HHS Public Access

Author manuscript

Nat Methods. Author manuscript; available in PMC 2021 October 05.

Published in final edited form as:

Nat Methods. 2021 April ; 18(4): 378–381. doi:10.1038/s41592-021-01103-9.

A 3D virtual mouse generates synthetic training data for behavioral analysis

Luis A. Bolaños[1,2,†], **Dongsheng Xiao**[1,2,†], **Nancy L. Ford**[4], **Jeff M. LeDue**[1,2], **Pankaj K. Gupta**[1,2], **Carlos Doebeli**[1,2], **Hao Hu**[1,2], **Helge Rhodin**[3], **Timothy H. Murphy**[1,2,*]

[1]Department of Psychiatry, Department of Oral Biological and Medical Sciences, University of British Columbia, Vancouver, British Columbia, Canada, V6T 1Z3

[2]Djavad Mowafaghian Centre for Brain Health, Department of Oral Biological and Medical Sciences, University of British Columbia, Vancouver, British Columbia, Canada, V6T 1Z3

[3]Department of Computer Science, Department of Oral Biological and Medical Sciences, University of British Columbia, Vancouver, British Columbia, Canada, V6T 1Z3

[4]Centre for High-Throughput Phenogenomics, Department of Oral Biological and Medical Sciences, University of British Columbia, Vancouver, British Columbia, Canada, V6T 1Z3

Abstract

We develop a 3D synthetic animated mouse based on CT scans that is actuated using animation and semi-random, joint-constrained movements to generate synthetic behavioural data with ground-truth label locations. Image-domain translation produced realistic synthetic videos used to train 2D and 3D pose estimation models with accuracy similar to typical manual training datasets. The outputs from the 3D model-based pose estimation yielded better definition of behavioral clusters than 2D videos and may facilitate automated ethological classification.

While powerful, feature tracking methods such as DeepLabCut (DLC)[1] require substantial user effort to label training images. We propose an approach using synthetic video data, where annotation data is readily available both in 2D and 3D space (Supplementary Video 1). We facilitate such an approach with a synthetic animated mouse (SAM), an open-source, high-resolution, realistic model of the C57BL/6 adult female mouse skeleton, skin, whiskers, and fur (Fig. 1a, b, Supplementary Video 2)[2]. We extend previous digital mouse models[3,4] by using CT scan images and introducing realistic skin and fur on an animation-ready model. Synthetic data has unique advantages[5] including the diversity, quantity, and quality of labels which are unbiased and placed based on ground-truth positions from a 3D model's

*Correspondence to: Timothy H Murphy thmurphy@mail.ubc.ca.

Author contributions statement: LAB, THM, JML, and DX developed the synthetic mouse concept. LAB generated the mouse model and all main figure animations, CD made supplemental models. LAB, THM, JML, and DX wrote the manuscript, HR edited the manuscript and advised on algorithm selection and validation statistics, DX collected experimental data and performed data analysis. HH, LAB, DX, and PKG performed modelling and data analysis. NLF collected the CT scans and advised on model selection and resolution.

†L.A.B. and D.X. are co-first authors.

Competing financial interests: The authors declare no competing financial interests.

anatomy and not reliant on rater inference[6]. We create short animation sequences of the mouse in repeated and randomized poses, providing a potentially infinite dataset where ground-truth body part labels are accessible from multiple camera angles (Fig. 1c, d). We introduced additional image diversity by adding lighting, and camera noise. The model in Blender uses a 3D environment, allowing placement of multiple virtual cameras providing training data for multi-camera pose estimation systems with shared ground-truth labels (Fig. 1d, Supplementary Video 1 and 2).

We generate synthetic data of common mouse behavioural experiments (Supplementary Videos 3, 4, and 5) as well as 3D models for freely moving and head-fixed animals during International Brain Lab (IBL)[7] and water reaching tasks[8] (Supplementary Videos 6 and 7), and for pellet-reaching mice[9] (Fig. 1c,d). We recreated the scenes in Blender to resemble the setups, including lighting and cameras, to match the real videos, manually generated short animation sequences (Supplementary Table 1) and rendered the images (Fig. 1c).

While initial renders captured most aspects of a crude mouse form, including skin and fur, pose estimators require a closer match between the training set and the real experimental setup. Accordingly, we used image domain transformation to transfer style (textures, lighting, and background), from real mouse videos onto the animation (Fig. 2a, Supplementary Videos 3–6). We use a single U-GAT-IT [10] image domain transformation model (Supplementary Table 1) for each mouse using the first trained model as pre-trained weights for subsequent models, and observe accurate transfer of the subsequent mouse style within the first few thousand iterations. Such synthetic approaches provide an infinite set of images for training while ensuring that the initial pose from the original animation remained largely unaffected. Through randomization, the poses generated spanned a larger space than real recordings alone, adding more diversity to the data (Fig. 2b Supplementary Videos 3–6.

We compute the structural similarity index measure (SSIM)[12] to assess differences between real, animated and synthetic images (after style transfer). Synthetic images were more similar to real videos than animation alone (Fig. 2c). The use of individual mouse U-GAT-IT models further increased the SSIM score compared to the use of a single aggregate model. We observed that the SSIM scores for the paired (same mouse) synthetic-real images were higher than for both the unpaired mouse sets and the aggregate model (mean = 0.543). Furthermore, SSIM scores for the comparison of real mouse images with other real mouse images from another mouse (real unpaired) resulted in lower SSIM scores than for all synthetic approaches. Pairing real images of single mice with other images of themselves (real paired) resulted in the highest SSIM scores given some frames would closely match, however the bulk of the distribution overlapped with the paired synthetic-real condition (mean = 0.696). For further evaluation we used Kernel Inception Distance (KID)[13]. The lower the KID, the more shared features between real and generated images. U-GAT-IT style transfer achieved similar KID scores when comparing real with real images and real with synthetic images on paired mice (Fig. 2d).

Using our realistic synthetic mouse videos, we trained a DLC model for 2D pose estimation using synthetic training data. We define the markers within Blender, and export the pixel coordinates on the camera frame as a CSV file using a custom Python script (Fig. 2a)[14]. The

format follows the training label text defined in DLC[1] to easily transfer the marker coordinates to a DLC project. A benefit of synthetic data is the ability to export a large number of labels with no added cost. With this in mind, we chose to export 28 markers with half defining the spine and half defining the left limbs. We trained DLC using the 1000-frame animation after style transfer (5 mouse models in total or 5000 frames) using the resnet-50 model (Fig. 2a). We then applied the synthetically trained model to real video data (see '1 - Videos/Synthetic-DLC_Performance.mp4' at https://osf.io/h3ec5/)[2] and assessed its performance by comparing the model to multiple models trained on various subsets of a manually labeled 1,280-frame project. We used a 27,000 frame long video, composed of multiple recordings from multiple mice, as input to the manually and synthetically trained models. We computed the mean Euclidean distance error for each frame for each of the paw labels using the full 1,280-frame manual model as the ground truth. The value is calculated after each of the 2D predictions are passed through a linear regression to correct the differences in labelling between the synthetic model and the manual model. The results show that the synthetically trained model achieved comparable errors to manually labeled models with ~200 labeled frames (Fig. 2e). We then computed the mean Euclidean distance error for the other labels that could be paired (n=11). Overall the mean errors were on average 6.7 pixels from the manual model prediction on an image frame of 640 × 320 pixels (Supplementary Fig. 1).

We test two methods for 3D pose estimation: a multi-camera system using DLC 3D [15], and a single camera 2D-to-3D pose lifter [5,16]. We used in-house mouse behavioral running wheel data where calibration images could be acquired and camera specifications were known (Supplementary Video 8). We could consistently and accurately label markers in multi-camera systems using synthetic data and show similar DLC 3D performance to a project of manually labelled 2-camera real videos (Supplementary Fig. 2). Generation of synthetic data, while potentially equal in labour cost for a single camera, can quickly become substantially less laborious when dealing with multiple camera setups and/or large amounts of labels.

The second method for 3D pose estimation leverages the ability to extract ground-truth 3D information from the digital model. We tested the ability to "lift" 2D poses onto 3D space by training a linear neural network using the previously extracted 2D pixel coordinates and 3D coordinates relative to a reference marker also using the running wheel dataset [5,16] (Fig. 2f). We compared the lifted poses to the previously described DLC 3D model trained on synthetic data after matching the scale and applying a rigid transformation to align coordinate systems, and show an RMSE of 3.97 mm and 4.17 mm for camera 2 and 3 respectively (Supplementary Fig. 3 and Supplementary Video 8). We believe 3D pose estimation could allow researchers to analyze their data in a manner less dependent on initial camera position and other setup-specific factors. The transformation of 2D pose estimation onto this relative 3D space, independent of camera position and rotation, could also allow us to combine data from different sources where mice share inherent behaviours (such as grooming or reaching), but the resulting 2D coordinates vary from having a different camera perspective. While true 3D datasets are expected to contain more information than 2D videos[17] or projections, it was unclear whether the use of the synthetic mouse 3D model could improve pose classification. Accordingly, we compared 2D and 3D poses, and found

that 3D approaches yield better qualitative and quantitative measures of clustering than 2D poses (Hopkins[18] score: 0.02, 3D; versus 0.04, 2D) (Fig. 2g), providing incentive to track in 3D using multi-camera systems, or if impractical, using the single-camera pose lifter.

While we have shown that 3D information from synthetic data can improve clustering, it comes with a setup time for the initial rendering. However, by creating multiple pre-made animated models (Supplementary Videos 6 and 7) and future automation of movement generation using scripting we hope that this barrier can be reduced. Scripting could generate potentially a wider repertoire of training data for models that examine specific movements necessary for advanced phenotyping. The 3D model space could be thought of as a mouse common behavioral framework, representing movements in a space less subject to variation due to camera placement, lighting, and other scene-related factors than typical laboratory recording conditions.

Methods

Blender.

Blender (https://www.blender.org/download/) is a free open-source program for 3D computer graphics with features spanning the entire pipeline for creating animated films. We use Blender as the primary software to generate synthetic data by animating a digital mouse which we use to extract ground-truth information for machine learning models. The digital mouse was created within Blender and is animation-ready providing a foundation for the generation of synthetic data of behaving mice (Supplementary Video 1, https://osf.io/h3ec5/, note: Blender v2.83+ required to illustrate full model).

CT scan preprocessing and mouse behavioral videos.

All animal experiments were performed with institutional ethics approval. Animal protocols were approved by the University of British Columbia Animal Care Committee and conformed to the Canadian Council on Animal Care and Use guidelines and animals were housed under standard 12 h light cycle, 20–24 degrees C and 40–60% humidity. Adult male approximately 4 month old C57BL/6 mice were used for headfixed video imaging during a running wheel task or water reaching, other real mouse videos were obtained from open source datasets or as cited. Female C57BL/6 mice, aged 8–13 weeks, were used for CT imaging, with an expected weight range of 18–22 g (jax.org). Mice were group-housed in a conventional animal facility. Mice were anaesthetized throughout the micro-CT imaging session with isoflurane in O_2 (5% to induce and 1.5–2% for maintenance) delivered via a nose-cone. Datasets from 3 mice were used (containing the head, thorax and hind end) and tiled together into a composite image. *In vivo* micro-CT images of the thorax and hind end were acquired using an eXplore CT120 (Trifoil Imaging, Chatsworth, USA) with 80 kVp, 40 mA, and 4–8 minute scan times. The head image was acquired on a Locus Ultra (GE Healthcare, London, Canada) with 80 kVp, 50 mA and a 1 minute scan time. The images were reconstructed with a filtered backprojection algorithm and scaled into Hounsfield units. The thorax and hind end images were reconstructed with 0.1 mm isotropic voxel spacing, while the head image was reconstructed at 0.15 mm spacing. The CT scans were converted to high density meshes at two thresholds (corresponding to the skeleton and the skin) using

3DSlicer (https://www.slicer.org, Fig. 1a). These meshes were combined in Blender to produce a full body model of a mouse.

3D model creation.

We create a mouse model through retopology of the high density meshes, generating a single unified mesh with a lower density of quad polygons useful for deformation and animation. The 3D model was UV unwrapped, and a texture was manually drawn using various images of C57BL/6 mice as reference. The addition of particle systems to simulate fur increased the realism of the final mouse model, and materials were defined for both the Cycles and Eevee render engines within Blender.

Rigging of the model.

Rigging—the process of adding control bones for the deformation of the mesh— was guided by the skeletal model generated using the CT scans. Bone segments were added to create the entire skeleton of the mouse with the addition of various control bones for inverse kinematics and volume preservation. Volume preserving bones were added to improve skin deformation in large body regions during extreme poses (Fig. 1b), while inverse kinematic controls were added to all paws and skull bones to make animation quicker and easier. Deformation bones were also added to the ears, cheeks, and chest to enable the user to animate breathing and sniffing.

Synthetic data generation.

Scene setup.—For the generation of synthetic data, a new scene is opened in Blender to model the experimental setup of interest. From our experience, it is not necessary to model all objects in high detail. For the running-wheel experiment, the mouse is animated on top of a simple wheel model. In the case of pellet reaching, more objects were used as the mouse would interact with each of them (bar, pellet, food wheel). Camera objects are added and Blender allows a ‘background’ image to be added to the camera view. For this, an image of the experimental setup is imported and used to match the camera perspective to the real videos (using calibration images here helps speed up the process and ensures proper alignment in the case of multi-camera systems). We try to keep the alignment between the animation and the real videos as close as possible to prevent shifting of objects during style transfer which could compromise the integrity of marker positions on the image frame. The mouse model is copied into the new scene, placed and posed to match the silhouette of the real mouse. Finally, lighting is added to the scenes, again matching the real videos, and materials—properties which define how an object is shaded—are created for the new objects.

Animation.—A two-part strategy is employed in order to generate a large amount of simulated mouse behaviour from a minimalist set of manually animated sequences. The first part of the strategy uses the skeletal model described above to pose the synthetic mouse. Poses are manually defined, and keyframes are set to create the 3D animations, with Blender automatically interpolating between keyframes. In an effort to reduce the manual work these primary animations (for example of running, reaching and grabbing) consist of a small number of frames (Supplementary Table 1). The primary animations are combined and

looped to create a longer animation using the Non-Linear Animation stack in Blender. In creating longer animations noise is added to several properties (limb offsets, lighting intensity, and camera angles) to increase the amount of parameter space covered by the synthetic mouse behaviour and to avoid identical, repetitive sequences. In the case of the running wheel, two animation sequences were created: a 20-frame long sequence of mice in various poses (running, resting, and grooming) and a 28-frame loop of a mouse running (Supplementary Video 2). A similar approach was taken for the multi-camera scenes with added care to ensure alignment of the two camera views (Fig. 1c).

Importantly, Blender supports Python scripting which allows libraries of movements to be imported or used in a format that supports inverse kinematics where video data can be used with pose estimators leading to the control of the Blender model's skeleton and generation of movements within the model. Our approach also supports forward kinematics where the Blender model's skeleton can be directly controlled using rotation matrices. To show the flexibility of our Blender model, we transfer unique human movements from the CMU Motion Capture dataset, like upright walking to the mouse model (see 1 - Videos/MOCAP_humanoid-walk.mp4 and MOCAP_humanoid-jump.mp4 at https://osf.io/h3ec5/). The toolkit we used for reading the dataset is AMCParser (https://github.com/CalciferZh/AMCParser/).

Noise.—To generate diverse training data we add noise to the animation. We added various 'empty' objects to the scenes in Blender and set a single keyframe of either their position or rotation. Using the Graph Editor, we added noise modifiers such that their position or rotation attributes were randomly animated throughout the entire timeline sequence. These objects could then be referenced to modify existing animations by adding their procedural movements to the object or bone. Using the Bone Constraint or Object Constraint panels, we could reference the empty noise objects by adding either the Copy Transform or Copy Rotation modifiers and choosing the empty object of interest. The settings were changed such that the noise was added to the original position or rotation of the constrained object rather than replacing it, and the constraint's influence was changed to limit the amount of noise to prevent overstretching or clipping. The Influence property could also be animated in the original sequences to allow users to specify sections of the animation where a limb or object could not deviate drastically from the original animation, for example at the end of a reaching sequence where the paw must be grabbing the pellet (though, other properties such as the other paw, elbow rotation, and digit spread could still have the noise added). For lighting changes, we either referenced a random position object and added its noise to change the direction of the light, or a keyframe could also be set for the intensity of the light which would have its own noise modifier.

Rendering and data export.—Once the animation is complete, rendering is done using the Eevee render engine as it is significantly faster than Cycles, but at a loss of realism which is compensated for by style transfer in the subsequent image domain translation step. We saved renders as .png files for lossless compression while saving current render progress if a system crash were to occur.

To export label coordinates, new empty objects are created and named accordingly. These empty objects serve as a proxy to the marker of interest, using a Copy Location object constraint modifier to follow a bone or object of choice. The 'mCBF-2d-3d_marker-extraction.py' python script is opened in the text editor under the scripting panel within Blender, and the variables are changed to match the markers of interest. The script is run, iterating through all the frames and saving both 2D and 3D relative coordinates (https://github.com/ubcbraincircuits/mCBF).

Image domain translation: *U-GAT-IT.* As it is unlikely for the render to truly be hyper-realistic, realism is added through an image-domain translation model. We use a U-GAT-IT-PyTorch model for its high performance capabilities in style transfer tasks. The folders are set up using all the rendered images for both the trainA and testA datasets. trainB and testB datasets are composed of frames extracted from the behavioural videos of interest that will ultimately be analyzed. We use Blender's video editing tool to extract these frames as .png files. Due to hardware limitations, we use a cut-down version of the default U-GAT-IT-PyTorch model to train using a GTX 1070 with 8GB of vRAM. The parameters used in training are described in Supplementary Table 1. These parameters provided images at a high enough resolution with good accuracy in style transfer, while retaining the original render's pose. However, smaller details such as individual digits were much harder to transfer accurately while keeping the exact pose of the render. While it could be a limitation due to the cut-down parameters, continued development of style transfer methods may lead to improvements. Once the translated images are visually indistinguishable from the real images and pose is retained, we halt the training and run the entire test set through the model. For further quantitative evaluation for the performance of the U-GAT-IT model we employ the structural similarity index. To reduce training time for multiple mice, a single mouse U-GAT-IT model was trained for 300,000 iterations, and that model was used as pre-trained weights for the subsequent individual mouse models. Style transfer performance could potentially be further improved by widening the real dataset, creating recordings for the sole purpose of generating more data by changing camera angles and lighting. The more diverse the real training dataset is, the easier it is to transfer style during new poses or camera angles in the animation. Differences in age and weight of mice can be managed by a general scaling of the 3D model and letting the nuances in appearance be handled by the style transfer. We suggest that users perform 300,000 iterations of the U-GAT-IT algorithm and halt based on visual inspection and that these conditions typically yield high similarity using quantitative measures. We suggest monitoring SSIM and KID scores and using these as a method of optimizing the number of iterations as performance may vary based on the initial animation or training data. In our experience, more iterations did not always generate a more realistic result.

DeepLabCut.

We created a new DeepLabCut[1] (DLC) project and edited the config file to have the labels and skeleton of interest. We created new folders inside of the labeled-data folder corresponding to each synthetic dataset and copied the frames generated by U-GAT-IT, after being cropped and scaled to original video size, into the folder. We modified a custom script "generate_csv_h5.py" to point to the correct .csv exported by Blender (input) and the folder

previously created (output). We ran the script and generated a .csv and h5 file with the training data. This was done for each synthetic dataset. Once complete, we trained the DLC model. We trained 2D models using the resnet-50 model to 750,000 iterations and 3D models to 1,030,000 iterations again on resnet-50.

For benchmarking DLC 2D we applied the synthetically trained model (5,000 frames) to real video data and assessed its performance by comparing the model to multiple models trained on various subsets of a 1,280 frame, manually labelled project. For this testing, we discarded 13 frames of a 27,000 frame long video due to having no illumination or frames in which the 1,280-frame manual labeled model performed poorly. We calculated the Euclidean distance error after the 2D predictions underwent a linear regression to correct differences in labelling between the synthetic and manual models.

For both real and synthetic video multi-view 3D pose generation we employed DLC 3D [15] using mouse running wheel real multi-camera data or synthetic data generated for matched views.

Single camera 2D-to-3D pose lifting.

Using a custom Google Colab notebook with a GPU instance, we train a linear neural network [5,16] with a linear size of 1024 nodes, with 2 stages and a “p_dropout” of 0.667 to an error of $5x10^{-5}$ to $8x10^{-5}$ [14]. The training data is composed of the extracted pixel coordinates of all the labels as input, with the target as the 3D relative coordinates from Blender. The input pixel coordinates have random noise added in the range of 1 to 2 pixels to prevent the model from quickly over-fitting. Once trained, we tested the model on real videos using the predicted 2D poses from DLC and saved results as CSV files in a similar format to DLC’s result with an added dimension.

2D and 3D poses clustering.

To classify the 2D and 3D poses, we used the unsupervised clustering algorithm PhenoGraph[19]. Each 2D or 3D pose matrix with N rows (each containing all joints from a frame) is partitioned into clusters by a graph that represents their similarity. The graph is built in two steps, first it finds k nearest neighbors for each pose (using Euclidean distance) resulting in N sets of k nearest neighbors. In the second step, a weighted graph was built such that the weight between nodes depends on the number of neighbors they share. We then performed Louvain community detection[20] on this graph to partition the graph that maximizes modularity.

Time and labour cost:

In general, for a two-camera set up, users will need to optimize the location and angle of the Blender virtual cameras to fit the pose to the scene. This is an iterative process that consists of fitting the scene to one camera, then switching views and fitting the scene to the second camera, until the scene fits both camera views properly (calibration images can simplify this process). More time will be required to set up the scene if additional objects are needed. However, this can be improved with the use of previously created CAD models of common objects (such as optomechanical components from Thorlabs).

To create an animation, key frames must be set up every few frames and the mouse's pose must once again match both camera angles in each scene. The animation data can then be looped with added noise and rendered. Looping the data means that once the original animation is made, the user can create many times more data quickly with added variation. In contrast to manual labeling, more synthetic training data can be produced quickly and independently of the time that it took to create the initial animation. In addition, once the animation has been created, different lighting and new cameras can be set up for the same scene. Small changes such as mouse coat color or adding experimental components can also be made quickly with minimal manual work in Blender and no requirement for manual re-labeling of data. Since labels used for DLC training are specified directly with the Blender 3D environment its greatest advantage is in labeling features of the subject that would be difficult or impossible to manually annotate due to their sheer number or ambiguity about their location within 2D projections that are typically used for manual annotation. From experience, a novice user may spend around 15–20 hours on scene setup with 4–8 hours taken on animation and randomization (6–10 and 4–8h respectively for experienced users). Rendering times vary substantially in time depending on frame count and scene complexity. We found, for the mouse-wheel experiment, a single frame took approximately 8 s to render using Eevee, and 25 s using Cycles render engines on a Ryzen 1700x 8-core processor. U-GAT-IT took approximately 60 hours for 300,000 iterations on a GTX 1070 8Gb. These steps are automated and do not require user intervention or supervision. For comparison, an experienced DLC labeler spends ~8 hours to label 208 frames (28 markers for each frame) but this number can quickly grow in proportion to the number of cameras and labels. The labour time investment to produce a workable Blender model for an experiment corresponds to the labelling of between 494 to 768 frames for a novice user and 260–468 frames for an experienced user. While the added utility (in generating additional training data for a new camera view for example) of the model may or may not be sufficient motivation for every researcher to adopt this approach, the true power of the synthetic approach will be realized if and when adopted for team science projects[21] with standardized rigs (such as the IBL example; https://osf.io/h3ec5, IBL-scene_non-rendered_demo.mp4). IBL behavioral video data from https://docs.google.com/document/d/e/2PACX-1vS2777bCbDmMre-wyeDr4t0jC-0YsV_uLtYkfS3h9zTwgC7qeMk-GUqxPqcY7ylH17I1Vo1nIuuj26L/pub. If adopted at the project outset the savings are magnified across all participating labs.

Sampling and statistics.

The sample size for the number of paired real and synthetic images was 1000 images per mouse, in total 10000 images. We used groups of 4 and 5 mice for the comparisons as this is a typical minimal number used in a behavioral group. The 5 mice were used for individual U-GAT-IT model training and an aggregate DLC 2D model as well as comparisons on SSIM and KID. Four mice were used to train U-GAT-IT models on two cameras and were used to train a DLC 3D project, a sample analysis was done on a single video as shown. The sample size for the comparison of 2D poses and 3D poses was 18,063 paired poses. Data was analyzed using GraphPad Prism 6 and custom software in MATLAB (2020) and Python (3.7). The Mann-Whitney non-parametric test was used to compare real and synthetic images generated by the U-GAT-IT model. Dunn's Multiple Comparison test was used to compare the performance between synthetically trained and manually trained DLC 3D

projects. *** denote p<0.001. No data points were excluded from the analysis. Each comparison presented in the paper was repeated in multiple animals as stated above. Real and synthetic images were assigned randomly for the SSIM comparison. Mice belonged to a single group and were not split into categories. Computational analysis was not performed blinded as there was only a single group of animals for each modeled scene.

For quantitative evaluation for the performance of the U-GAT-IT model we employ the structural similarity index measure (SSIM) [22]. This is a well-accepted method of evaluating changes in image quality after compression or other manipulations. Its values range between 0 and 1 with 1 being an identical image and 0 being no similarity. Histograms of SSIM scores were created by comparing 100 frames from each pair of mouse videos per condition and calculating their SSIM, resulting in 10,000 scores for each mouse pairing. Same conditions were combined to generate single histograms of 50,000 measurements during paired conditions and 200,000 in unpaired (bin counts: 100 & 400 respectively, Fig. 2c)

For further quantitative evaluation for the performance of the U-GAT-IT model, we use the recently proposed Kernel Inception Distance (KID) [13], which computes the squared Maximum Mean Discrepancy between the feature representations of real and generated images (https://github.com/abdulfatir/gan-metrics-pytorch). The feature representations are extracted from the Inception network [23]. The lower the KID, the more shared visual similarities between real and generated images. Therefore, if well translated, the KID will have a small value in several datasets.

Hopkins score [18] is a statistical test that examines the distribution of data working from the premise that normally distributed data would have less tendency to cluster. It can be used as an independent way to compare and quantify a clustering algorithm on two or more different datasets. Values range from 0 which is highly cluster-able to 1 which indicates random but uniformly distributed data.

Data availability

All data and raw behavioral video images are available online at Open Science Framework https://osf.io/h3ec5/ [2]. Source data for Fig. 2 and Supplementary Figures 1–3 are provided with this paper.

Code availability

All code and models are available on public repositories: https://github.com/ubcbraincircuits/mCBF [14] and https://osf.io/h3ec5/ [2] see workflow document for an overview of software steps. Code (KID score, U-GAT-IT, pose-lifting and clustering) is set up for reproducible runs via a capsule on Code Ocean[24].

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgements:

This work was supported by a Canadian Institutes of Health Research (CIHR) FDN-143209, NIH R21, a Fondation Leducq grant, Brain Canada for the Canadian Neurophotonics Platform, and a Canadian Partnership for Stroke Recovery Catalyst grant to THM. HR is supported by grants from Natural Sciences and Engineering Research Council of Canada (NSERC). D.X. was supported in part by funding provided by Brain Canada, in partnership with Health Canada, for the Canadian Open Neuroscience Platform initiative. This work was supported by computational resources made available through the NeuroImaging and NeuroComputation Centre at the Djavad Mowafaghian Centre for Brain Health (RRID: SCR_019086). NLF is supported by an NSERC Discovery Grant. Micro-CT imaging was performed at the UBC Centre for High-Throughput Phenogenomics, a facility supported by the Canada Foundation for Innovation, British Columbia Knowledge Development Foundation, and the UBC Faculty of Dentistry. We thank Ellen Koch and Lynn Raymond of the University of British Columbia Djavad Mowafaghian Centre for Brain Health for the mouse open field video.

References

1. Mathis A et al. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. Nat. Neurosci. 21, 1281–1289 (2018). [PubMed: 30127430]
2. Bolanos L et al. Synthetic Animated Mouse (SAM), University of British Columbia, Data and 3D-models. (2021) doi:10.17605/OSF.IO/H3EC5.
3. Khmelinskii A et al. Articulated whole-body atlases for small animal image analysis: construction and applications. Mol. Imaging Biol. 13, 898–910 (2011). [PubMed: 20824510]
4. Klyuzhin IS, Stortz G & Sossi V Development of a digital unrestrained mouse phantom with non-periodic deformable motion. 2015 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC) (2015) doi:10.1109/nssmic.2015.7582140.
5. Li S et al. Deformation-Aware Unpaired Image Translation for Pose Estimation on Laboratory Animals. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020) doi:10.1109/cvpr42600.2020.01317.
6. Synthetic to Real World Image Translation Using Generative Adversarial Networks - IEEE Conference Publication. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8493745.
7. The International Brain Laboratory et al. Standardized and reproducible measurement of decision-making in mice. Cold Spring Harbor Laboratory 2020.01.17.909838 (2020) doi:10.1101/2020.01.17.909838.
8. Galiñanes GL, Bonardi C & Huber D Directional Reaching for Water as a Cortex-Dependent Behavioral Framework for Mice. Cell Rep. 22, 2767–2783 (2018). [PubMed: 29514103]
9. Kwak IS, Guo J-Z, Hantman A, Kriegman D & Branson K Detecting the starting frame of actions in video. arXiv [cs.CV] (2019).
10. Kim J, Kim M, Kang H & Lee K U-GAT-IT: Unsupervised Generative Attentional Networks with Adaptive Layer-Instance Normalization for Image-to-Image Translation. arXiv [cs.CV] (2019).
11. Kim Junho, Kim Minjae, Kang Hyeonwoo, Lee Kwanghee. U-GAT-IT: Unsupervised Generative Attentional Networks with Adaptive Layer-Instance Normalization for Image-to-Image Translation.
12. Wang Z, Simoncelli EP & Bovik AC Multiscale structural similarity for image quality assessment. in The Thirty-Seventh Asilomar Conference on Signals, Systems Computers, 2003 vol. 2 1398–1402 Vol.2 (2003).
13. Bińkowski M, Sutherland DJ, Arbel M & Gretton A Demystifying MMD GANs. arXiv [stat.ML] (2018).
14. Bolanos L et al. github.com/ubcbraincircuits/mCBF. (2021).
15. Nath T et al. Using DeepLabCut for 3D markerless pose estimation across species and behaviors. Nat. Protoc. 14, 2152–2176 (2019). [PubMed: 31227823]
16. Martinez J, Hossain R, Romero J & Little JJ A Simple Yet Effective Baseline for 3d Human Pose Estimation. 2017 IEEE International Conference on Computer Vision (ICCV) (2017) doi:10.1109/iccv.2017.288.
17. Günel S et al. DeepFly3D, a deep learning-based approach for 3D limb and appendage tracking in tethered, adult Drosophila. Elife 8, (2019).

18. Hopkins B & Skellam JG A New Method for determining the Type of Distribution of Plant Individuals. Annals of Botany vol. 18 213–227 (1954).

References Methods

19. Levine JH et al. Data-Driven Phenotypic Dissection of AML Reveals Progenitor-like Cells that Correlate with Prognosis. Cell 162, 184–197 (2015). [PubMed: 26095251]
20. Blondel VD, Guillaume J-L, Lambiotte R & Lefebvre E Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment vol. 2008 P10008 (2008).
21. Vogt N Collaborative neuroscience. Nat. Methods 17, 22 (2020).
22. Wang Z, Bovik AC, Sheikh HR & Simoncelli EP Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. 13, 600–612 (2004). [PubMed: 15376593]
23. Szegedy C, Vanhoucke V, Ioffe S, Shlens J & Wojna Z Rethinking the Inception Architecture for Computer Vision. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) doi:10.1109/cvpr.2016.308.
24. Bolaños LA et al. Code Ocean Computer Code Run Capsule, Synthetic animated mouse University of British Columbia. A 3D virtual mouse generates synthetic training data for behavioral analysis (2021) doi:10.24433/CO.5412865.v1.

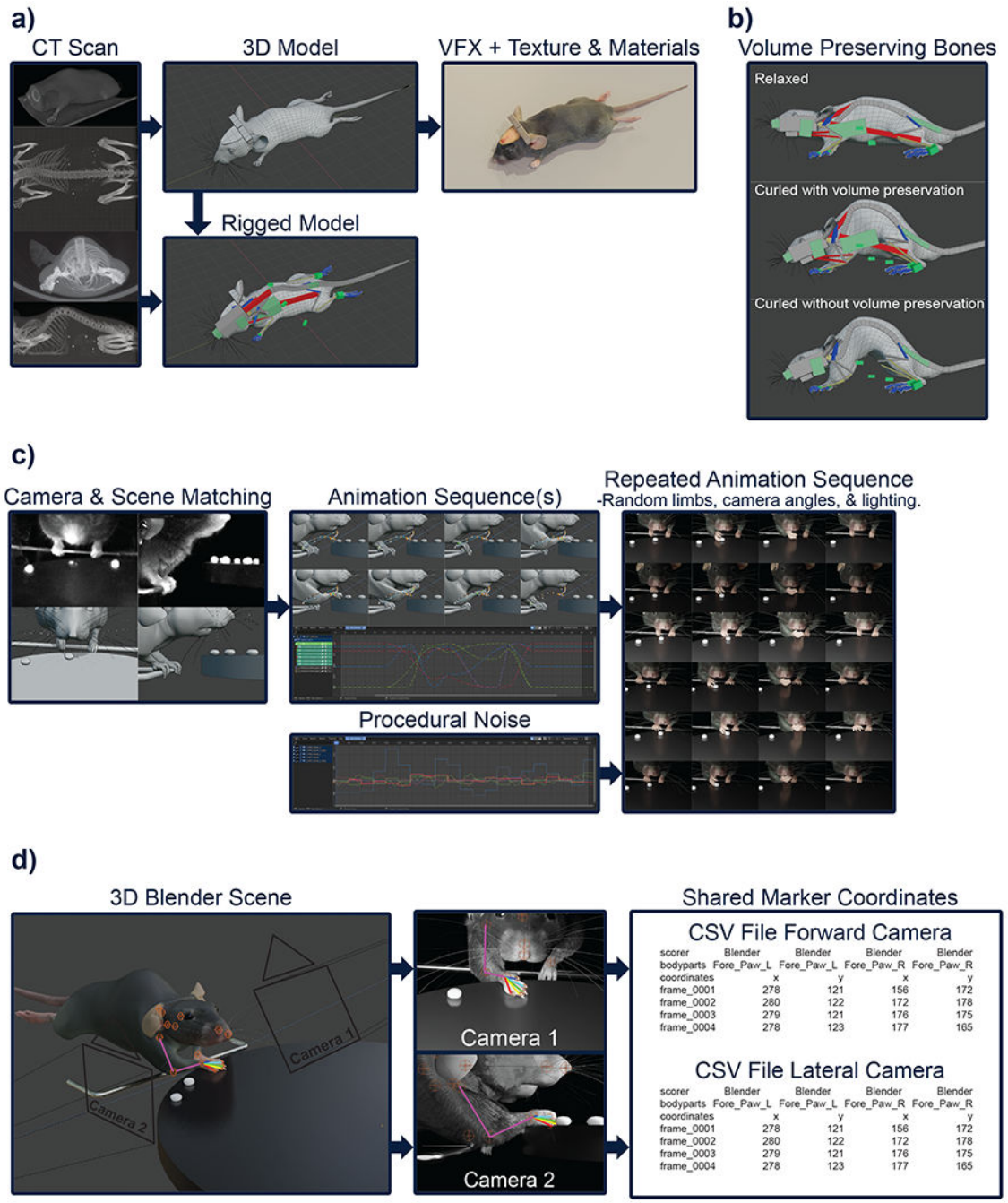

Fig. 1. A realistic 3D mouse model suitable for generating labels for machine learning.

a, Creation of the 3D model, starting with high resolution CT scans and their respective high-density triangle meshes to model a single mesh of lower-density quad faces. Rigging of the model using the skeleton shown in the CT scans as guides, and the addition of textures, materials, and fur. **b**, The effect of the volume preserving bones used to enhance large body regions during deformation. **c**, The process of creating the initial renders by modelling the scene of interest, creating animations based on real videos, and the addition of noise. **d**,

Demonstration in a multi-camera synthetic setup with shared labels suitable for machine learning training.

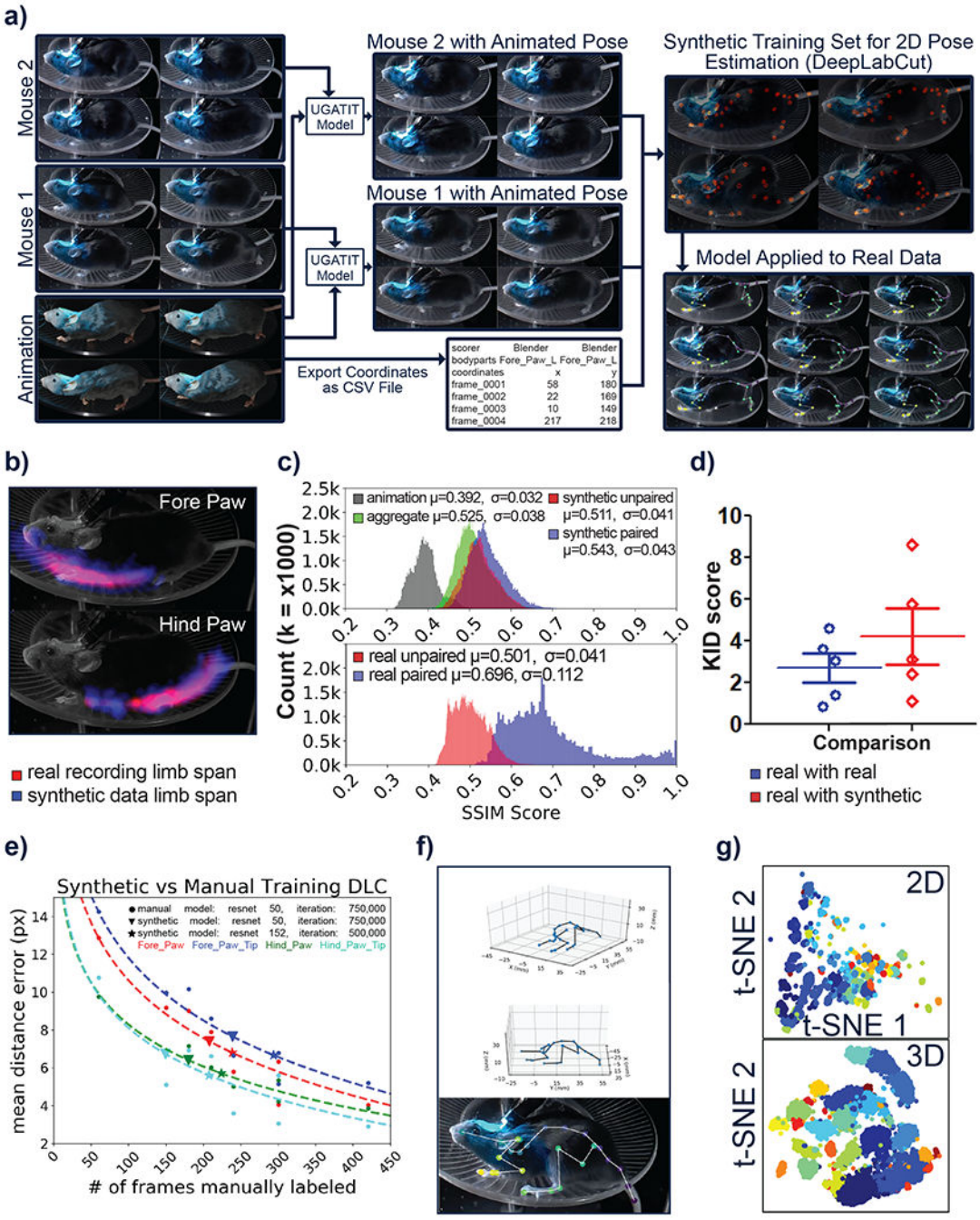

Fig. 2. Image domain translation makes realistic videos of animated mouse models.

a, Implementation of image domain translation into our workflow to generate realistic synthetic videos. **b**, Kernel density estimation plots of both fore paw and hind paw positions in synthetic data generated with limb and camera noise (n=1,000), and real recordings (n=26,477). **c**. Histograms of SSIM scores for the indicated comparisons (n = 50,000 for paired conditions, n = 200,000 for unpaired). **d**, Kernel Inception Distances (KID) for further analysis of synthetic images. KID scores show no significant difference between real and synthetic images of paired mice (scatter dot plot, line at mean with SEM, real with real,

mean ± SEM=2.701± 0.6941; real vs synthetic, mean ± SEM= 4.207± 1.339, n=5 mice, 1000 paired real and synthetic images for each mouse, real and synthetic images were assigned randomly to the comparison). Mann Whitney test, two-tailed test results in a p value of 0.5476. **e**, Comparison of 2D pose estimation between a synthetically trained model and multiple manually labeled models. **f**, Single-camera 2D-to-3D pose lifting using a linear neural network. **g**, Comparison of 2D and 3D poses generated by the pose lifter. Unsupervised clustering algorithm (Phenograph) was used to classify the poses (n = 18,063 paired poses). Different colors in the t-SNE plot indicate different pose clusters. Hopkins test shows 3D poses (Hopkins score: 0.02) are more clusterable than 2D poses (Hopkins score: 0.04).