truth of each frame via the Multiple Object Tracking Accuracy (MOTA) [48]. Calculating the MOTA requires definition of a maximum distance $dist_{max}$ between the ground truth detection centres and the inferred tracks beyond which tracks are no longer considered correct. We chose a $dist_{max}$ of 50 px at 4k resolution, equivalent to 2% deviation relative to the frame size. The MOTA is then a combined measure of three distinct errors: False Negatives (FN_t) – no detection was registered at the location of a ground truth track; False Positives (FP_t) – a detection was registered in the absence of a ground truth track within $dist_{max}$; ID switches (IDS_t) – the identity of an inferred track does not correspond to the identity of the ground truth track previously associated with the inferred track identity. Provided that the identity of an object in the ground truth track at some frame (t) is matched with the identity of an inferred track, we keep track of this correspondence. If, at a later frame $(t+i)$, the ground truth track is associated with a different inferred track, this occurrence is counted as an ID switch. The new identity is then considered "correct" for all subsequent frames, and forms the basis for the identification of any further identity switches. ID switches thus capture "true" identity switches, but also changes in identity due to fragmented tracks that were terminated early and restarted with a new ID. The MOTA is simply the sum of all FN_t, FP_t, and IDS_t errors, divided by the sum of all detections in the ground truth tracks.

$$MOTA = 1 - \frac{\sum_t FN_t + FP_t + IDS_t}{\sum_t det_t} \tag{3}$$

A maximum MOTA score of unity thus implies that all instances have been detected in every frame, were associated with the correct tracks, and no false positives were produced. Multiple-object tracking evaluation can also yield falsely identified ID switches, IDS_t, due to correspondence problems of IDs in close proximity. We keep track of all relevant tracks before and after overlap events, defined as two or more ground truth tracks closer than $dist_{max}$, to check whether possible ID switches are simply the result of changes in distances to the respective closest ground truth detection, although identities are retained correctly after the overlap event. Therefore, when assessing overlapping tracks, track identities are only evaluated before and after the event, so that incorrectly identified identity switches are suppressed. If, however, the number of detections is under- or overestimated during overlapping events, those occurrences are still counted towards FN_t or FP_t respectively.

4.8 Pose-estimation

4.8.1 Benchmark data

Two pose-estimation datasets were procured. Both datasets used first instar *Sungaya inexpectata* (Zompro 1996) stick insects as a model species. Recordings from an evenly lit platform served as representative for controlled laboratory conditions (Fig 5 e); recordings from a hand-held phone camera (Fig 5 h) served as approximate example for serendipitous recordings in the field.

For the platform experiments, walking *S. inexpectata* were recorded using a calibrated array of five FLIR blackfly colour cameras (Blackfly S USB3, Teledyne FLIR LLC, Wilsonville, Oregon, U.S.), each equipped with 8 mm c-mount lenses (M0828-MPW3 8MM 6MP F2.8-16 C-MOUNT, CBC Co., Ltd., Tokyo, Japan). All videos were recorded with 55 fps, and at the sensors' native resolution of 2048 px by 1536 px. The cameras were synchronised for simultaneous capture from five perspectives (top, front right and left, back right and left), allowing for time-resolved, 3D reconstruction of animal pose via DeepLabCut[20] (DLC) and Anipose[59].

The handheld footage was recorded in landscape orientation with a Huawei P20 (Huawei Technologies Co., Ltd., Shenzhen, China) in stabilised video mode: *S. inexpectata* were recorded walking across cluttered environments (hands, lab benches, PhD desks etc), resulting in frequent partial occlusions, magnification changes, and uneven lighting, so creating a more varied pose-estimation dataset.

Representative frames were extracted from videos using DeepLabCut (DLC)-internal k-means clustering[20]. 46 key points in 805 and 200 frames for the platform and hand-held case, respectively, were subsequently hand-annotated using the DLC annotation GUI (see SI B5,B6,B7 for additional details regarding dataset splits and composition).

4.8.2 Synthetic data

We generated a synthetic dataset of 10,000 images at a resolution of 1500 by 1500 px, based on a 3D model of a first instar S. *inexpectata* specimen, generated with the *scAnt* photogrammetry workflow[40]. Generating 10,000 samples took about three hours on a consumer-grade laptop (6 Core 4 GHz CPU, 16 GB RAM, RTX 2070 Super). We applied 70% scale variation, and enforced hue, brightness, contrast, and saturation shifts, to generate 10 separate sub-datasets containing 1000 samples each, which were combined to form the full dataset (see SI B6 for details).

4.8.3 Network training

DeepLabCut[20] version 2.1, built with Tensorflow 2.0 and CUDA 11.2 , was used for markerless pose-estimation by way of example. Other excellent choices such as SLEAP[16] or DeepPoseKit[17] exists, and are compatible with the provided data parsers. The best pose-estimation network may depend on the use case, and thus should be chosen by the end-user.

For all experiments, we used a ResNet101 backbone pre-trained on ImageNet[9], with the skeleton configuration and hierarchy as outlined in the base armature (see SI Fig D1). All networks were trained for 800,000 iterations with a batch size of two; trained weights were saved every 50,000 iterations. These parameters were chosen to prevent excessive overfitting; no further decrease in validation error was observed after 800,000 iterations in preliminary trials. Networks pre-trained on synthetic data were then refined with a small number of real frames for an additional 800,000 iterations (see SI B5 for dataset sizes and splits). We used DeepLabCut's default training and augmentation parameters, with image mirroring disabled, to ensure comparability between the trained networks, rather than attempting to tailor network and parameter choice to any singular dataset. All training was performed on a dedicated computational cluster with compute nodes using 16 CPU cores, 32 GB of RAM, and single NVIDIA Quadro RTX 6000 GPU. For a full list of trained networks, refer to the SI.

4.8.4 Evaluation

Pose-estimation performance was quantified as the mean pixel and mean relative percentage error. The pixel error reported by DLC, $error_{px}$, is the Euclidean distance between the ground truth annotated key point coordinate and the inferred key point coordinate, averaged across all inferred key points with a sufficient confidence value; we chose DLC's default confidence threshold of 0.6. In order to derive a relative error metric, we determined the body length in pixels as the distance between the most distal point on the head and abdomen for three frames for each ground truth video. Dividing the mean pixel error by this length proxy yields a resolution-independent measure of pose-estimation performance, expressed as a percentage via:

$$error_{rel} = \frac{100}{n} \sum_{n} \frac{error_{px_n}}{bodylength} \tag{4}$$

We performed five-fold cross validation for all trained networks, with an 80/20 data split between training and withheld data.

4.9 Semantic segmentation

4.9.1 Benchmark data

Semantic and instance segmentation is used only rarely in non-human animals, partially due to the laborious process of curating sufficiently large annotated datasets. *replicAnt* can produce pixel-perfect segmentation maps with minimal manual effort. In order to assess the quality of the segmentations inferred by networks trained with these maps, semi-quantitative verification was conducted using a set of macro-photographs of *Leptoglossus zonatus* (Dallas, 1852) and *Leptoglossus phyllopus* (Linnaeus, 1767), provided by Prof. Christine Miller (University of Florida), and Royal Tyler (Bugwood.org; see Fig 6 d-f). For further qualitative assessment of instance segmentation, we used laboratory footage, and field photographs of *Atta vollenweideri* provided by Prof. Flavio Roces (Fig 6 g). More extensive quantitative validation was infeasible, due to the considerable effort involved in hand-annotating larger datasets on a per-pixel basis.

4.9.2 Synthetic data

We generated two synthetic datasets from a single 3D scanned *Leptoglossus zonatus* (Dallas, 1852) specimen: one using the default pipeline, and one with additional plant assets, spawned by three dedicated scatterers. The plant assets were taken from the Quixel library and include 20 grass and 11 fern and shrub assets. Two dedicated grass scatterers were configured to spawn between 10,000 and 100,000 instances; the fern and shrub scatterer spawned between 500 to 10,000 instances. A total of 10,000 samples were generated for each sub dataset, leading to a combined dataset comprising 20,000 image render and ID passes. The addition of plant assets was necessary, as many of the macro-photographs also contained truncated plant stems or similar fragments, which networks trained on the default data struggled to distinguish from insect body segments. The ability to simply supplement the asset library underlines one of the main strengths of *replicAnt*: training data can be tailored to specific use cases with minimal effort.

For an additional qualitative demonstration of instance and semantic segmentation, we use the image render and ID passes of the *Atta vollenweideri* "group" dataset also used for detection (Section XX, Fig 2 a-c).

4.9.3 Network training

We trained three different semantic segmentation networks: Mask-R-CNN[6], UperNet with SWIN Transformers[52], and PSPNet[53]. All networks use ResNet101 backbones, and were pre-trained on ImageNet[9]. We used the official Matterport implementation of Mask-R-CNN[6]. For both PSPNet and UperNet training, we use the MMSegmentation[57] implementation, favoured for its versatility and comprehensive architecture support. The generated synthetic data was converted to match the required file format and folder structure conventions, using separate data parsers for the COCO (Common Objects in COntext) and MMSegmentation annotations (see GitHub).

In order to leverage Mask-R-CNN's ability to provide both semantic and instance segmentations, we use the COCO data parser to train a separate Mask-R-CNN model with a ResNet101 backbone, using otherwise identical training parameters and the "group" dataset used also for detection and tracking (see Results — Detection). In all cases, the segmentation problem is treated as a binary task: class label of zero or unity are assigned to pixels or polygon areas attributed to the background or any subject, respectively. All networks were trained for a total of 160 epochs, and all training and evaluation of was performed on a desktop workstation with a 14 core CPU, 64 GB of RAM, and a NVIDIA RTX 2080 Ti GPU with 11 GB of VRAM. The full configuration and training schedule is provided in the SI.

4.9.4 Evaluation

To provide an indicative quantitative performance metric, we hand-annotated binary masks for the images shown in Fig 6 d-e, and computed the Average Class-wise Recall (ACR):

$$ACR = \frac{100}{c} \sum_{i}^{c} \sum_{j}^{n} \frac{px_j - (FP_{i,j} + FN_{i,j})}{px_j} \tag{5}$$

Here, FP is the number of false positives – the number of pixels falsely attributed to a class; FN is the number of false negatives – the number of pixels falsely attributed to other classes; and px is the total number of pixels in the ground truth mask for each class c. The resulting value is multiplied by 100 to provide a percentage score which quantifies the fraction of correctly identified pixels.

References

[1] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, ??? (2016)

[2] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A System for Large-Scale Machine Learning. In: 12th Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283. {USENIX} Association, Savannah, GA (2016). https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[3] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 1–14 (2015) arXiv:arXiv:1409.1556v6

[4] Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y.M.: YOLOv4: Optimal Speed and Accuracy of Object Detection (2020) arXiv:2004.10934

[5] Insafutdinov, E., Pishchulin, L., Andres, B., Andriluka, M., Schiele, B.: Deepercut: A deeper, stronger, and faster multi-person pose estimation model. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **9910 LNCS**, 34–50 (2016) https://doi.org/10.1007/978-3-319-46466-4_3 arXiv:1605.03170

[6] He, K., Gkioxari, G., Dollár, P., Girshick, R., Dollar, P., Girshick, R.: Mask R-CNN. Proceedings of the IEEE International Conference on Computer Vision **2017-Octob**, 2980–2988 (2017) https://doi.org/10.1109/ICCV.2017.322 arXiv:arXiv:1703.06870v3

[7] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., Chen, M.: Hierarchical Text-Conditional Image Generation with CLIP Latents (Figure 3) (2022)

arXiv:2204.06125

[8] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-Resolution Image Synthesis With Latent Diffusion Models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10684–10695 (2022)

[9] Krizhevsky, A., Sutskever, I., Hinton., G.E.: Imagenet classification with deep convolutional neural networks. In Advances in neural information. Advances in neural information processing systems, 1097–1105 (2012) arXiv:1102.0183

[10] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **8693 LNCS**(PART 5), 740–755 (2014) https://doi.org/10.1007/978-3-319-10602-1_48 arXiv:1405.0312

[11] Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical report (2009)

[12] Dell, A.I., Bender, J.A., Branson, K., Couzin, I.D., Polavieja, G.G., Noldus, L.P.J.J., Pérez-Escudero, A., Perona, P., Straw, A.D., Wikelski, M., Brose, U.: Automated image-based tracking and its application in ecology. Trends in Ecology and Evolution **29**(7), 417–428 (2014) https://doi.org/10.1016/j.tree.2014.05.004

[13] Valletta, J.J., Torney, C., Kings, M., Thornton, A., Madden, J.: Applications of machine learning in animal behaviour studies. Animal Behaviour **124**, 203–220 (2017) https://doi.org/10.1016/j.anbehav.2016.12.005

[14] Høye, T.T., Ärje, J., Bjerge, K., Hansen, O.L.P., Iosifidis, A., Leese, F., Mann, H.M.R., Meissner, K., Melvad, C., Raitoharju, J.: Deep learning and computer vision will transform entomology. Proceedings of the National Academy of Sciences of the United States of America **118**(2), 1–10 (2021) https://doi.org/10.1073/PNAS.2002545117

[15] Mathis, A., Mamidanna, P., Cury, K.M., Abe, T., Murthy, V.N., Mathis, M.W., Bethge, M.: DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. Nature Neuroscience **21**(9), 1281–1289 (2018) https://doi.org/10.1038/s41593-018-0209-y

[16] Pereira, T.D., Tabris, N., Matsliah, A., Turner, D.M., Li, J., Ravindranath, S., Papadoyannis, E.S., Normand, E., Deutsch, D.S., Wang, Z.Y., McKenzie-Smith, G.C., Mitelut, C.C., Castro, M.D., D'Uva, J., Kislin, M., Sanes, D.H., Kocher, S.D., Wang, S.S.H., Falkner, A.L., Shaevitz, J.W., Murthy, M.: SLEAP: A deep learning system for multi-animal pose tracking. Nature Methods **19**(4), 486–495 (2022) https://doi.org/10.1038/s41592-022-01426-1

[17] Graving, J.M., Chae, D., Naik, H., Li, L., Koger, B., Costelloe, B.R., Couzin, I.D.: Fast and robust animal pose estimation. bioRxiv, 620245 (2019) https://doi.org/10.1101/620245

[18] Hsu, A.I., Yttri, E.A.: B-SOiD, an open-source unsupervised algorithm for identification and fast prediction of behaviors. Nature Communications **12**(1), 1–13 (2021) https://doi.org/10.1038/s41467-021-25420-x

[19] Pereira, T.D., Aldarondo, D.E., Willmore, L., Kislin, M., Wang, S.S.H., Murthy, M., Shaevitz, J.W.: Fast animal pose estimation using deep neural networks. Nature Methods **16**(1), 117–125 (2019) https://doi.org/10.1038/s41592-018-0234-5

[20] Nath, T., Mathis, A., Chen, A.C., Patel, A., Bethge, M., Mathis, M.W.: Using DeepLabCut for 3D markerless pose estimation across species and behaviors. Nature Protocols **14**(7), 2152–2176 (2019) https://doi.org/10.1038/s41596-019-0176-0

[21] Minakshi, M., Bharti, P., Bhuiyan, T., Kariev, S., Chellappan, S.: A framework based on deep neural networks to extract anatomy of mosquitoes from images. Scientific Reports **10**(1), 1–10 (2020) https://doi.org/10.1038/s41598-020-69964-2

[22] Arent, I., Schmidt, F.P., Botsch, M., Dürr, V.: Marker-Less Motion Capture of Insect Locomotion With Deep Neural Networks Pre-trained on Synthetic Videos. Frontiers in Behavioral Neuroscience **15**(April), 1–12 (2021) https://doi.org/10.3389/fnbeh.2021.637806

[23] Bjerge, K., Mann, H.M.R., Høye, T.T.: Real-time insect tracking and monitoring with computer vision and deep learning. Remote Sensing in Ecology and Conservation **8**(3), 315–327 (2022) https://doi.org/10.1002/rse2.245

[24] Perez, L., Wang, J.: The Effectiveness of Data Augmentation in Image Classification using Deep Learning (2017) https://doi.org/10.1109/IGARSS.2016.7730324 arXiv:1712.04621

[25] Dyk, D.A., Meng, X.-L.: The Art of Data Augmentation. Journal of Computational and Graphical Statistics **10**(1), 1–50 (2001) https://doi.org/10.1198/10618600152418584

[26] Martinez-Gonzalez, P., Oprea, S., Castro-Vargas, J.A., Garcia-Garcia, A., Orts-Escolano, S., Garcia-Rodriguez, J., Vincze, M.: UnrealROX+: An Improved Tool for Acquiring Synthetic Data from Virtual 3D Environments. Proceedings of the International Joint Conference on Neural Networks **2021-July** (2021) https://doi.org/10.1109/IJCNN52387.2021.9534447 arXiv:2104.11776

[27] Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., Birchfield, S.: Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects

(CoRL), 1–11 (2018) arXiv:1809.10790

[28] Lambrecht, J., Kastner, L.: Towards the usage of synthetic data for marker-less pose estimation of articulated robots in RGB images. 2019 19th International Conference on Advanced Robotics, ICAR 2019, 240–247 (2019) https://doi.org/10.1109/ICAR46387.2019.8981600

[29] Kar, A., Prakash, A., Liu, M.Y., Cameracci, E., Yuan, J., Rusiniak, M., Acuna, D., Torralba, A., Fidler, S.: Meta-sim: Learning to generate synthetic datasets. Proceedings of the IEEE International Conference on Computer Vision **2019-Octob**, 4550–4559 (2019) https://doi.org/10.1109/ICCV.2019.00465 arXiv:1904.11621

[30] Kong, Y., Fu, Y.: Human Action Recognition and Prediction: A Survey **13**(9) (2018) arXiv:1806.11230

[31] Doersch, C., Zisserman, A.: Sim2real transfer learning for 3D human pose estimation: Motion to the rescue. Advances in Neural Information Processing Systems **32**(NeurIPS) (2019) arXiv:1907.02499

[32] Varol, G., Romero, J., Martin, X., Mahmood, N., Black, M.J., Laptev, I., Schmid, C.: Learning from synthetic humans. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 **2017-Janua**, 4627–4635 (2017) https://doi.org/10.1109/CVPR.2017.492 arXiv:1701.01370

[33] Rao, Q., Frtunikj, J.: Deep learning for self-driving cars: Chances and challenges: Extended Abstract. Proceedings - International Conference on Software Engineering, 35–38 (2018) https://doi.org/10.1145/3194085.3194087

[34] Prakash, A., Boochoon, S., Brophy, M., Acuna, D., Cameracci, E., State, G., Shapira, O., Birchfield, S.: Structured domain randomization: Bridging the reality gap by context-aware synthetic data. Proceedings - IEEE International Conference on Robotics and Automation **2019-May**, 7249–7255 (2019) https://doi.org/10.1109/ICRA.2019.8794443 arXiv:1810.10093

[35] Sakaridis, C., Dai, D., Van Gool, L.: Semantic Foggy Scene Understanding with Synthetic Data. International Journal of Computer Vision **126**(9), 973–992 (2018) https://doi.org/10.1007/s11263-018-1072-8 arXiv:1708.07819

[36] Greff, K., Belletti, F., Beyer, L., Doersch, C., Du, Y., Duckworth, D., Fleet, D.J., Gnanapragasam, D., Golemo, F., Herrmann, C., Kipf, T., Kundu, A., Lagun, D., Laradji, I., Hsueh-Ti, Liu, Meyer, H., Miao, Y., Nowrouzezahrai, D., Oztireli, C., Pot, E., Radwan, N., Rebain, D., Sabour, S., Sajjadi, M.S.M., Sela, M., Sitzmann, V., Stone, A., Sun, D., Vora, S., Wang, Z., Wu, T., Yi, K.M., Zhong, F., Tagliasacchi, A.: Kubric: A scalable dataset generator (2022) arXiv:2203.03570

[37] Deane, J., Kearney, S., Kim, K.I., Cosker, D.: DynaDog+T: A Parametric Animal Model for Synthetic Canine Image Generation (2021) arXiv:2107.07330

[38] Biggs, B., Roddick, T., Fitzgibbon, A., Cipolla, R.: Creatures Great and SMAL: Recovering the Shape and Motion of Animals from Video vol. 11365 LNCS, pp. 3–19. Springer, ??? (2019). https://doi.org/10.1007/978-3-030-20873-8_1 . http://dx.doi.org/10.1007/978-3-030-20873-8{_}1

[39] Fangbemi, A.S., Lu, Y.F., Xu, M.Y., Luo, X.W., Rolland, A., Raissi, C.: ZooBuilder: 2D and 3D Pose Estimation for Quadrupeds Using Synthetic Data. 19th ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2020, SCA 2020 - Showcases **39**(8), 1–2 (2020) arXiv:2009.05389

[40] Plum, F., Labonte, D.: scAnt — An open-source platform for the creation of 3D models of arthropods (and other small objects). PeerJ **9**(851705) (2021) https://doi.org/10.7717/peerj.11155

[41] Irschick, D.J., Martin, J., Siebert, U., Kristensen, J.H., Madsen, P.T., Christiansen, F.: Devices and Methods for Rapid 3D Photo-Capture and Photogrammetry of Small Reptiles and Amphibians in the Laboratory and the Field. Marine Mammal Science **37**(2), 482–491 (2021) https://doi.org/10.1111/mms.12759

[42] Irschick, D.J., Christiansen, F., Hammerschlag, N., Martin, J., Madsen, P.T., Wyneken, J., Brooks, A., Gleiss, A., Fossette, S., Siler, C., Gamble, T., Fish, F., Siebert, U., Patel, J., Xu, Z., Kalogerakis, E., Medina, J., Mukherji, A., Mandica, M., Zotos, S., Detwiler, J., Perot, B., Lauder, G.: 3D visualization processes for recreating and studying organismal form. iScience **25**(9), 104867 (2022) https://doi.org/10.1016/j.isci.2022.104867

[43] Zuffi, S., Kanazawa, A., Jacobs, D., Black, M.J.: 3D menagerie: Modeling the 3D shape and pose of animals. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 **2017-Janua**, 5524–5532 (2017) https://doi.org/10.1109/CVPR.2017.586 arXiv:1611.07700

[44] Plum, F., Lenz, S.: BlenderMotionExport. GitHub (2021). https://github.com/FabianPlum/blenderMotionExport

[45] Plum, F.: OmniTrax. GitHub (2023). https://github.com/FabianPlum/OmniTrax

[46] Redmon, J., Farhadi, A.: YOLOv3: An Incremental Improvement (2018) arXiv:1804.02767

[47] Kuhn, H.W.: The Hungarian method for the assignment problem. Naval Research Logistics (NRL) **52** (1955)

[48] Bernardin, K., Stiefelhagen, R.: Evaluating Multiple Object Tracking Performance : The CLEAR MOT Metrics. Eurasip Journal on Image and Video Processing **2008** (2008) https://doi.org/10.1155/2008/246309

[49] Walter, T., Couzin, I.D.: Trex, a fast multi-animal tracking system with markerless identi cation, and 2d estimation of posture and visual elds. eLife **10**, 1–73 (2021) https://doi.org/10.7554/eLife.64000

[50] Lauer, J., Zhou, M., Ye, S., Menegas, W., Schneider, S., Nath, T., Rahman, M.M., Di Santo, V., Soberanes, D., Feng, G., Murthy, V.N., Lauder, G., Dulac, C., Mathis, M.W., Mathis, A.: Multi-animal pose estimation, identification and tracking with DeepLabCut. Nature Methods **19**(4), 496–504 (2022) https://doi.org/10.1038/s41592-022-01443-0

[51] Romero-Ferrero, F., Bergomi, M.G., Hinz, R.C., Heras, F.J.H., Polavieja, G.G.: Idtracker.Ai: Tracking All Individuals in Small or Large Collectives of Unmarked Animals. Nature Methods **16**(2), 179–182 (2019) https://doi.org/10.1038/s41592-018-0295-5

[52] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 9992–10002 (2021). https://doi.org/10.1109/ICCV48922.2021.00986

[53] Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J., Limited, S.G.: Pyramid Scene Parsing Network (2017) arXiv:arXiv:1612.01105v2

[54] Sun, C., Shrivastava, A., Singh, S., Gupta, A.: Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. Proceedings of the IEEE International Conference on Computer Vision **2017-Octob**, 843–852 (2017) https://doi.org/10.1109/ICCV.2017.97 arXiv:1707.02968

[55] McHenry, M.J., Hedrick, T.L.: The science and technology of kinematic measurements in a century of Journal of Experimental Biology. The Journal of experimental biology **226**(1) (2023) https://doi.org/10.1242/jeb.245147

[56] Yuan, S., Garcia-Hernando, G., Stenger, B., Moon, G., Chang, J.Y., Lee, K.M., Molchanov, P., Kautz, J., Honari, S., Ge, L., Yuan, J., Chen, X., Wang, G., Yang, F., Akiyama, K., Wu, Y., Wan, Q., Madadi, M., Escalera, S., Li, S., Lee, D., Oikonomidis, I., Argyros, A., Kim, T.K.: Depth-Based 3D Hand Pose Estimation: From Current Achievements to Future Goals. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2636–2645 (2018) https://doi.org/10.1109/CVPR.2018.00279 arXiv:1712.03917

[57] Contributors, M.: MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark. \url{https://github.com/open-mmlab/mmsegmentation} (2020)

[58] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine

Learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[59] Karashchuk, P., Rupp, K.L., Dickinson, E.S., Azim, E., Brunton, B.W., Tuthill, J.C., Rupp, K.L., Dickinson, E.S., Walling-bell, S., Sanders, E., Azim, E.: Anipose : A toolkit for robust markerless 3D pose estimation. CellReports **36**(13), 109730 (2021) https://doi.org/10.1016/j.celrep.2021.109730

Supplementary information.
All produced code, documentation, and software releases are hosted via GitHub: https://github.com/evo-biomech/replicAnt

All datasets, both generated and real, additional SI, and the best performing networks are hosted via Zenodo:

- 3D Models https://zenodo.org/record/7849059 **DOI** : 10.5281/zenodo.7849059
- Detection and Tracking Datasets and Trained networks https://zenodo.org/record/7849417 **DOI** : 10.5281/zenodo.7849417
- Pose-Estimation Datasets and Trained networks https://zenodo.org/record/7849596 **DOI** : 10.5281/zenodo.7849596
- Semantic And Instance Segmentation Datasets and Trained networks https://zenodo.org/record/7849570 **DOI** : 10.5281/zenodo.7849570

Acknowledgments.
This study received funding from Imperial College's President's PhD Scholarship (to Fabian Plum), and is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant agreement No. 851705, to David Labonte). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Appendix A Detection and tracking datasets

Table A1 Benchmark *Atta* detection datasets

dataset	samples[1]	recording settings
base	1,000	camera = OpenCV OAK-D fps = 25 shutter speed = 1/50 s image size (px) = 1920 x 1080 individuals = 81 notes: well exposed, no motion blur,clean background
bright	1,000	camera = OpenCV OAK-D fps = 25 shutter speed = 1/25 s image size (px) = 1920 x 1080 individuals = 103 notes: highly overexposed, motion blur, clean background
dark	1,000	camera = OpenCV OAK-D fps = 25 shutter speed = 1/200 s image size (px) = 1920 x 1080 individuals = 65 notes: under exposed, extreme rolling shutter and compression artefacts, clean back-ground
noisy	1,000	camera = OpenCV OAK-D fps = 25 shutter speed = 1/50 s image size (px) = 1920 x 1080 individuals = 61 notes: well exposed, no motion blur, highly textured background
close	1,000	camera = OpenCV OAK-D fps = 25 shutter speed = 1/50 s image size (px) = 1920 x 1080 individuals = 36 notes: close-up top-down footage, slightly out-of-focus, clean background
all	6,000	camera = OpenCV OAK-D fps = 25 image size (px) = 1920 x 1080 individuals = 36 to 103 notes: containing all real samples

[1]Using 5-fold cross-validation, 20% of the samples are withheld during training.

Table A2 Synthetic *Atta* detection datasets

dataset	samples[1]	generator settings
multi ant (S, M, L)	100 (S) 1,000 (M) 10,000 (L)	individuals = 100 scale variation = 33% subject meshes = 3 image size (px) = 1024 x 1024 compression rand. = True new terrain every = 15 new scatterers every = 10 new subject placement every = 5
single ant	10,000	individuals = 100 scale variation = 90% subject meshes = 1 image size (px) = 1024 x 1024 compression rand. = True new terrain every = 15 new scatterers every = 10 new subject placement every = 5

[1]Using 5-fold cross-validation, 20% of the samples are withheld during training.

Table A3 Augmented / mixed *Atta* detection datasets

dataset	samples[1]	recording settings
base (augmented)	1,000	camera = OpenCV OAK-D fps = 25 shutter speed = 1/50 s image size (px) = 1920 x 1080 individuals = 81 notes: background replacement of base dataset using thresholding and randomly drawn ImageNet 2019 samples[9]
bs10 − 10 real /1 synth	1,100	notes: combining the (synthetic) multi-ant dataset with the (real) base dataset in various ratios
bs100 − 1 real / 1 synth	2,000	
bs1000 − 1 real / 10 synth	11,000	
sb5 − 1 real / 50 synth	10,500	
sb1 − 1 real / 100 synth	10,100	

[1]Using 5-fold cross-validation, 20% of the samples are withheld during training.

Table A4 *Gnathamitermes* detection datasets

dataset	data type	samples[1]	generator settings
termites S	synthetic	10,000	individuals = 100 scale variation = 50% subject meshes = 2 image size (px) = 1024 x 1024 compression rand. = True new terrain every = 15 new scatterers every = 10 new subject placement every = 1
termites R	real	10,000	camera = Nikon D810, 105 mm Nikkor lens fps = 30 shutter speed = 1/200 s ISO = 200 image size (px) = 3840 x 2160 individuals = 76 notes: filmed in the dessert, without controlled lighting conditions, daylight, highly textured background

[1]Using 5-fold cross-validation, 20% of the samples are withheld during training.

Appendix B Pose-estimation datasets

Table B5 Benchmark pose-estimation datasets

dataset	samples[1]	generator settings
top-down (platform)	179	camera = FLIR Blackfly, 18 mm lens fps = 55 shutter speed = 1/200 s image size (px) = 2048 x 1536 individuals = 1 notes: well exposed, no motion blur, static camera
left view (platform)	345	camera = FLIR Blackfly, 18 mm lens fps = 55 shutter speed = 1/200 s image size (px) = 2048 x 1536 individuals = 1 notes: well exposed, no motion blur, static camera
right view (platform)	381	camera = FLIR Blackfly, 18 mm lens fps = 55 shutter speed = 1/200 s image size (px) = 2048 x 1536 individuals = 1 notes: well exposed, no motion blur, static camera
all (platform)	805	camera = FLIR Blackfly, 18 mm lens fps = 55 shutter speed = 1/200 s image size (px) = 2048 x 1536 individuals = 1 notes: combining all platform views
handheld	200	camera = Huawei P20 Pro, Leica fps = 25 shutter speed = 1/50 s image size (px) = 1440 x 720 to 1920 x 1080 individuals = 1 notes: handheld phone camera, frequent occlusions, camera shake, harsh shadows

[1]Using 5-fold cross-validation, 20% of the samples are withheld during training.

Table B6 Synthetic pose-estimation datasets

dataset	samples[1]	generator settings
single stick	10,000	individuals = 1 scale variation = 70 subject meshes = 1 image size (px) = 2048 x 2048 compression rand. = True new terrain every = 10 new scatterers every = 5 new subject placement every = 1
multi stick	10,000	individuals = 5 scale variation = 70 subject meshes = 1 image size (px) = 2048 x 2048 compression rand. = True new terrain every = 10 new scatterers every = 5 new subject placement every = 1

[1]Using 5-fold cross-validation, 20% of the samples are withheld during training.

Table B7 Augmented / mixed *Atta* detection datasets

dataset	samples[1]	recording settings
handheld (refine) 12.5% real 25% real 50% real 100% real	 10,000 + 25 10,000 + 50 10,000 + 100 10,000 + 200	notes: combining the (synthetic) multi-ant dataset with the (real) base dataset in various ratios
platform (refine) 10 samples per camera	10,000 + 50	notes: combining the (synthetic) multi-ant dataset with the (real) base dataset in various ratios

[1]Using 5-fold cross-validation, 20% of the samples are withheld during training.

Appendix C Semantic segmentation datasets

Table C8 Synthetic pose-estimation datasets

dataset	samples[1]	generator settings
Leptoglossus (default)	10,000	individuals = 20 scale variation = 33 subject meshes = 1 image size (px) = 2048 x 2048 compression rand. = True new terrain every = 10 new scatterers every = 5 new subject placement every = 1 notes: using default scatterers
Leptoglossus (plants)	10,000	individuals = 20 scale variation = 33 subject meshes = 1 image size (px) = 2048 x 2048 compression rand. = True new terrain every = 10 new scatterers every = 5 new subject placement every = 1 notes: adding asset scatterers using plant assets from Quixel library including 20 grass and 11 fern and shrub assets

[1]Using 5-fold cross-validation, 20% of the samples are withheld during training.

Appendix D Insect rigging convention

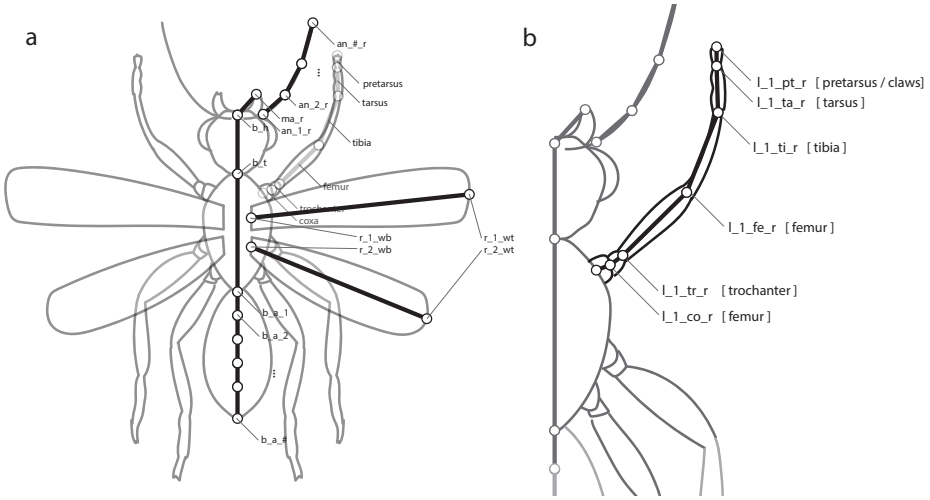

Fig. D1 A standardised rig forms the basis of insect model actuation within the *replicAnt* pipeline. (a) As all insects follow the same fundamental body layout, we have created a standardised rig that is used to actuate all insect models inside the *replicAnt* generator pipeline. The rig can be extended to provide further articulation, such as additional segmentation or additional appendages, e.g., articulated proboscis. (b) The employed Inverse Kinematics Solver (IKS) within *replicAnt* expects the depicted rigging and naming convention to use our provided animation blueprints. We provide additional documentation and a template Blender project file on out GitHub page to simplify the rigging process for new users. (https://github.com/evo-biomech/replicAnt)

Appendix E Glossary

Armature An Armature is a collection of virtual bones and joints which define how the the mesh deforms as pose changes. The process of assigning an armature to a mesh is referred to as rigging.

Asset An element from a library of pre-existing meshes placed in the generated 3D environment.

Buffer temporarily stored visual information, in this context generated 2D texture maps.

Decal A generated 2D material, projected onto multiple assets in the 3D environment; albedo, alpha, roughness, metalness, and normal maps tie the appearance of distinct assets together.

Iteration An iteration describes a single step of a procedurally generated dataset. Its output is a collection of associated passes and labels.

Label A label contains the information found in a corresponding image sample in ordered text form, including bounding boxes, IDs, as well as 2D and 3D joint locations.

Mesh A 3D object defined by polygons.

Network Various deep neural networks for computer vision applications such as classification, detection, tracking, 2D and 3D pose estimation, or semantic segmentation.

Parser A program written to interpret a given input and translate it into a different format. We use parsers extensively to convert generated data (image passes and labels) into formats readable by various deep learning frameworks.

Pass A pass (or render pass) is a 2D image containing either the rendered camera view, segmentation data, depth data, or normal data in object or world space.

Subject A subject is an imported, rigged mesh which is part of the generated population, i.e., a group of mesh-es for which we export annotated data.

Texture maps :

Albedo The albedo or base colour map contains the colour information of a material in Physically Based Rendering (PRB) workflows

Alpha The alpha map controls the opacity of specific areas of the material.

Depth The depth or displacement map is a single channel texture which describes (large) height changes in the mesh topology. These maps are usually combined with a weight and a bias to influence the magnitude and base height of the resulting displacement.

Metalness There are two types of reflection models used in PBR materials: dielectric and conductive. The metalness map functions as a mask to locally blend between the two.

normal The normal map contains small scale surface detail information to increase the perceived complexity of the mesh's appearance without increasing polygon count. Normal maps consist of three colour channels to encode local position information of the surface normal in X, Y, and Z respectively (see also displacement maps).

Roughness The roughness map controls the reflectivity of a material, i.e.it controls transitions between diffuse and glossy appearance.