Automatic tracking of mouse social posture dynamics by 3D videography, deep learning and GPU-accelerated robust optimization

Christian L. Ebbesen[1,2,*] & Robert C. Froemke[1,2,3,*]

[1] Skirball Institute of Biomolecular Medicine, Neuroscience Institute, Departments of Otolaryngology, Neuroscience and Physiology, New York University School of Medicine, New York, NY, 10016, USA.

[2] Center for Neural Science, New York University, New York, NY, 10003, USA.

[3] Howard Hughes Medical Institute Faculty Scholar.

* Correspondence to: C.L.E. (christian.ebbesen@nyumc.org) or R.C.F. (robert.froemke@med.nyu.edu)

ABSTRACT

Social interactions powerfully impact both the brain and the body, but high-resolution descriptions of these important physical interactions are lacking. Currently, most studies of social behavior rely on labor-intensive methods such as manual annotation of individual video frames. These methods are susceptible to experimenter bias and have limited throughput. To understand the neural circuits underlying social behavior, scalable and objective tracking methods are needed. We present a hardware/software system that combines 3D videography, deep learning, physical modeling and GPU-accelerated robust optimization. Our system is capable of fully automatic multi-animal tracking during naturalistic social interactions and allows for simultaneous electro-physiological recordings. We capture the posture dynamics of multiple unmarked mice with high spatial (~2 mm) and temporal precision (60 frames/s). This method is based on inexpensive consumer cameras and is implemented in python, making our method cheap and straightforward to adopt and customize for studies of neurobiology and animal behavior.

INTRODUCTION

Objective quantification of natural social interactions is difficult. The majority of our knowledge about animal social behavior comes from hand-annotation of videos, yielding ethograms of discrete social behaviors such as 'social following', 'mounting', or 'anogenital sniffing'[1]. It is widely appreciated that these methods are susceptible to experimenter bias and have limited throughput. There is an additional problem with these approaches, in that manual annotation of video frames yields no detailed information about movement kinematics and physical body postures. This shortcoming is especially critical for studies relating neural activity patterns or other physiological signals to social behavior. For example, neural activity in many areas of the cerebral cortex are strongly modulated by movement and posture[2,3], and activity profiles in somatosensory regions can be difficult to analyze without understanding the physics and high-resolution dynamics of touch. Important aspects of social behavior, from gestures to light touch and momentary glances can be transient and challenging to observe in most settings, but critical to capturing the details and changes to social relationships and networks[4,5]. Together the potential for false positives and false negatives can be high, and to date these issues have thwarted our understanding of the neural basis of somatic physiology and social behavior.

The use of deep convolutional networks to recognize objects in images has revolutionized computer vision, and consequently, also led to major advances in behavioral analysis. Drawing upon these methodological advances, several recent publications have developed algorithms for tracking, such as 'DeepLabCut'[6], '(S)LEAP'[7] and 'DeepPoseKit'[8]. These methods function by detection of key-points in 2D videos, and estimation of 3D postures is not straightforward in interacting animals[9]. Spatiotemporal regularization is needed to ensure that tracking is stable and error-free, even when multiple animals are closely interacting. During mounting or allo-grooming, for example, interacting animals block each other from the camera view and tracking algorithms can fail. Having a large number of cameras film the animals from all sides can solve these

problems[9,10], but this has required extensive financial resources for equipment, laboratory space and processing power, which renders widespread use infeasible.

In parallel, other studies have used depth-cameras for animal tracking, fitting a physical body-model of the animal to 3D data[11,12]. These methods are powerful because they explicitly model the 3D movement and poses of multiple animals. However, due to technical limitations of depth imaging hardware (frame rate, resolution, motion blur), it is to date only possible to extract partial posture information about small and fast-moving animals, such as lab mice. Consequently, when applied to mice, these methods are prone to tracking mistakes when interacting animals get close to each other and the tracking algorithms require continuous manual supervision to detect and correct errors. This severely restricts throughput, making tracking across long time scales infeasible.

Here we describe a novel system for multi-animal tracking that combines ideal features from both approaches. Our method fuses physical modeling of depth data and deep learning-based analysis of synchronized color video to estimate the body postures, enabling us to reliably track multiple mice during naturalistic social interactions. Our method is fully automatic (i.e., quantitative, scalable, and free of experimenter bias), is based on inexpensive consumer cameras, and is implemented in Python, a simple and widely used computing language. Together, this makes our method inexpensive to adopt and easy to use and customize, paving a way for more widespread study of naturalistic social behavior in neuroscience and experimental biomedicine.

RESULTS

Raw data acquisition

We established an experimental setup that allowed us to capture synchronized color images and depth images from multiple angles, while simultaneously recording synchronized neural data (**Fig. 1a**). We used inexpensive, state-of-the-art 'depth cameras' for computer vision and robotics. These cameras contain several imaging modules: one color sensor, two infrared sensors and an infrared laser projector (**Fig. 1b**). Imaging data pipelines, as well as intrinsic and extrinsic sensor calibration parameters can be accessed over USB through a C/C++ SDK with Python bindings. We placed four depth cameras, as well as four synchronization LEDs around a transparent acrylic cylinder which served as our behavioral arena (**Fig. 1c**).

Each depth camera projects a static dot pattern across the imaged scene, adding texture in the infrared spectrum to reflective surfaces (**Fig. 1d**). By imaging this highly-textured surface simultaneously with two infrared sensors per depth camera, it is possible to estimate the distance of each pixel in the infrared image to the depth camera by stereopsis (by locally estimating the binocular disparity between the textured images). Since the dot pattern is static and only serves to add texture, multiple cameras do not interfere with each other and it is possible to image the same scene from multiple angles. This is one key aspect of our method, not possible with depth imaging systems that rely on actively modulated light (such as the Microsoft Kinect system and earlier versions of the Intel Realsense cameras).

Since mouse movement is fast[13], it is vital to minimize motion blur in the infrared images and thus the final 3D data ('point-cloud'). To this end, our method relies on two key features. First, we use depth cameras where the infrared sensors have a global shutter (e.g., Intel D435) rather than a rolling shutter (e.g., Intel D415). Using a global shutter reduces motion blur in individual image frames, but also enables synchronized image capture across cameras. Without synchronization between cameras, depth images are taken at different times,

which adds blur to the composite point-cloud. We set custom firmware configurations in our recording program, such that all infrared sensors on all four cameras are hardware-synchronized to each other by TTL-pulses via custom-built, buffered synchronization cables (**Fig. 1b**).

We wrote a custom multithreaded Python program with online compression, that allowed us to capture the following types of raw data from all four cameras simultaneously: 8-bit RGB images (320 x 210 pixels, 60 frames/s), 16-bit depth images (320 x 240 pixels, 60 frames/s) and the 8-bit intensity trace of a blinking LED (60 samples/s, automatically extracted in real-time from the infrared images). Our program also captures camera meta-data, such as hardware time-stamps and frame numbers of each image, which allows us to identify and correct for possible dropped frames. On a standard desktop PC, the recording system had very few dropped frames and the video recording frame rate and the imaging and USB image transfer pipeline was stable (**Fig. 1e,f**).

Temporal stability and temporal alignment

In order to relate tracked behavioral data to neural recordings, we need precise temporal synchronization. Digital hardware clocks are generally stable but their internal speed can vary, introducing drift between clocks. Thus, even though all depth cameras provide hardware timestamps for each acquired image, for long-term recordings, across behavioral time scales (hours to days), a secondary synchronization method is required.

For synchronization to neural data, our recording program uses a USB-controlled Arduino microprocessor to output a train of randomly-spaced voltage pulses during recording. These voltage pulses serve as TTL triggers for our neural acquisition system (sampled at 30 kHz) and drive LEDs, which are filmed by the depth cameras (**Fig. 1a**). The cameras sample an automatically detected ROI to sample the LED state at 60 frames/s, integrating across a full infrared frame exposure (**Fig. 1g**). We use a combination of cross-correlation and robust

regression to automatically estimate and correct for shift and drift between the depth camera hardware clocks and the neural data. Since we use random pulse trains for synchronization, alignment is unambiguous and we can achieve super-frame-rate-precision. In a typical experiment, we estimated that the depth camera time stamps drifted with ~49 µs/min. We corrected for this drift to yield stable residuals between TTL flips and depth frame exposures (**Fig. 1h**). Note that the neural acquisition system is not required for synchronization and for a purely behavioral study, we can run the same LED-based protocol to correct for potential shift and drift between cameras by choosing one camera as a reference.

Detection of body key-points by deep learning

We preprocessed the raw image data to extract two types of information for the tracking algorithm: the location in 3D in space of body key-points and the 3D point-cloud corresponding to the body surface of the animals. We used a deep convolutional neural network to detect key-points in the RGB images, and extracted the 3D point-cloud from the depth images (**Fig. 2a**). For key-point detection (nose, ears, base of tail, and neural implant for implanted animals), we used a 'stacked hourglass network'[14]. This type of network architecture combines residuals across successive upsampling and downsampling steps to generate its output, and has been successfully applied to human pose estimation[14] and limb tracking in immobilized flies[15] (**Fig. 2b, Supplementary Fig. 1**).

We used back-propagation to train the network to output four 'target maps', each indicating the pseudo-posterior probability of each type of key-point, given the input image. The target maps were generated by manually labeling the key-points in training frames, followed by down-sampling and convolution with Gaussian kernels (**Fig. 2c**, 'target maps'). We selected the training frames using image clustering to avoid redundant training on very similar frames[8]. The manual key-point labeling can be done with any labeling software. We

138 customized a version of the lightweight, open source labeling GUI from the 'DeepPoseKit' package[8] for the
139 four types of key-points, which we provide as supplementary software (**Supplementary Fig. 2**).
140
141 In order to improve key-point detection, we used two additional strategies. First, we also trained the network
142 to predict 'affinity fields'[16]. We used '1D' affinity fields[8], generated by convolving the path between labeled
143 body key-points that are anatomically connected in the animal. With our four key-points, we added seven
144 affinity fields ('nose-to-ears', 'nose-to-tail', etc.), that together form a skeletal representation of each body
145 (**Fig. 2c,** 'affinity fields'). Thus, from three input channels (RGB pixels), the network predicts eleven output
146 channels (**Fig. 2d**). As the stacked hourglass architecture involves intermediate prediction, which feeds back
147 into subsequent hourglass blocks (repeated encoding and decoding, **Fig 2b**), prediction of affinity fields feeds
148 into downstream predictions of body key-points. This leads to improvement of downstream key-point predic-
149 tions, because the affinity fields give the network access to holistic information about the body. The intuitive
150 probabilistic interpretation is that instead of simply asking questions about the keypoints (e.g., 'do these pixels
151 look like an ear?'), we can increase predictive accuracy by considering the body context (e.g., 'these pixels
152 sort of look like an ear, and those pixels sort of look like a nose – but does this path between the pixels also
153 look like the path from an ear to a nose?').
154
155 The second optimization approach was image data augmentation during training[17]. Instead of only training
156 the network on manually-labeled images, we also trained the network on morphed and distorted versions of
157 the labeled images (**Supplementary Fig. 3**). Training the network on morphed images (e.g., rotated or en-
158 larged), gives a similar effect to training on a much larger dataset of labeled images, because the network then
159 learns to predict many artificially generated, slightly different views of the animals. Training the network on
160 distorted images is thought to reduce overfitting on single pixels and reduce the effect of motion blur[17].
161

Using a training set of 526 images, and by automatically adjusting learning rate during training, the network was well-trained (plateaued) within one hour of training on a standard desktop computer (**Fig. 2e**), yielding good predictions of both body key-points and affinity fields (**Fig. 2f**).

Pre-processing of 3D video

By aligning the color images to the depth images, and aligning the depth images in 3D space, we could assign three dimensional coordinates to the detected key-points. We pre-processed the depth data to accomplish two goals. First, we wanted to align the cameras to each other in space, so we could fuse their individual depth images to one single 3D point-cloud. Second, we wanted to extract only points corresponding to the animals' body surfaces from this composite point-cloud.

To align the cameras in space, we filmed the trajectory of a sphere that we moved around the behavioral arena. We then used a combination of motion filtering, color filtering, smoothing and thresholding to detect the location of the sphere in the color frame, extracted the partial 3D surface from the aligned depth data, and used a robust regression method to estimate the center coordinate (**Fig. 3a**). This procedure yielded a 3D trajectory in the reference frame of each camera (**Fig. 3b**) that we could use to robustly estimate the transformation matrices needed to bring all trajectories into the same frame of reference (**Fig. 3c**). This robust alignment is a key aspect of our method, as errors can easily be introduced by moving the sphere too close to a depth camera or out of the field of view during recording (**Fig. 3b,c**, arrow). After alignment, the median camera-to-camera difference in the estimate of the center coordinate of the 40-mm-diameter sphere was only 2.6 mm across the entire behavioral arena (**Fig. 3d,e**).

We used a similar robust regression method to automatically detect the base of the behavioral arena. We detected planes in composite point-cloud (**Fig. 3f**) and used the location and normal vector, estimated across

60 random frames (**Fig. 3g**), to transform the point-cloud such that the base of the behavioral arena laid in the *xy*-plane (**Fig. 3h**). To remove imaging artifacts stemming from light reflection and refraction due to the curved acrylic walls, we automatically detected the location and radius of the acrylic cylinder (**Fig. 3i**). With the location of both the arena base and the acrylic walls, we used simple logic filtering to remove all points associated with the base and walls, leaving only points inside the behavioral arena (**Fig. 3j**). Note that if there is no constraint on laboratory space, an elevated platform can be used as a behavioral arena, eliminating imaging artifacts associated with the acrylic cylinder.

Loss function design

The pre-processing pipeline described above takes color and depth images as inputs, and outputs two types of data: a point-cloud, corresponding to the surface of the two animals, and the 3D coordinates of detected body key-points (**Fig. 4a, Supplementary Video 1**). To track the body postures of interacting animals across space and time, we developed an algorithm that incorporates information from both data types. The basic idea of the tracking algorithm is that for every frame, we fit the mouse bodies by minimizing a loss function of both the point-cloud and key-points, subject to a set of spatiotemporal regularizations.

For the loss function, we made a simple parametric model of the skeleton and body surface of a mouse. The body model consists of two prolate spheroids (the 'hip ellipsoid' and 'head ellipsoid'), with dimensions based on an average adult mouse (**Fig. 4b**). The head ellipsoid is rigid, but the hip ellipsoid has a free parameter (s) modifying the major and minor axes to allow the hip ellipsoids to be longer and narrower (e.g., during stretching, running, or rearing) or shorter and wider (e.g., when still or self-grooming). The two ellipsoids are connected by a joint that allows the head ellipsoid to turn left/right and up/down within a cone corresponding to the physical movement limits of the neck.

Keeping the number of degrees of freedom low is vital to make loss function minimization computationally feasible[18]. Due to the rotational symmetry of the ellipsoids, we could choose a parametrization with 8 degrees of freedom per mouse body: the central coordinate of the hip ellipsoid (*x, y, z*), the rotation of the major axis of the hip ellipsoid around the *y*- and *z*-axis (β, γ), the left/right and up/down rotation of the head ellipsoid (θ, φ), and the stretch of the hip ellipsoids (*s*). For the implanted animal, we added an additional sphere to the body model, approximating the surface of the head-mounted neural implant (**Fig. 4b**). The sphere is rigidly attached to the head ellipsoid and has one degree of freedom; a rotational angle (ψ) that allows the sphere to rotate around the head ellipsoid, capturing head tilt of the implanted animal. Thus, in total, the joint pose (the body poses of both mice) was parametrized by only 17 variables.

To fit the body model, we adjusted these parameters to minimize a weighted sum of two loss terms: *(i)* The shortest distance from every point in the point-cloud to body model surface. *(ii)* The distance from detected key-points to their corresponding location on the body model surface (e.g., nose key-points near the tip of one of the head ellipsoids, tail key-points near the posterior end of a hip ellipsoid).

We then used several different approaches for optimizing the tracking. First, for each of the thousands of point in the point-cloud, we needed to calculate the shortest distance to the body model ellipsoids. Calculating these distances exactly is not computationally feasible, as this requires solving a six-degree polynomial for every point[19]. As an approximation, we instead used the shortest distance to the surface, along a path that passes through the centroid (**Supplementary Fig. 4a,b**). Calculating this distance could be implemented as pure tensor algebra[20], which could be executed efficiently on a GPU in parallel for all points simultaneously. Second, to reduce the effect of imaging artifacts in the color and depth imaging (which can affect both the point-cloud or the 3D coordinates of the key-points), we clipped distance losses at 3 cm, such that distant 'outliers'

do contribute and not skew the fit (**Supplementary Fig. 4c**). Third, because pixel density in the depth images depends on the distance from the depth camera, we weighed the contribution of each point in the point-cloud by the squared distance to the depth camera (**Supplementary Fig. 4d**). Fourth, to ensure that the minimization does not converge to unphysical joint postures (e.g., where the mouse bodies are overlapping), we added a penalty term to the loss function if the body models overlap. Calculating overlap between two ellipsoids is computationally expensive[21], so we computed overlaps between implant sphere and spheres centered on the body ellipsoids with a radius equal to the minor axis (**Supplementary Fig. 4f**). Fifth, to ensure spatiotemporal continuity of body model estimates, we also added a penalty term to the loss function, penalizing overlap between the mouse body in the current frame, and other mouse bodies in the previous frame. This ensures that the bodies do not switch place, something that could otherwise happen if the mice are in joint poses with certain mirror symmetries (**Supplementary Fig. 4g,h**).

GPU-accelerated robust optimization

Minimizing the loss function requires solving three major challenges. The first challenge is computational speed. The number of key-points and body parts is relatively low (~tens), but the number of points in the point-cloud is large (~thousands), which makes the loss function computationally expensive. For minimization, we need to evaluate the loss function multiple times per frame (at 60 frames/s). If loss function evaluation is not fast, tracking becomes unusably slow. The second challenge is that the minimizer has to properly explore the loss landscape within each frame and avoid local minima. In early stages of developing this algorithm, we were only tracking interacting mice with no head implant (**Supplementary Video 2**). In that case, for the small frame-to-frame changes in body posture, the loss function landscape was nonlinear, but approximately convex, so we could use a fast, derivative-based minimizer to track changes in body posture (geodesic Levenberg-Marquardt steps[18]). For use in neuroscience experiments, however, one or more mice might carry a neural implant for recording or stimulation. The implant is generally at a right angle and offset from the

'hinge' between the two hip and head ellipsoids, which makes the loss function highly non-convex[22]. The final challenge is robustness against local minima in state space. Even though a body posture minimizes the loss in a single frame, it might not be an optimal fit, given the context of other frames (e.g., spatiotemporal continuity, no unphysical movement of the bodies).

To solve these three challenges – speed, state space exploration, and spatiotemporal robustness – we designed a custom GPU-accelerated minimization algorithm, which incorporates ideas from annealed particle filters[23] and online Bayesian filtering[24]. To maximize computational speed, the algorithm was implemented as pure tensor algebra in Pytorch, a high-performance GPU computing library[25]. Annealed particle filters are suited to explore highly non-convex loss surfaces[23], which allowed us to avoid local minima within each frame. Between frames, we used online Bayesian filtering, to avoid being trapped in low-probability solutions given the preceding tracking. For every frame, we first proposed the state of the 17-parameters using kernel-recursive least-squares tracking ('KRLS-T'[24]) from a Bayesian filter bank based on preceding frames. After particle filter-based loss function minimization within a single frame, we updated the Bayesian filter bank, and proposed a particle filter starting point for the next frame. This strategy has three major advantages. First, by proposing a solution, taking into account previous variables and their covariances, we often already started loss function minimization close to the new minimum. Second, if the Bayesian filter deems that the fit for a single frame is unlikely, based on the preceding frames, this fit will only weakly update the Bayesian filter bank, and thus only weakly perturb the upcoming tracking. This gave us a convenient way to balance the information provided by the fit of a single frame, and the 'context' provided by previous frames. Third, the Bayesian filter-based approach is only dependent on previously tracked frames, not future frames. This is in contrast to other approaches to incorporating context that rely on versions of backwards belief propagation[5,15,26]. Since our algorithm only uses past data, it is in principle possible to optimize our algorithm for real-time use in closed-loop experiments.

For each frame, we explored the loss surface with 200 particles (**Fig. 4b,c**). We generated the particles by perturbing the proposed minimum, based on the previous frames, by quasi-random, low-discrepancy sampling[27] (**Supplementary Fig. 5**). We exploited the fact that the loss function structure allowed us to execute several key steps in parallel, across multiple independent dimensions, and implemented these calculations as vectorizes tensor operations. This allowed us to leverage the power of CUDA kernels for fast tensor algebra on the GPU[25]. Specifically, to efficiently calculate the point-cloud loss (shortest distance from a point in the point-cloud to the surface of a body model), we calculated the distance to all five body model spheroids for all points in the point-cloud and for all 200 particles, in parallel (**Fig. 4c**). We then applied fast minimization kernels across the two body models, to generate a smallest distance to either mouse, for all points in the pointcloud. Because the mouse body models are independent, we only had to apply a minimization kernel to calculate the smallest distance, for every point, to 40,000 (200 x 200) joint poses if the two mice. These parallel computation steps are a key aspect of our method, which allows our tracking algorithm to avoid the 'curse of dimensionality', by not exploring a 17-dimensional space, but rather explore the intersection of two independent 8-dim and 9-dim subspaces in parallel.

Tracking algorithm performance

To ensure that the tracking algorithm did not get stuck in suboptimal solutions, we forced the particle filter to explore a large search space within every frame (**Fig. 5a-c**). In successive iterations, we gradually made perturbations to the particles smaller and smaller by annealing the filter[23]), to approach the minimum. At the end of each iteration, we 'resampled' the particles by picking the 200 joint poses with the lowest losses in the 200-by-200 matrix of losses. This resampling strategy has two advantages. First, it can be done without fully sorting the matrix[28], the most computationally expensive step in resampling[29]. Second, it provides a kind of quasi–'importance sampling'. During resampling, some poses in the next iteration might be duplicates (picked

from the same row or column in the 200-by-200 loss matrix.), allowing particles in each subspace to collapse at different rates (if the particle filter is very certain about one body pose, but not the other, for example).

By investigating the performance of the particle filter across iterations, we found that the filter generally converged within five iterations (**Fig. 5d**), providing good tracking across frames (**Fig. 5e**). In every frame, the particle filter fit yields a noisy estimate of the 3D location of the mouse bodies. The transformation from the joint pose parameters (e.g., rotation angles, spine scaling) to 3D space is highly nonlinear, so simple smoothing of the trajectory in pose parameter space would distort the trajectory in real space. Thus, we filtered the tracked trajectories by a combination of Kalman-filtering and maximum likelihood-based smoothing[30,31] and 3D rotation smoothing in quaternion space[32] (**Supplementary Fig. 6c-e, Supplementary Video 3**).

Representing the joint postures of the two animals with this parametrization was highly data efficient, reducing the memory footprint from ~3.7 GB/min for raw color/depth image data, to ~0.11 GB/min for pre-processed point-cloud/key-point data to ~1 MB/min for tracked body model parameters. On a regular desktop computer with a single GPU, we could do key-point detection in color image data from all four cameras in ~2x real time (i.e. it took 30 mins to process a 1 hr experimental session). Depth data processing (point-cloud merging and key-point deprojection) ran at ~0.7x real time, and the tracking algorithm ran at ~0.2x real time (if the filter uses 200 particles and 5 filter iterations per frame). Thus, for a typical experimental session (~ hours), we would run the tracking algorithm overnight, which is possible because the code is fully unsupervised.

Note that this version of the algorithm is written for active development, not pure speed. For example, a large part of the processing time is spent reading/writing data to disk, and – while convenient for modifying and experimenting with the code – it is not necessary to first process color, then depth and then run a tracking

329 algorithm step, for example. In its present form, the code is fast enough to be useful, but not optimized to the
330 theoretical maximum speed.
331
332 ***Error detection***
333 Error detection and correction is a critical component of behavioral tracking. Even if error rates are nominally
334 low, errors are non-random, and errors often happen exactly during the behaviors in which we are most inter-
335 ested: interactions. In multi-animal tracking, two types of tracking error are particularly fatal as they com-
336 pound over time: identity errors and body orientation errors (**Supplementary Fig. 7a**). In conventional track-
337 ing approaches using only 2D videos, it is often difficult to correctly track identities when interacting mice
338 are closely interacting, allo-grooming, or passing over and under each other. Although swapped identities can
339 be corrected later once the mice are well-separated again, this still leaves individual behavior during the actual
340 social interaction unresolved[5,26]. We found that our tracking algorithm was robust against both identity swaps
341 (**Supplementary Fig. 7b-e**) and body direction swaps (**Supplementary Fig. 8**). This observation agrees with
342 the fact that tracking in three-dimensional space (subject to our implemented spatiotemporal regularizations)
343 *a priori* ought to allow better identity tracking; In full 3D space it is easier to determine who is rearing over
344 whom during an interaction, for example.
345
346 To test our algorithm for more subtle errors, we manually inspected 500 frames, randomly selected across an
347 example 21 minute recording session. In these 500 frames, we detected one tracking mistake, corresponding
348 to 99.8% correct tracking (**Supplementary Fig. 9a**). The identified tracking mistake was visible as a large,
349 transient increase in the point-cloud loss function (**Supplementary Fig. 9b**). After the tracking mistake, the
350 robust particle filter quickly recovered to correct tracking again (**Supplementary Fig. 9c**). By detecting such
351 loss function anomalies, or by detecting 'unphysical' postures or movements in the body models, potential
352 tracking mistakes can be automatically 'flagged' for inspection (**Supplementary Fig. 9c,d**). After inspection,

errors can be manually corrected or automatically corrected in many cases, for example by tracking the particle filter backwards in time after it has recovered. As the algorithm recovers after a tracking mistake, it is generally unnecessary to actively supervise the algorithm during tracking, and manual inspection for potential errors can be performed after running the algorithm overnight.

Automated analysis of movement kinematics and social behavior

Despite the high level of data compression (from raw images to pre-processed data to only 17 dimensions), a human observer can clearly distinguish social events in the tracked data (**Fig. 5f**). The major motivation behind developing our method, however, was to eschew manual labeling especially for large-scale datasets on the order of days to months of video tracking of the same animals. As a validation of our tracking method, we demonstrate that out methods can automatically extract both movement kinematics and behavioral states (movement patterns, social events) during spontaneous social interactions. Moreover, data generated by our tracking method are compatible with two types of analyses: (i) Modern data-mining methods for unsupervised discovery of behavioral states (specifically, state space modeling) and (ii) Template-based analysis, detecting behaviors of interest based on prior knowledge. Template-based methods are better suited than unsupervised methods for detecting certain types of behaviors (see Discussion), so it is a major advantage that our data are amenable to both types of analysis. Both types of analysis are quantitative and fully automatic, solving two major issues with manual labeling (subjective experimenter bias and limited throughput).

To demonstrate template-based analysis, we defined social behaviors of interest as templates and matched these templates to tracked data. We know that anogenital sniffing[33] and nose-to-nose touch[34] are prominent events in rodent social behavior, so we designed a template to detect these events. In this template, we exploited the fact that we could easily calculate both body postures and movement kinematics, in the reference frame of each animal's own body. For every frame, we first extracted the 3D coordinates of the body model

skeleton (**Supplementary Fig. 5**). From these skeleton coordinates, we calculated the position (**Fig. 6a**) and a three-dimensional speed vector for each mouse ('forward speed', along the hip ellipsoid, 'left speed' perpendicular the body axis and 'up speed' along the z-axis; **Fig. 6b, Supplementary Fig. 8**). We also calculated three instantaneous 'social distances', defined as the 3D distance between the tip of each animal's noses ('nose-to-nose'; **Fig. 6b**), and from the tip of each animal's nose to the posterior end of the conspecific's hip ellipsoid ('nose-to-tail'; **Fig. 6b**). From these social distances, we could automatically detect when the mouse bodies were in a nose-to-nose or a nose-to-tail configuration, and in a single 20 min experimental session, we observed multiple nose-to-nose and nose-to-tail events (**Fig. 6c**). It is straightforward to further subdivide these social events by body postures and kinematics, to, e.g., separate stationary nose-to-tail configurations (anogenital sniffing/grooming) and nose-to-tail configurations during locomotion (social following).

To demonstrate unsupervised behavioral state discovery, we used GPU-accelerated probabilistic programming[35] and state space modeling to automatically detect and label movement states. To discover types locomotor behavior, we fitted a 'sticky' multivariate hidden Markov model[36] to the two components of the speed vector that lie in the *xy*-plane (**Supplementary Fig. 9a-h**). With five hidden states, this model yielded interpretable movement patterns that correspond to known mouse locomotor 'syllables': resting (no movement), turning left and right, and moving forward at slow and fast speeds (**Fig. 6d**). Fitting a similar model with three hidden states to the z-component of the speed vector (**Supplementary Fig. 9i-n**) yielded interpretable and known 'rearing syllables': rest, rearing up and ducking down (**Fig. 6e**). Using the maximum a posterior probability from these fitted models, we could automatically generate locomotor ethograms and rearing ethograms for the two mice (**Fig. 6b**).

In line with previous observations, we found that movement bouts were short (medians, rest/left/right/fwd/ffwd: 0.83/0.50/0.52/0.45/0.68 s, a 'sub-second' timescale[13]). In the locomotion ethograms,

bouts of rest were longer than bouts of movement (all $p < 0.05$, Mann-Whitney U-test; **Fig. 6f**) and bouts of fast forward locomotion was longer than other types of locomotion (all $p < 0.001$, Mann-Whitney U-test; **Fig. 6f**). In the rearing ethograms, the distribution of rests was very wide, consisting of both long (~seconds) and very short (~tenths of a second) periods of rest (**Fig. 6g**). As expected, by plotting the rearing height against the duration of rearing syllables, we found that short rests in rearing were associated with standing high on the hind legs (the mouse rears up, waits for a brief moment before ducking back down), while longer rests happened when the mouse was on the ground (ation of rearing syllabpearman rank; **Fig. 6h**). Like the movement types and durations, the transition probabilities from the fitted hidden Markov models were also in agreement with known behavioral patterns. In the locomotion model, for example, the most likely transition from "rest" was to "slow forward". From "slow forward", the mouse was likely to transition to "turning left", "fast forward" or "turning right", it was unlikely to transition directly from "fast forward" to "rest" or from "turning left" to "turning right, and so on (**Supplementary Fig. 9o,p**).

Finally, our method recovered the 3D head direction of both animals. The head direction of the implanted animal was given by the skeleton of the body model (the implant is fixed to the head). As mentioned above, we exploited the rotational symmetry of the body model of the conspecific to decrease the dimensionality of the search space during tracking (**Fig. 4c**). However, from the 3D coordinates of the detected key-points, we could still infer the 3D head direction (**Supplementary Fig. 10**) and it matched known mouse behavior (**Supplementary Fig. 11**). This feature is of particular interest to social neuroscience, since – while rodents clearly respond to the behavior of conspecifics – we are still only beginning to discover how the rodent brain encodes the gaze direction and body postures of others[37].

DISCUSSION

We combined 3D videography, deep learning and GPU-accelerated robust optimization to estimate the posture dynamics of multiple freely-moving mice, engaging in naturalistic social interactions. Our method is cost-effective (requiring only inexpensive consumer depth cameras and a GPU), has high spatiotemporal precision, is compatible with neural implants for continuous electrophysiological recordings, and tracks unmarked animals of the same coat color (e.g., enabling behavioral studies in transgenic mice). Our method is fully unsupervised, which makes the method scalable across multiple PCs or GPUs. Unsupervised tracking allows us to investigate social behavior across long behavioral time scales – beyond what is feasible with manual annotation – to elucidate developmental trajectories, dynamics of social learning, or individual differences among animals[38,39], among other types of questions. Finally, our method uses no message-passing from future frames, but only relies on past data, which makes the method a promising starting point for real-time tracking.

Reasons to study naturalistic social interactions in 3D

Social dysfunctions can be devastating symptoms in a multitude of mental conditions, including autism spectrum disorders, social anxiety, depression, and personality disorders[40]. Social interactions also powerfully impact somatic physiology, and social interactions are emerging as a promising protective and therapeutic element in somatic conditions, such as inflammation[41] and chronic pain[42]. These disorders have high incidence but generally lack effective treatment options, largely because even the neurobiological basis of 'healthy' social behavior is poorly understood.

In neuroscience and experimental biomedicine, there has been major technical progress in recording techniques for freely moving animals, with high-density electrodes[43,44], and head-mounted multi-photon microscopes[45,46]. Moreover, newer methods are being developed for tracking complex patterns of animal behav-

445 ior[47,48]. Here we provide a new method to complement these approaches for feasible, quantitative, and auto-
446 mated behavioral analysis. A major next step for future work is to apply such algorithms to animal behavior
447 in different conditions. For example, the algorithm can easily be adapted to track other animal body shapes
448 such as juvenile mice or other species, or movable, deformable objects that might be important for foraging
449 or other behaviors in complex environments.
450
451 ***What is the advantage of a body model?***
452 In automated analysis of behavioral states, there are three main approaches: nonlinear clustering[7,49–55],
453 probabilistic state space modeling[13,56–59] and template matching[5,11,26]. In nonlinear clustering, tracked body
454 coordinates (and derived quantities, such as time derivatives or spectral components) are segmented into dis-
455 crete behaviors by density-based clustering, typically after nonlinear projection down to a low-dimensional
456 2D space[7,49–53,55] or 3D space[60]. Density-based clusters are manually inspected and curated, such that clusters
457 judged as similar are merged and clusters are assigned names (e.g., 'locomotion', 'grooming', etc.). This
458 approach is simple and robust, but still flexible enough to discover behavioral changes due to interactions
459 with conspecifics[52,53]. A limitation of this approach, however, is that nonlinear clustering directly on the
460 tracked kinematic features does not allow explicit modeling of history dependence or hierarchical structure.
461
462 In principle, state space models are highly expressive, allowing for complex nested structures of hidden states,
463 observational models, covariance structures and temporal dependencies, e.g., autoregressive terms[13], linear
464 dynamics[61], and hierarchies[39]. In practice, however, state space models are not easily fit to data. Complex
465 models quickly become prohibitively computationally expensive and approximate fitting strategies, such as
466 variational inference, show poor convergence for complex models[62]. Finally, and most importantly, model
467 comparison is still an unsolved problem and methods guaranteed to discover a 'true' latent structure from data
468 (e.g., number of states or transition graph structure) do not exist[36,63].

469

470 Neither state space modeling nor nonlinear clustering requires an explicit body model of the animal. In prin-
471 ciple, any tracked points on the body can be used, as long as their statistics differ enough between distinct
472 behaviors that they still form significantly different clusters. What is the advantage of an actual body model?
473 First, even though in principle, any tracked points on an animal can be used for unsupervised behavioral
474 analysis – body posture features based on the actual body geometry are often both more powerful and more
475 interpretable in practice[64]. For example, a body model lets us specify appropriate generative models for state
476 space modeling. Specifying appropriate probability distributions over variables with unknown densities and
477 covariances is not straightforward, but specifying priors over the parameters of an understandable body model
478 with known physical constraints is an advantage for this type of analysis.

479

480 Second, unsupervised methods are not well-suited for the discovery of behaviors that happen rarely or are
481 kinematically similar to other behaviors. In order for rare or kindred behaviors to form identifiable clusters,
482 it may be necessary to collect extremely large datasets, beyond what is realistic to collect in typical neurosci-
483 ence experiments, and beyond what it is computationally feasible to analyze. These complications get even
484 worse when considering the joint statistics of multiple animals. A physical model of the bodies allows us to
485 overcome these problems by simple template matching and we can easily specify templates that flag social
486 poses (nose-to-nose and nose-to-tail touch, in our example). Moreover, of particular interest to neural record-
487 ings, a body model lets us regress out proprioceptive and movement-related signals, known to align to an
488 egocentric, body-centered reference frame[3,65–67].

489

The tracking algorithm is easy to update

Our data acquisition pipeline and tracking algorithm is open source and implemented in Python, a widely used programming language in machine learning. This makes it easy to update the algorithm with methodological advances in the field. Deep learning in 3D is still thought to be in its infancy[68], but along with technical developments in depth imaging hardware (for video games, self-driving cars and other industrial applications), there are exciting developments in analysis, including deep leaning methods for detection of deformable objects from image[69–71] and point-cloud[72,73] data, geometric and graph-based tricks for GPU-accelerated analysis of 3D data[74–76], and methods for physical modeling of deformable bodies[77–79].

Our pre-processing pipeline generates a highly compressed data representation, consisting of a 3D point-cloud and key-points, sampled at 60 fps. Storing and sharing the raw depth and color video frames from multiple cameras, across long behavioral time scales (hours to days) is not feasible, but storing and sharing the compressed, pre-processed data format is possible. This is a major advantage for two reasons. First, as deep learning methods for 3D data improve, old datasets can be re-analyzed and mined for new insights. Second, abundantly available 3D datasets of animal behavior let machine learning researchers test and benchmark new methods on experimental data. This enables a development cycle for improving behavioral analysis, by which adoption of 3D-videography-based behavioral tracking in biology can contribute positively to future methodological developments in the machine learning community.

Moving towards real-time tracking

Our algorithm is unsupervised, does not use any message-passing from future frames, and robustly recovers from tracking mistakes. Since the algorithm relies purely on past data, it is in principle possible to run the algorithm in real-time. Currently, the processing time per frame is higher than the camera frame rate (60 frames/s). However, our algorithm is not fully optimized and there are multiple speed improvements, which

514 are straightforward to implement. For example, in the current version of the algorithm, we first record the
515 images to disk, and then read and pre-process the images later. This is convenient for algorithm development
516 and exploration, but writing and reading the images to disk, and moving them onto and off a GPU are time-
517 intensive steps. During pre-processing, it is possible to increase the speed and precision of key-point detection
518 by implementing peak detection as a convolutional layer[8] and it may be possible to perform key-point detec-
519 tion directly on the 1-channel infrared images instead of the 3-channel color images. Grayscale infrared im-
520 ages are faster to process and we would be able to perform experiments in visual darkness, which is less
521 stressful and more appropriate for nocturnal mice. As shown in **Figure 1d**, the infrared images are 'contami-
522 nated' with a dotted grid of points from the infrared laser projector, but – as evidenced by the usefulness of
523 pixel dropout in image data augmentation[17] (**Supplementary Fig. 3**) – it should be possible to train a network
524 to 'disregard' the dotted grid during key-point detection. We may also be able to reduce the number of required
525 particle filtering steps between frames. For example, we could force a network to learn to draw particle sam-
526 ples more intelligently, based on learned covariance patterns in natural mouse movement. Additionally, we
527 could try to combine our particle filter with gradient-based optimization methods (start with a particle filter
528 step to search for loss function basins and then use fast, parallel Levenberg-Marquardt steps[18] to quickly move
529 particles to the bottom of the basins, for example).
530
531 Beyond these optimizations, tracking at a lower frame rate would allow more data processing time per frame.
532 Our robust, particle-filter-based tracker with online forecasting is an ideal candidate for this task. Going for-
533 ward, we will investigate the performance of the algorithm at lower frame rates and explore ways to increase
534 tracking robustness further, by implementing other recently described tracking algorithm tricks, such as using
535 the optical flow between video frames to link key-points together in multi-animal tracking ('SLEAP'[7,80]),
536 real-time painting-in of depth artifacts[81] and even better online trajectory forecasting, for example using deep

Gaussian processes[82] or a neural network trained to propose trajectories based on mouse behavior. Experimentation and optimization is clearly needed, but our fully unsupervised algorithm – requiring data transfer from only a few cameras, with deep convolutional networks, physical modeling and particle filter tracking implemented as tensor algebra on the same GPU – is a promising starting point for the development of real-time, multi-animal 3D tracking.

REFERENCES

1. Crusio, W. E., Sluyter, F. & Gerlai, R. T. Ethogram of the mouse. in *Behavioral Genetics of the Mouse* 17–22 (Cambridge Univeristy Press, 2013). doi:10.1017/CBO9781139541022.004.
2. Angelaki, D. E. *et al.* A gravity-based three-dimensional compass in the mouse brain. *Nat. Commun.* **11**, 1855 (2020).
3. Mimica, B., Dunn, B. A., Tombaz, T., Bojja, V. P. T. N. C. S. & Whitlock, J. R. Efficient cortical coding of 3D posture in freely behaving rats. *Science* **362**, 584–589 (2018).
4. Shemesh, Y. *et al.* High-order social interactions in groups of mice. *eLife* **2**, e00759 (2013).
5. Weissbrod, A. *et al.* Automated long-term tracking and social behavioural phenotyping of animal colonies within a semi-natural environment. *Nat. Commun.* **4**, 1–10 (2013).
6. Mathis, A. *et al.* DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nat. Neurosci.* **21**, 1281–1289 (2018).
7. Pereira, T. D. *et al.* Fast animal pose estimation using deep neural networks. *Nat. Methods* **16**, 117–125 (2019).
8. Graving, J. M. *et al.* DeepPoseKit, a software toolkit for fast and robust animal pose estimation using deep learning. *eLife* **8**, e47994 (2019).
9. Nath, T. *et al.* Using DeepLabCut for 3D markerless pose estimation across species and behaviors. *Nat. Protoc.* **14**, 2152–2176 (2019).
10. Bala, P. C. *et al.* OpenMonkeyStudio: Automated Markerless Pose Estimation in Freely Moving Macaques. *bioRxiv* 2020.01.31.928861 (2020) doi:10.1101/2020.01.31.928861.
11. Matsumoto, J. *et al.* A 3D-Video-Based Computerized Analysis of Social and Sexual Interactions in Rats. *PLOS ONE* **8**, e78460 (2013).
12. Nakamura, T. *et al.* A Markerless 3D Computerized Motion Capture System Incorporating a Skeleton Model for Monkeys. *PLOS ONE* **11**, e0166154 (2016).
13. Wiltschko, A. B. *et al.* Mapping Sub-Second Structure in Mouse Behavior. *Neuron* **88**, 1121–1135 (2015).
14. Newell, A., Yang, K. & Deng, J. Stacked Hourglass Networks for Human Pose Estimation. *ArXiv160306937 Cs* (2016).
15. Günel, S. *et al.* DeepFly3D, a deep learning-based approach for 3D limb and appendage tracking in tethered, adult Drosophila. *eLife* **8**, e48571 (2019).
16. Cao, Z., Simon, T., Wei, S.-E. & Sheikh, Y. Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *ArXiv161108050 Cs* (2017).
17. Shorten, C. & Khoshgoftaar, T. M. A survey on Image Data Augmentation for Deep Learning. *J. Big Data* **6**, 60 (2019).
18. Transtrum, M. K., Machta, B. B. & Sethna, J. P. Geometry of nonlinear least squares with applications to sloppy models and optimization. *Phys. Rev. E* **83**, 036701 (2011).
19. Hart, J. C. Distance to an ellipsoid. in *Graphics Gems* (ed. Heckbert, P.) vol. 1994 113–119 (Academic Press).
20. Kleinsteuber, M. & Hüper, K. Approximate Geometric Ellipsoid Fitting: A CG-Approach. in *Recent Advances in Optimization and its Applications in Engineering* (eds. Diehl, M., Glineur, F., Jarlebring, E. & Michiels, W.) 73–82 (Springer, 2010). doi:10.1007/978-3-642-12598-0_7.
21. Wang, W., Wang, J. & Kim, M.-S. An algebraic condition for the separation of two ellipsoids. *Comput. Aided Geom. Des.* **18**, 531–539 (2001).
22. Choset, H. M. *Principles of robot motion: theory, algorithms, and implementation*. (MIT Press, 2005).
23. Deutscher, J. & Reid, I. Articulated Body Motion Capture by Stochastic Search. *Int. J. Comput. Vis.* **61**, 185–205 (2005).

24. Lázaro-Gredilla, M., Van Vaerenbergh, S. & Santamaría, I. A Bayesian approach to tracking with kernel recursive least-squares. in *2011 IEEE International Workshop on Machine Learning for Signal Processing* 1–6 (2011). doi:10.1109/MLSP.2011.6064585.
25. Paszke, A. *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. in *Advances in Neural Information Processing Systems 32* (eds. Wallach, H. et al.) 8024–8035 (Curran Associates, Inc., 2019).
26. Chaumont, F. de *et al.* Real-time analysis of the behaviour of groups of mice via a depth-sensing camera and machine learning. *Nat. Biomed. Eng.* **3**, 930–942 (2019).
27. Sobol, I. M. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Comput. Math. Math. Phys.* **7**, 86–112 (1967).
28. Das, G. Top-k Algorithms and Applications. in *Database Systems for Advanced Applications* (eds. Zhou, X., Yokota, H., Deng, K. & Liu, Q.) 789–792 (Springer, 2009). doi:10.1007/978-3-642-00887-0_74.
29. Murray, L. M., Lee, A. & Jacob, P. E. Parallel Resampling in the Particle Filter. *J. Comput. Graph. Stat.* **25**, 789–805 (2016).
30. Rauch, H. E., Tung, F. & Striebel, C. T. Maximum likelihood estimates of linear dynamic systems. *AIAA J.* **3**, 1445–1450 (1965).
31. Simon, D. *Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches*. (John Wiley & Sons, Inc., 2006). doi:10.1002/0470045345.
32. Landis, M. F., Cheng, Y., Crassidis, J. L. & Oshman, Y. Averaging quaternions. *J. Guid. Control Dyn.* **30**, 1193–1197 (2007).
33. Barnett, S. A. *The rat: A study in behaviour*. xvi, 248 (Aldine, 1963).
34. Wolfe, J., Mende, C. & Brecht, M. Social facial touch in rats. *Behav. Neurosci.* **125**, 900–910 (2011).
35. Bingham, E. *et al.* Pyro: Deep Universal Probabilistic Programming. *ArXiv181009538 Cs Stat* (2018).
36. Fox, E., Sudderth, E., Jordan, M. & Willsky, A. Bayesian Nonparametric Methods for Learning Markov Switching Processes. *IEEE Signal Process. Mag.* 5563110 (2010) doi:10.1109/MSP.2010.937999.
37. Tombaz, T. *et al.* Action representation in the mouse parieto-frontal network. *Sci. Rep.* **10**, 1–14 (2020).
38. Díaz López, B. When personality matters: personality and social structure in wild bottlenose dolphins, Tursiops truncatus. *Anim. Behav.* **163**, 73–84 (2020).
39. Tao, L., Ozarkar, S., Beck, J. M. & Bhandawat, V. Statistical structure of locomotion and its modulation by odors. *eLife* **8**, e41235 (2019).
40. Porcelli, S. *et al.* Social brain, social dysfunction and social withdrawal. *Neurosci. Biobehav. Rev.* **97**, 10–33 (2019).
41. Uchino, B. N. *et al.* Social support, social integration, and inflammatory cytokines: A meta-analysis. *Health Psychol. Off. J. Div. Health Psychol. Am. Psychol. Assoc.* **37**, 462–471 (2018).
42. Che, X., Cash, R., Ng, S. K., Fitzgerald, P. & Fitzgibbon, B. M. A Systematic Review of the Processes Underlying the Main and the Buffering Effect of Social Support on the Experience of Pain. *Clin. J. Pain* **34**, 1061–1076 (2018).
43. Buzsáki, G. *et al.* Tools for Probing Local Circuits: High-Density Silicon Probes Combined with Optogenetics. *Neuron* **86**, 92–105 (2015).
44. Steinmetz, N. A., Koch, C., Harris, K. D. & Carandini, M. Challenges and opportunities for large-scale electrophysiology with Neuropixels probes. *Curr. Opin. Neurobiol.* **50**, 92–100 (2018).
45. Aharoni, D. & Hoogland, T. M. Circuit Investigations With Open-Source Miniaturized Microscopes: Past, Present and Future. *Front. Cell. Neurosci.* **13**, (2019).
46. Klioutchnikov, A. *et al.* Three-photon head-mounted microscope for imaging deep cortical layers in

freely moving rats. *Nat. Methods* **17**, 509–513 (2020).

47. Krakauer, J. W., Ghazanfar, A. A., Gomez-Marin, A., MacIver, M. A. & Poeppel, D. Neuroscience Needs Behavior: Correcting a Reductionist Bias. *Neuron* **93**, 480–490 (2017).
48. Datta, S. R., Anderson, D. J., Branson, K., Perona, P. & Leifer, A. Computational Neuroethology: A Call to Action. *Neuron* **104**, 11–24 (2019).
49. Berman, G. J., Choi, D. M., Bialek, W. & Shaevitz, J. W. Mapping the stereotyped behaviour of freely moving fruit flies. *J. R. Soc. Interface* **11**, 20140672 (2014).
50. Berman, G. J., Bialek, W. & Shaevitz, J. W. Predictability and hierarchy in Drosophila behavior. *Proc. Natl. Acad. Sci.* **113**, 11943–11948 (2016).
51. Werkhoven, Z. *et al.* The structure of behavioral variation within a genotype. *bioRxiv* 779363 (2019) doi:10.1101/779363.
52. Klibaite, U., Berman, G. J., Cande, J., Stern, D. L. & Shaevitz, J. W. An unsupervised method for quantifying the behavior of paired animals. *Phys. Biol.* **14**, 015006 (2017).
53. Klibaite, U. & Shaevitz, J. W. Interacting fruit flies synchronize behavior. *bioRxiv* 545483 (2019) doi:10.1101/545483.
54. Braun, E., Geurten, B. & Egelhaaf, M. Identifying Prototypical Components in Behaviour Using Clustering Algorithms. *PLOS ONE* **5**, e9361 (2010).
55. Mearns, D. S., Donovan, J. C., Fernandes, A. M., Semmelhack, J. L. & Baier, H. Deconstructing Hunting Behavior Reveals a Tightly Coupled Stimulus-Response Loop. *Curr. Biol.* **30**, 54-69.e9 (2020).
56. Markowitz, J. E. *et al.* The Striatum Organizes 3D Behavior via Moment-to-Moment Action Selection. *Cell* **174**, 44-58.e17 (2018).
57. Johnson, R. E. *et al.* Probabilistic Models of Larval Zebrafish Behavior Reveal Structure on Many Scales. *Curr. Biol.* **30**, 70-82.e4 (2020).
58. DeRuiter, S. L. *et al.* A multivariate mixed hidden Markov model to analyze blue whale diving behaviour during controlled sound exposures. *ArXiv160206570 Q-Bio Stat* (2016).
59. Adam, T. *et al.* Joint modelling of multi-scale animal movement data using hierarchical hidden Markov models. *Methods Ecol. Evol.* **10**, 1536–1550 (2019).
60. Hsu, A. I. & Yttri, E. A. *B-SOiD: An Open Source Unsupervised Algorithm for Discovery of Spontaneous Behaviors*. http://biorxiv.org/lookup/doi/10.1101/770271 (2019) doi:10.1101/770271.
61. Linderman, S. W. *et al.* Recurrent switching linear dynamical systems. *ArXiv161008466 Stat* (2016).
62. Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. Variational Inference: A Review for Statisticians. *J. Am. Stat. Assoc.* **112**, 859–877 (2017).
63. Pohle, J., Langrock, R., van Beest, F. M. & Schmidt, N. M. Selecting the Number of States in Hidden Markov Models: Pragmatic Solutions Illustrated Using Animal Movement. *J. Agric. Biol. Environ. Stat.* **22**, 270–293 (2017).
64. Javer, A., Ripoll-Sánchez, L. & Brown, A. E. X. Powerful and interpretable behavioural features for quantitative phenotyping of Caenorhabditis elegans. *Philos. Trans. R. Soc. B Biol. Sci.* **373**, 20170375 (2018).
65. Kropff, E., Carmichael, J. E., Moser, M.-B. & Moser, E. I. Speed cells in the medial entorhinal cortex. *Nature* **523**, 419–424 (2015).
66. Georgopoulos, A. P., Kalaska, J. F., Caminiti, R. & Massey, J. T. On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *J. Neurosci.* **2**, 1527–1537 (1982).
67. Kalaska, J. F. The representation of arm movements in postcentral and parietal cortex. *Can. J. Physiol. Pharmacol.* **66**, 455–463 (1988).
68. Guo, Y. *et al.* Deep Learning for 3D Point Clouds: A Survey. *ArXiv191212033 Cs Eess* (2019).

69. Novotny, D., Ravi, N., Graham, B., Neverova, N. & Vedaldi, A. C3DPO: Canonical 3D Pose Networks for Non-Rigid Structure From Motion. *ArXiv190902533 Cs* (2019).
70. Sanakoyeu, A., Khalidov, V., McCarthy, M. S., Vedaldi, A. & Neverova, N. Transferring Dense Pose to Proximal Animal Classes. *ArXiv200300080 Cs* (2020).
71. Zuffi, S., Kanazawa, A., Berger-Wolf, T. & Black, M. J. Three-D Safari: Learning to Estimate Zebra Pose, Shape, and Texture from Images 'In the Wild'. *ArXiv190807201 Cs* (2019).
72. Gkioxari, G., Malik, J. & Johnson, J. Mesh R-CNN. *ArXiv190602739 Cs* (2020).
73. Niemeyer, M., Mescheder, L., Oechsle, M. & Geiger, A. Occupancy Flow: 4D Reconstruction by Learning Particle Dynamics. in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* 5378–5388 (IEEE, 2019). doi:10.1109/ICCV.2019.00548.
74. Fey, M. & Lenssen, J. E. Fast Graph Representation Learning with PyTorch Geometric. *ArXiv190302428 Cs Stat* (2019).
75. Jatavallabhula, K. M. *et al.* Kaolin: A PyTorch Library for Accelerating 3D Deep Learning Research. *ArXiv191105063 Cs* (2019).
76. Ravi, N. *et al. PyTorch3D*. (2020).
77. Deng, B. *et al.* NASA: Neural Articulated Shape Approximation. *ArXiv191203207 Cs* (2020).
78. Zuffi, S., Kanazawa, A., Jacobs, D. & Black, M. J. 3D Menagerie: Modeling the 3D shape and pose of animals. *ArXiv161107700 Cs* (2017).
79. Min, S., Won, J., Lee, S., Park, J. & Lee, J. SoftCon: Simulation and Control of Soft-Bodied Animals with Biomimetic Actuators. *ACM Trans. Graph.* **38**, (2019).
80. Pereira, T. D., Tabris, N., Turner, D., Shaevitz, J. & Murthy, M. https://sleap.ai/. https://sleap.ai/ (2020).
81. Gillis, W. *et al.* Revealing elements of naturalistic reinforcement learning through closed-loop action identification. in *2019 Neuroscience Meeting Planner* Program No. 146.17 (Society for Neuroscience, 2019).
82. Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q. & Wilson, A. G. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. *ArXiv180911165 Cs Stat* (2019).

ACKNOWLEDGEMENTS

This work was supported by The Novo Nordisk Foundation (C.L.E.), the BRAIN Initiative (NS107616 to R.C.F.) and a Howard Hughes Medical Institute Faculty Scholarship (R.C.F.).

AUTHOR CONTRIBUTIONS

C.L.E. designed and implemented the system, performed experiments, and analyzed the data. R.C.F. supervised the study. C.L.E and R.C.F. wrote the manuscript.

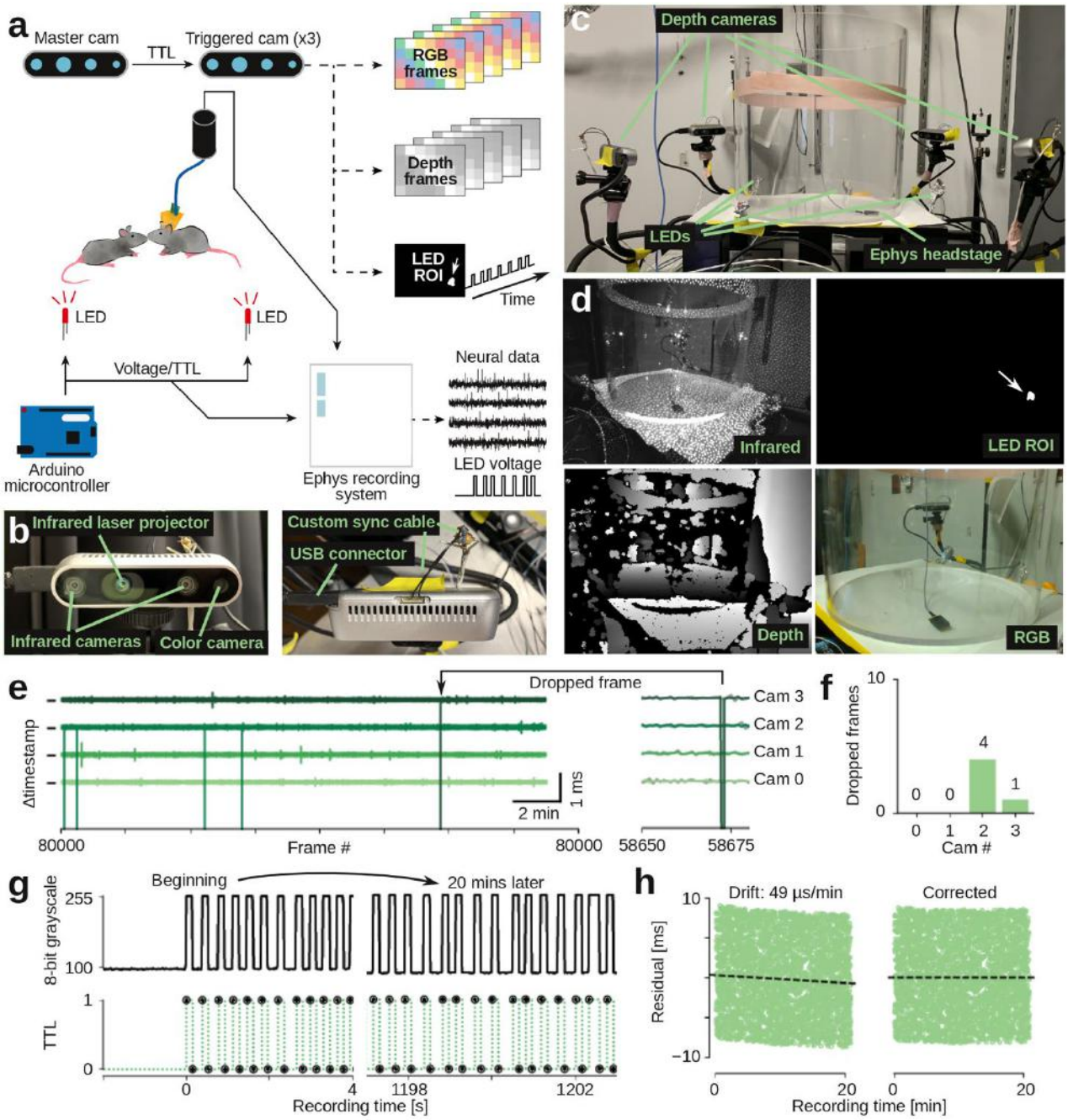
COMPETING INTERESTS

The authors declare no competing interests.

DATA AND CODE AVAILABILITY

All code and an example dataset were submitted with this manuscript. All code and an example dataset will be available on Github before or upon publication.

725 **FIGURES**



726

727 **Figure 1. Raw data acquisition, temporal alignment and recording stability. a,** Schematic of recording
728 setup, showing flow of synchronization pulses and raw data. We use a custom Python program to record RGB
729 images, depth images, and state (on/off) of synchronization LEDs from all four cameras. Neural data and TTL
730 state of LEDs are recorded with a standard electrophysiology recording system. We use a custom Python
731 program to record video frames over USB (60 frames/s) and automatically deliver LED synchronization

pulses with randomized delays via Arduino microcontroller. **b,** Close-up images of the depth cameras, showing the two infrared sensors, color sensor, and cables for data transfer and synchronization. **c,** Photograph of recording setup, showing the four depth cameras, synchronization LEDs, and circular behavioral arena (transparent acrylic, 12" diameter). **d,** Example raw data images (top left: single infrared image with visible infrared laser dots; top right: corresponding automatically-generated mask image for recording LED synchronization state (arrow, LED location); bottom left: corresponding depth image, estimated from binocular disparity between two infrared images; bottom right: corresponding color image). **e,** Inter-frame-interval from four cameras (21 min of recording). Vertical ticks indicate 16.66 ms (corresponding to 60 frames/s), individual cameras are colored and vertically offset. Frame rate is very stable (jitter across all cameras: ±26 μs). Arrow, example dropped frame. **f,** Number of dropped frames across the example 21 min recording. **g,** Top row, LED state (on/off) as captured by one camera (the 8-bit value of central pixel of LED ROI mask), at start of recording and after 20 minutes of recording. Bottom row, aligned LED trace, as recorded by electrophysiology recording system. **h,** Temporal residuals between recorded camera LED trace (**g**, top) and recorded TTL LED trace (**g**, bottom) are stable, but drift slightly (49 μs/min, left panel). We can automatically detect and correct for this small drift (right panel).

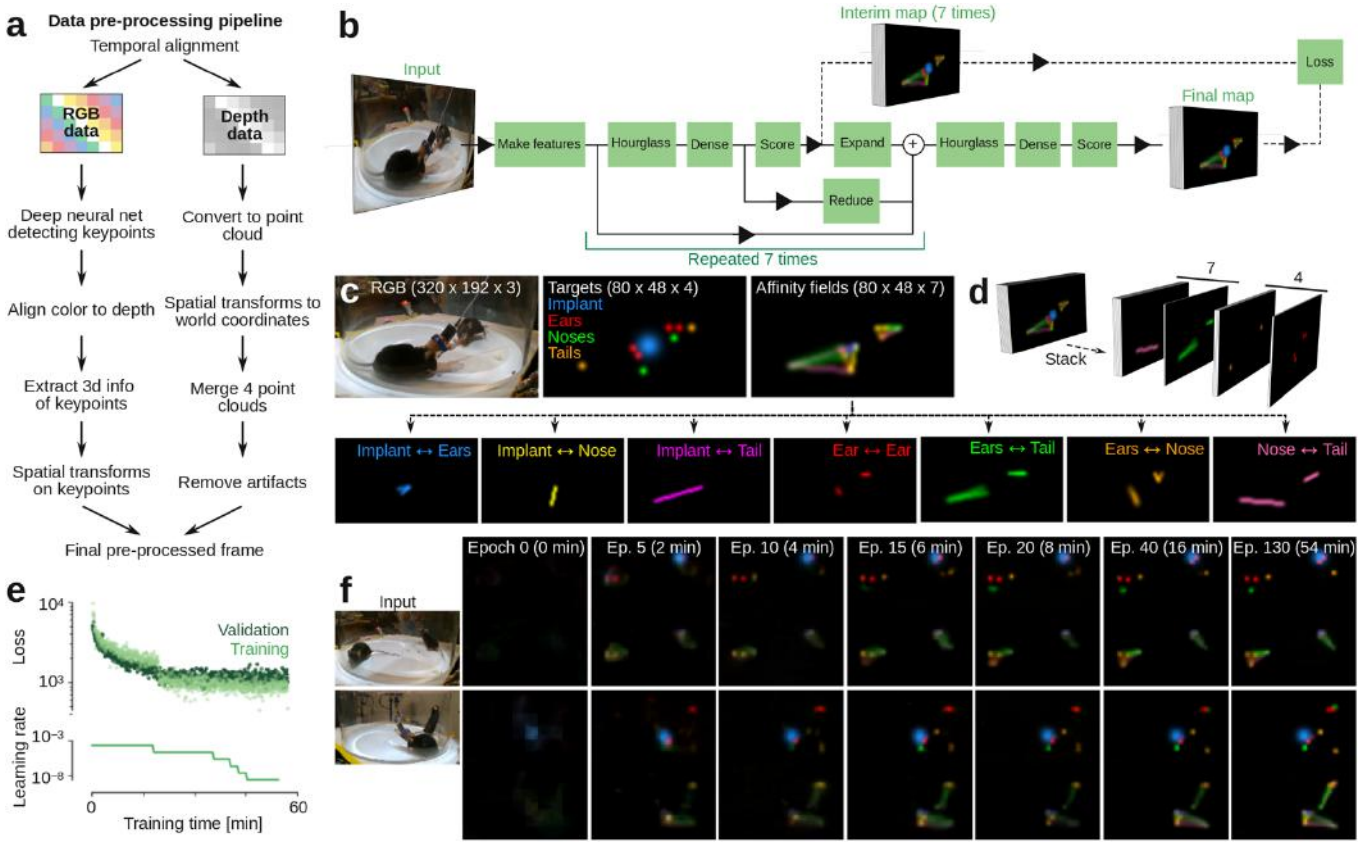
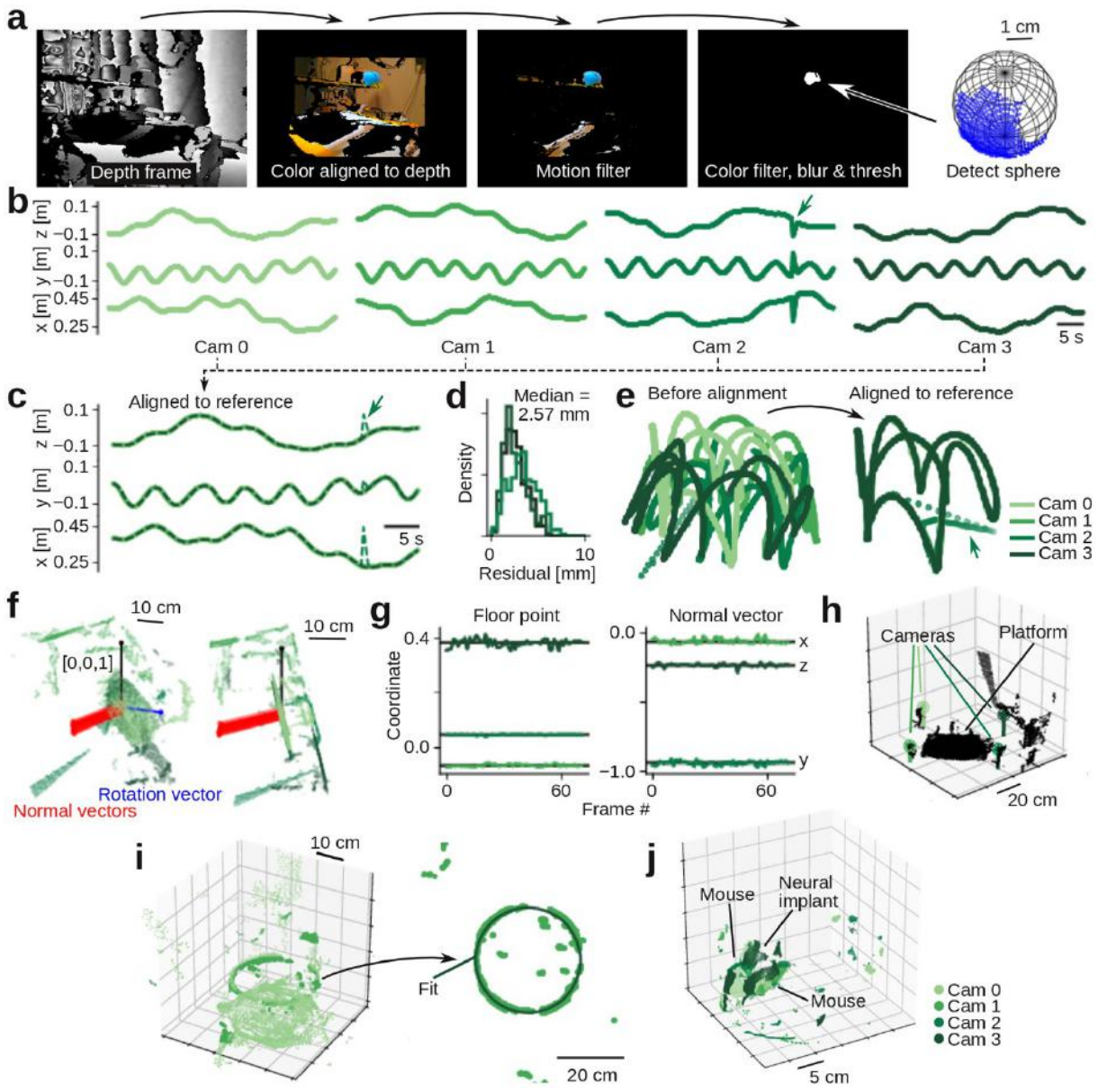


Figure 2. Detection of body key-points with a deep convolutional neural network. a, Workflow for pre-processing of raw image data. **b**, Example training data for the deep convolutional neural network. The network is trained to output four types of body key-points (Implant, Ears, Noses, Tails) and seven 1-D affinity fields, connecting key-points within each body. **c**, Example of full training target tensor. **d**, The 'stacked hourglass' convolutional network architecture. Each 'hourglass' block of the network uses pooling and upsampling to incorporate both fine (high-resolution) and large-scale (low-resolution) information in the target prediction. The hourglass and scoring blocks are repeated seven times (seven 'stacks'), such that intermediate key-point and affinity field predictions feed into subsequent hourglass blocks. Both the intermediate and final target maps contribute to the training loss, but only the final output map is used for prediction. **e**, Convergence plot of example training set. Top, loss function for each mini-batch of the training set (526 images) and validation set (50 images). Bottom, learning rate. Network loss is trained (plateaued) after ~ 60 minutes. **f**, Network performance as function of training epoch for two example images in the validation set. Left, input images; right, final output maps for key-points and affinity fields.



762

763 **Figure 3. Depth data alignment and pre-processing. a**, Calibration ball detection pipeline. We use a com-
764 bination of motion filtering, color filtering, and smoothing filters to detect and extract 3D ball surface. We
765 estimate 3D location of the ball by fitting a sphere to the extracted surface. **b**, Estimated 3D trajectories of
766 calibration ball as seen by the four cameras. One trajectory has an error (arrow) where ball trajectory was out
767 of view. **c**, Overlay of trajectories after alignment in time and space. Our alignment pipeline uses a robust
768 regression method and is insensitive to errors (arrow) in the calibration ball trajectory. **d**, Distribution of
769 residuals, using cam 0 as reference. **e**, Estimated trajectory in 3D space, before and after alignment of camera
770 data. **f**, Example frame used in automatic detection of the behavioral arena location. Show are pixels from the

four cameras, after alignment (green), estimated normal vectors to the behavioral platform floor (red), the estimated rotation vector (blue), and the reference vector (unit vector along z-axis, black). **g**, Estimated location (left) and normal vector (right) to the behavioral platform floor, across 60 random frames. **h**, Example frame, after rotating the platform into the xy-plane, and removing pixels below and outside the arena. Inferred camera locations are indicated with stick and ball. **i**, Automatic detection of behavioral arena location. **j**, Example 3D frame, showing merged data from four cameras, after automatic removal of the arena floor and imaging artifacts induced by the acrylic cylinder. Colors, which camera captured the pixels.

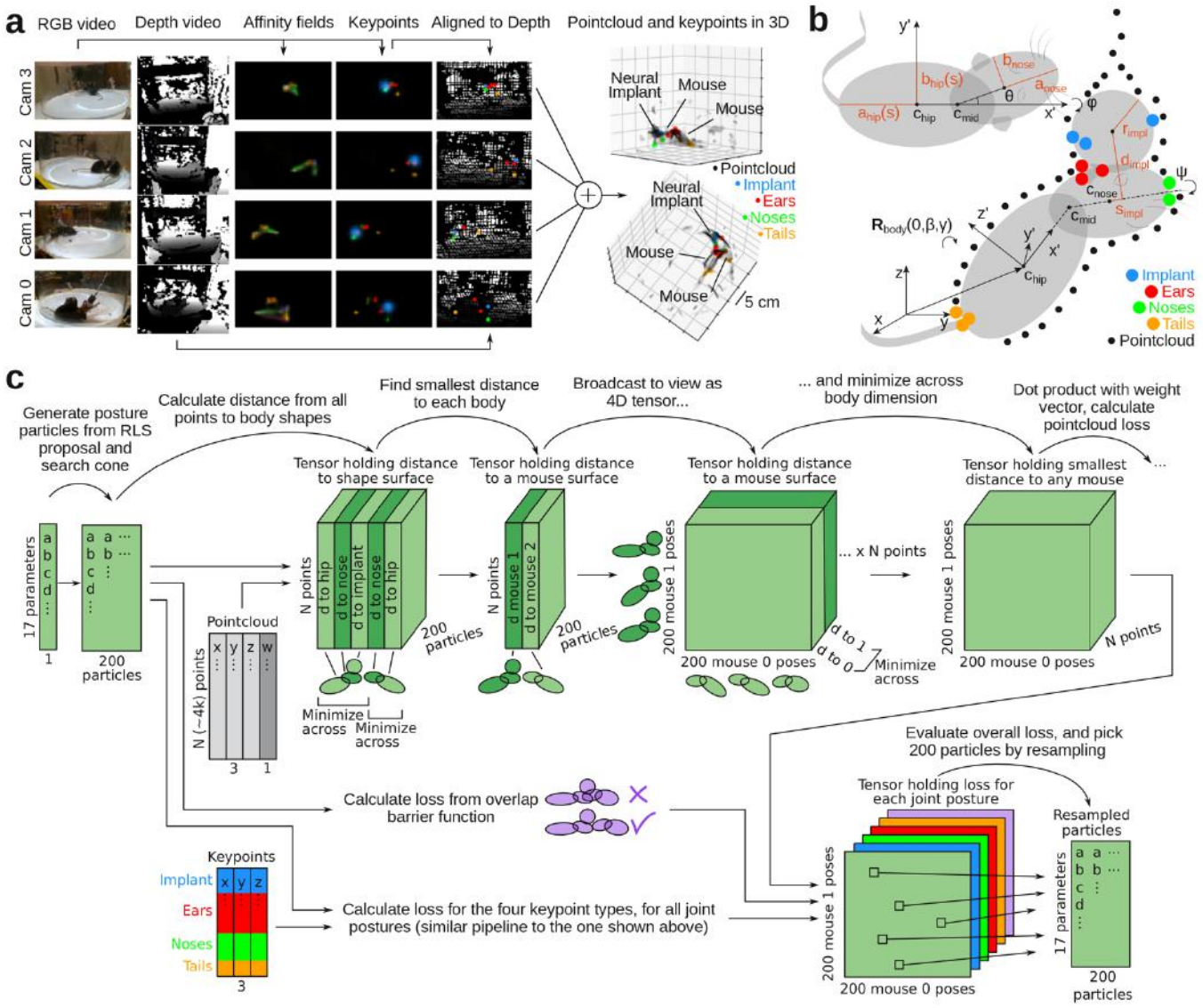


Figure 4. Mouse body model and fully vectorized, GPU-executable tracking algorithm. a, Full assembly pipeline for a single pre-processed data frame, going from raw RGB and depth images (left columns) to assembled 3D point-cloud (black dots, right) and body key-point positions in 3D space (colored dots, right). **b**, Schematic depiction of mouse body model (grey, deformable ellipsoids) and implant model (grey sphere), fit to point-cloud (black dots) and body key-points (colored dots). The loss function assigns loss to distance from the point-cloud to the body model surface (black arrows) and from key-point locations to landmark locations on the body model (e.g., from nose key-points to the tip of the nose ellipsoids; colored arrows). **c**, Schematic of loss function calculation and tracking algorithm. All operations implemented as GPU-accelerated tensor algebra.

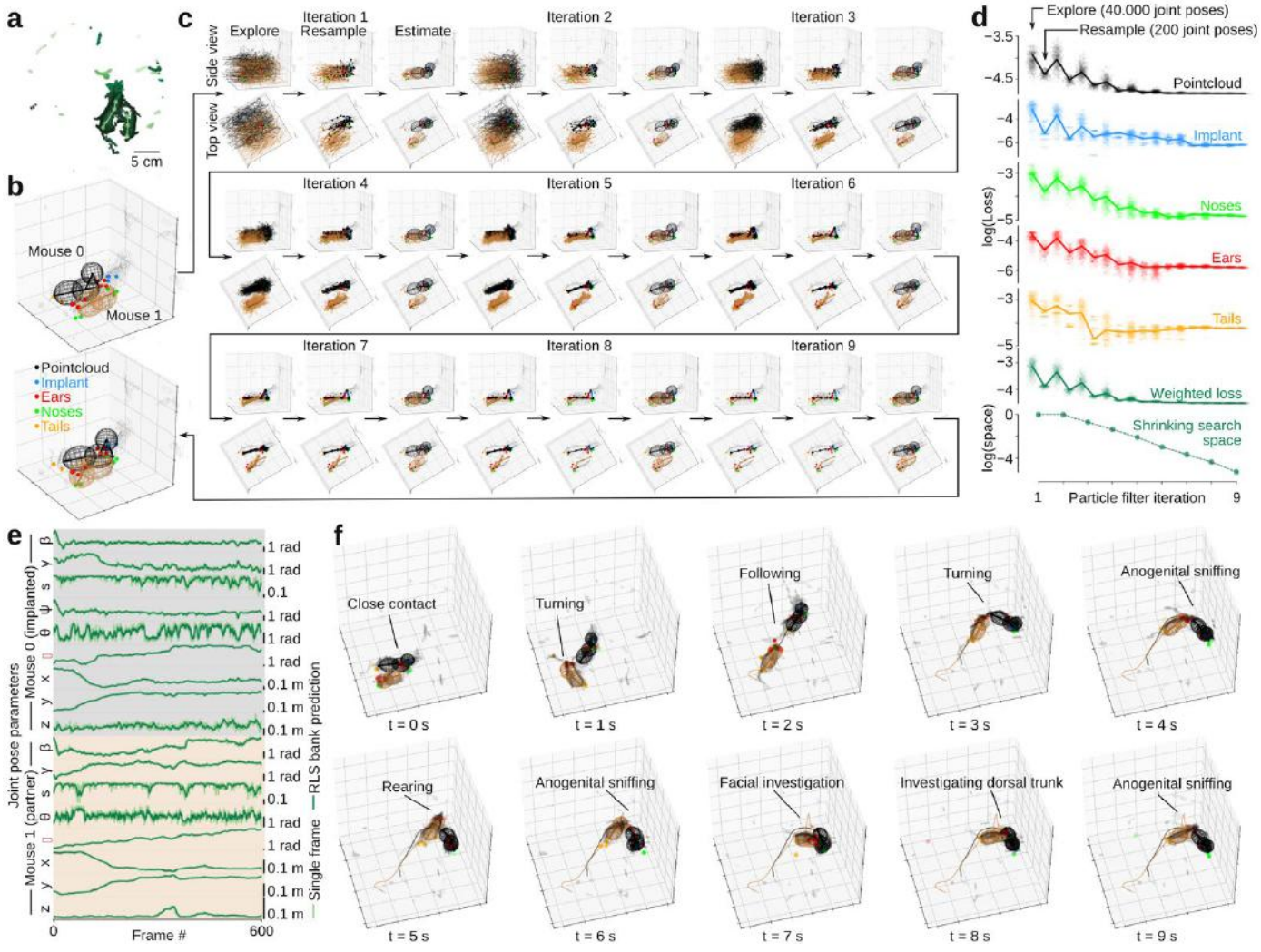


Figure 5. Particle filter convergence and examples of tracked behavioral data. a, Tracking algorithm is initialized by manual clicking of approximate locations of the two animals (light green dots, lines) on a top-down view of the behavioral arena (dark green dots, shade indicates z-coordinate). **b,** 3D view of initialized body model (top) and fitted body model (bottom) after running tracking algorithm on the frame. Black wireframe model, implanted mouse; brown wireframe model, partner animal. **c,** Particle filter state across 9 iterations of the fitting algorithm. After iteration 2, we shrink ('anneal') the exploration space with each step. **d,** Loss function values and size of filter search space across filter iterations. **e**, Tracked data (light green) and running adaptive estimate (dark green) across 600 frames (10 s). **f**, Data and fitted joint posture model, across 10 seconds of behavior. Trailing lines, location of hip ellipsoid center in last 10 seconds.

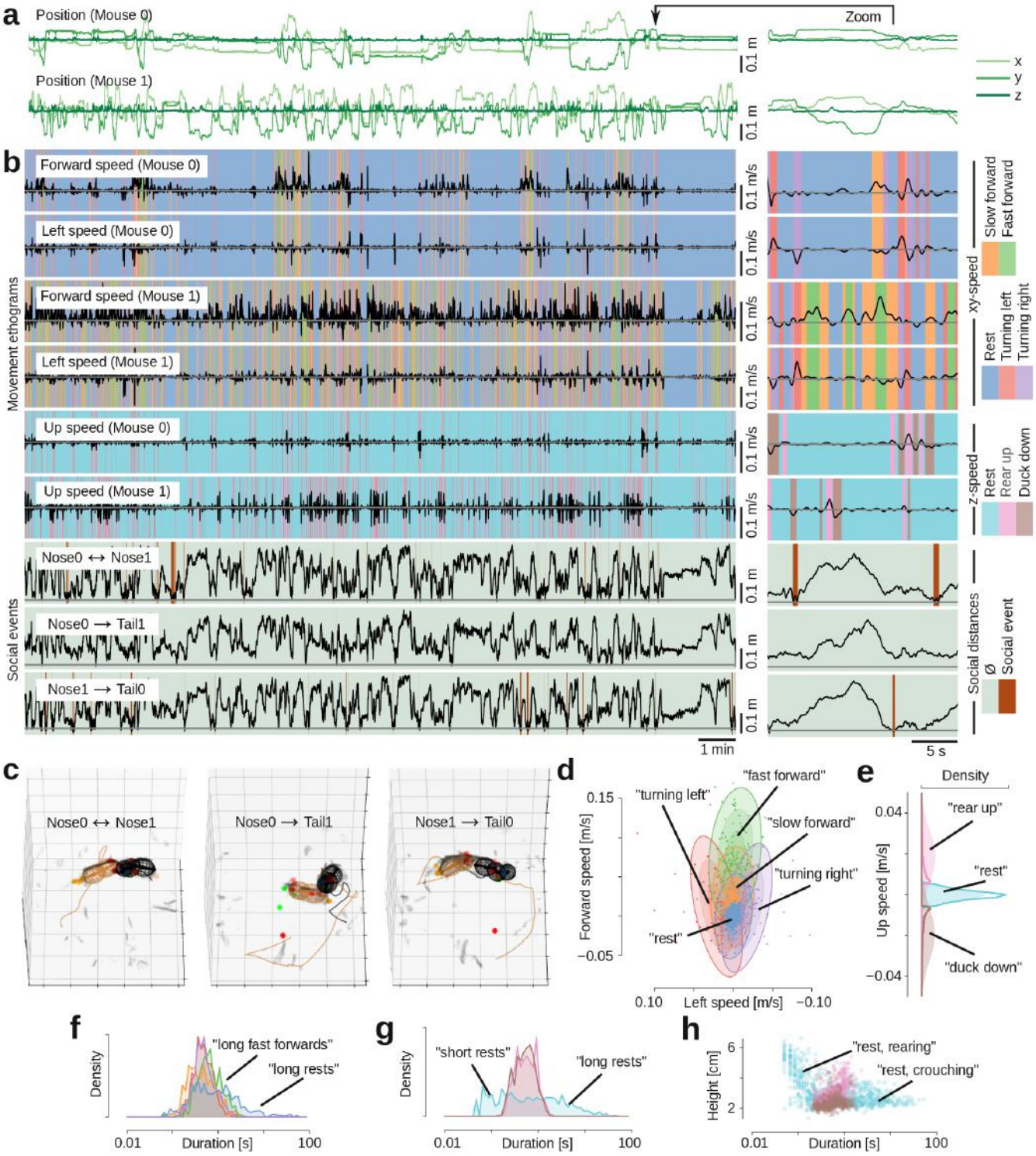


Figure 6. Automatic classification of movement patterns and behavioral states during social interactions. a, Tracked position of both mice, across an example 21 min recording. **b**, Extracted behavioral features: three speed components (forward, left and up in the mice's egocentric reference frames), and three

803 'social distances' (nose-to-nose distance and two nose-to-tail distances). Colors indicate ethograms of auto-
804 matically detected behavioral states. **c**, Examples of identified social events: nose-to-nose-touch, and ano-
805 genital nose-contacts. **e**, Mean and covariance (3 standard deviations indicated by ellipsoids) for each latent
806 state for the forward/leftward running (dots indicate a subsample of tracked speeds, colored by their most
807 likely latent state) **e**, Mean and variance of latent states in the z-plane (shaded color) as well as distribution
808 of tracked data assigned to those latent states (histograms) **f**, Distribution of the duration of the five behav-
809 ioral states in the *xy*-plane. Periods of rest (blue) are the longest ($p < 0.05$, Mann-Whitney U-test) and bouts
810 of fast forward movement (green) are to be longer other movement bouts ($p < 0.001$, Mann-Whitney U-
811 test). **g**, Distribution of duration of the three behavioral states in the z-plane. Periods of rest (light blue) are
812 either very short or very long. **h**, Plot of body elevation against behavior duration. Short periods of rest hap-
813 pen when the *z*-coordinate is high (the mouse rears up, waits for a brief moment before ducking back down),
814 whereas long periods of rest happen when the *z*-coordinate is low (when the mouse is resting or moving
815 around the arena, $\rho = -0.47$, $p < 0.001$, Spearman rank).

METHODS

Hardware

Necessary hardware:

Item	Recommendation	Price (USD)	N	Total (USD)
Depth cameras	Intel RealSense D435	179.00	4	716.00
Camera stands	Etubby 26” gooseneck webcam stand	24.96	4	99.84
PCIe card with 4 independent USB 3.0 controllers	Startech 4-port superspeed	83.54	1	83.54
Active, repeating USB 3.0 cables	UGREEN, USB 3.0 Active Repeater Cable	18.89	4	75.56
Arduino with USB cable	Arduino Uno R3	13.98	1	13.98
Pytorch-compatible GPU	Any NVIDIA card with CUDA support	500.00	1	500.00
Behavioral arena (acrylic cylinder or elevated platform)	12”-diameter, 5/32” thick acrylic cylinder	71.20	1	71.20
Depth camera GPIO pin connector (jumper)	JST ASSHSSH28K305	0.54	8	4.32
Depth camera GPIO pin connector (jumper housing)	JST SHR-09V-S	0.19	4	0.76
Colored ping-pong balls (for calibration)	Stiga 40 mm ITTF Regulation size	6.64	1	6.64
Total				1571.84

General lab electronics (tape, wire, soldering equipment, etc.) and:

Item	N
Infrared or red LEDs	4
0.1” pin headers or jumper wires	2
20 kOhm resistors	4
22 nF capacitors	4
200 Ohm resistor (or same order of magnitude)	1
Stick (for moving ping-pong ball during calibration)	1

827 ***Software***

828 Our system uses the following software: Linux (tested on Ubuntu 16.04 LTE, but should work on others,
829 https://ubuntu.com/), Intel Realsense SDK (https://github.com/IntelRealSense/librealsense), Python (tested
830 on Python 3.6, we recommend Anaconda, https://www.anaconda.com/distribution/). Required Python pack-
831 ages will be installed with PIP or conda (script in supplementary software). All required software is free and
832 open source.

833

834 ***Animal welfare***

835 All experimental procedures were performed according to animal welfare laws under the supervision of lo-
836 cal ethics committees. Animals were kept on a 12hr/12hr light cycle with ad libitum access to food and wa-
837 ter. Mice presented as partner animals were housed socially in same-sex cages, and post-surgery implanted
838 animals were housed in single animal cages. Neural recordings electrodes were implanted on the dorsal
839 skull under isoflurane anesthesia, with a 3D-printed electrode drive and a hand-built mesh housing. All pro-
840 cedures were approved under NYU School of Medicine IACUC protocols.

841

842 ***Recording data structure***

843 The Python program is set to pull raw images at 640 x 480 (color) and 640 x 480 (depth), but only saves 320
844 x 210 (color) and 320 x 240 (depth). We do this to reduce noise (multi-pixel averaging), save disk space and
845 reduce processing time. Our software also works for saving images up to 848 x 480 (color) and 848 x 480
846 (depth) at 60 frames/s, in case the system is to be used for a bigger arena, or to detect smaller body parts (eyes,
847 paws, etc). Images were transferred from the cameras with the python bindings for the Intel Realsense SDK
848 (https://github.com/IntelRealSense/librealsense), and saved as 8-bit, 3-channel PNG files with opencv (for
849 color images) or as 16-bit binary files (for depth images).

850

3D data structure

For efficient access and storage of the large datasets, we save all pre-processed data to hdf5 files. Because the number of data points (point-cloud and key-points) per frame varies, we save every frame as a jagged array. To this end, we pack all pre-processed data to a single array. If we detect N points in the point-cloud and M key-points in the color images, we save a stack of the 3D coordinates of the points in the point-cloud (Nx3, raveled to 3N), the weights (N), the 3D coordinates of the key-points (Mx3, raveled to 3M), their pseudo-posterior (M), an index indicating key-point type (M), and the number of key-points (1). Functions to pack and unpack the pre-processed data from a single line ('pack_to_jagged' and 'unpack_from_jagged') are provided.

Temporal synchronization

LED blinks were generated with voltage pulses from an Arduino (on digital pin 12), controlled over USB with a python interface for the Firmata protocol (https://github.com/tino/pyFirmata). To receive the Firmata messages, the Arduino was flashed with the 'StandardFirmata' example, that comes with the standard Arduino IDE. TTL pulses were 150 ms long and spaced by $\sim U(150,350)$ ms.. We recorded the emitted voltage pulses in both the infrared images (used to calculate the depth image) and on a TTL input on an Open Ephys Acquisition System (https://open-ephys.org/). We detected LED blinks and TTL flips by threshold crossing and roughly aligned the two signals by the first detected blink/flip. We first refined the alignment by cross correlation in 10 ms steps, and then identified pairs of blinks/flips by detecting the closest blink, subject to a cutoff (zscore < 2, compared to all blink-to-flip time differences) to remove blinks missed by the camera (because an experimenter moved an arm in front of a camera to place a mouse in the arena, for example). The final shift and drift was estimated by a robust regression (Theil-Sen estimator) on the pairs of blinks/links.

Deep neural network

We used a stacked hourglass network[14] implemented in Pytorch[25] (https://github.com/pytorch/pytorch). The network architecture code is from the implementation in 'PyTorch-Pose' (https://github.com/bearpaw/pytorch-pose). The full network architecture is shown in **Supplementary Fig 1**. The Image augmentation during training was done with the 'imgaug' library (https://github.com/aleju/imgaug). Our augmentation pipeline in shown in Supplementary Fig. 3. The network was trained by RMSProp ($\alpha = 0.99$, $\varepsilon = 10^{-8}$) with an initial learning rate of 0.00025. During training, the learning rate was automatically reduced by a factor of 10 if the training loss decreased by less than 0.1% for five successive steps (using the built-in learning rate scheduler in Pytorch). After training, we used the final output map of the network for key-point detection, and used a maximum filter to detect key-point locations as local maxima in network output images with a posterior pseudo-probability of at least 0.5.

Image labeling and target maps

For training the network to recognize body parts, we need to generate labeled frames by manual annotation. For each frame, 1-5 body parts are labeled on the implanted animal and 1-4 body parts on the partner animal. This can be done with any annotation software; we used a modified version of the free 'DeepPoseKit-Annotator'[8] (https://github.com/jgraving/DeepPoseKit-Annotator/) included in the supplementary code. This software allows easy labeling of the necessary points, and pre-packages training data for use in our training pipeline. Body parts are indexed by i/p for implanted/partner animal ('nose_p' is the nose of the partner animal, for example). Target maps were generated by adding a Gaussian function ($\sigma = 3$ px for implant, $\sigma = 1$ px for other body parts, scaled to peak value = 1) to an array of zeros (at 1/4th the resolution of the input color image) at the location of every labeled body key-point. 1D part affinity maps were created by connecting labeled key-points in an array of zeros with a 1 px wide line (clipped to max value = 1), and blurring the resulting image with a Gaussian filter ($\sigma = 3$ px).

Aligning depth and color data

The camera intrinsics (focal lengths, f, optical centers, p, depth scale, d_{scale}) and extrinsics (rotation matrices, R, translation vectors, $\bar{t}$) for both the color and depth sensors can be accessed over the SDK. Depth and color images were aligned to each other using a pinhole camera model. For example, the z coordinate of a single depth pixel with indices (i_c, i_d) and 16-bit depth value, d_{ij}, is given by:

$$z_d = d_{ij} \cdot d_{scale}$$

And the x and y coordinates are given by:

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{matrix} (j_d - p_{x,d}) \cdot z_d / f_{x,d} \\ (i_d - p_{y,d}) \cdot z_d / f_{y,d} \end{matrix}$$

Using the extrinsics between the depth and color sensors, we can move the coordinate to the reference frame of the color sensor:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_c = \mathrm{R}_{d \to c} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_d + \bar{t}_{d \to c}$$

Using the focal length and optical center, we can project the pixel onto the color image:

$$\begin{bmatrix} i_c \\ j_c \end{bmatrix} = \begin{matrix} f_{y,c} \cdot y_c / z_c + p_{y,c} \\ f_{x,c} \cdot x_c / z_c + p_{x,c} \end{matrix}$$

For assigning color pixel values to depth pixels, we simply rounded the color pixel indices (i_c, i_d) to the nearest integer and cloned the value. More computationally intensive methods based on ray-tracking exist ('rs2_project_color_pixel_to_depth_pixel' in the Librealsense SDK, for example), but the simple pinhole camera approximation yielded good results (small jitter average out across multiple key-points) which allowed us to skip the substantial computational overhead of ray tracing for our data pre-processing.

3D calibration and alignment

To align the cameras in space, we first mounted a blue ping-pong ball on a stick and moved it around the behavioral arena while recording both color and depth video. For each camera, we used a combination of

motion filtering, color filtering, smoothing and thresholding to detect the location of the ping-pong ball in the color frame (details in code). We then aligned the color frames to depth frames and extracted the corresponding depth pixels, yielding a partial 3D surface of the ping-pong ball. By fitting a sphere to this partial surface, we could estimate the 3D coordinate of the center of the ping-pong ball (**Fig. 3a**). This procedure yielded a 3D trajectory of the ping-pong ball in the reference frame of each camera (**Fig. 3b**). We used a robust regression method (RANSAC routines to fit a sphere with a fixed radius of 40 mm, modified from routines in https://github.com/daavoo/pyntcloud), insensitive to errors in the calibration ball trajectory to estimate the transformation matrices needed to bring all trajectories into the same frame of reference (**Fig. 3c**).

Body model

We model each mouse at two prolate ellipsoids. The model is specified by the 3D coordinate of the center of the hip ellipsoid, $\bar{c}_{hip} = [x, y, z]$, and the major and minor axis of the ellipsoids are scaled by a coordinate, $s \in [0,1]$ that can morph the ellipsoid from long and narrow to short and fat:

$$a_{hip} = a_{hip,0} + a_{hip,\Delta} \cdot s$$

$$b_{hip} = b_{hip,0} + b_{hip,\Delta} \cdot (1 - s)$$

The 'neck' (the joint of rotation between the hip and nose ellipsoid) is sitting a distance, $d_{hip} = 0.75 \cdot a_{hip}$, along the central axis of the hip ellipsoid. In the frame of reference of the mouse body (taking $\bar{c}_{hip}$ as the origin, with the major axis of the hip ellipsoid along the x-axis), a unit vector pointing to of the nose ellipsoid, from the 'neck' to the center of the nose ellipsoid along it's major axis is:

$$\bar{e}_{nose} = \begin{bmatrix} \cos\theta \\ \sin\theta\cos\phi \\ \sin\theta\sin\phi \end{bmatrix}$$

In the frame of reference of the laboratory ('world coordinates'), we allow the hip ellipsoid to rotate around the z-axis ('left'/'right') and around the y-axis ('up'/'down', in the frame of reference of the mouse). We

define $\mathbf{R}(\alpha_x, \alpha_y, \alpha_z)$ as a 3D rotation matrix specifying the rotation by an angle α around the three axes, and $\mathbf{R}(\bar{v}_1, \bar{v}_2)$ as a 3D rotation matrix that rotates the vector $\bar{v}_1$ onto $\bar{v}_2$. The we can define:

$$\mathbf{R}_{hip} = \mathbf{R}(0, \beta, \gamma)$$

$$\mathbf{R}_{head} = \mathbf{R}(\bar{e}_x, \bar{e}_{nose})$$

, where $\bar{e}_x$ is a unit vector along the x-axis. In the frame of reference of the mouse body, the center of the nose ellipsoid is:

$$\bar{c}_{nose,mouse} = \mathbf{R}_{head} \begin{bmatrix} d_{nose} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} d_{hip} \\ 0 \\ 0 \end{bmatrix}$$

So, in world coordinates, the center is:

$$\bar{c}_{nose,world} = \mathbf{R}_{hip} c_{nose,mouse} + \bar{c}_{hip}$$

The center of the neural implant if offset from the center of the nose ellipsoid by a distance x_{impl} along the major axis of the nose ellipsoid, and a distance z_{impl} orthogonal to the major axis. We allow the implant to rotate around the nose ellipsoid by an angle, ψ. Thus, in the frame of reference of the mouse body, the center of the ellipsoid is:

$$\bar{c}_{impl,mouse} = \mathbf{R}_{head} \begin{bmatrix} s_{impl} \\ d_{impl} \cdot \cos\psi \\ d_{impl} \cdot \sin\psi \end{bmatrix} + \begin{bmatrix} d_{hip} \\ 0 \\ 0 \end{bmatrix}$$

And in world coordinates, same as the center of the nose ellipsoid:

$$\bar{c}_{impl,world} = R_{hip} c_{impl,mouse} + \bar{c}_{hip}$$

We calculated other skeleton points (tip of the nose ellipsoid, etc.) in a similar method. We can use the rotation matrices for the hip and the nose ellipsoids to calculate the characteristic ellipsoid matrices:

$$\mathbf{Q}_{hip} = \mathbf{R}_{hip} \begin{bmatrix} 1/a_{hip}^2 & 0 & 0 \\ 0 & 1/b_{hip}^2 & 0 \\ 0 & 0 & 1/b_{hip}^2 \end{bmatrix} \left(\mathbf{R}_{hip}\right)^T$$

$$\mathbf{Q}_{nose} = \mathbf{R}_{hip}\mathbf{R}_{head}\begin{bmatrix} 1/a_{nose}^2 & 0 & 0 \\ 0 & 1/b_{nose}^2 & 0 \\ 0 & 0 & 1/b_{nose}^2 \end{bmatrix}\left(\mathbf{R}_{hip}\mathbf{R}_{head}\right)^T$$

Calculating the shortest distance from a point to the surface of an 3D ellipsoid in 3 dimensions requires solving a computationally-expensive polynomial[19]. Doing this for each of the thousands of points in the point-cloud, multiplied by four body ellipsoids, multiplied by 200 particles pr. fitting step is not computationally tractable. Instead, we use the shortest distance to the surface, $\tilde{d}$, along a path that passes through the centroid (Supplementary Fig. 4a-b). This is a good approximation to d (especially when averaged over many points), and the calculation of $\tilde{d}$ can be implemented as pure vectorized linear algebra, which can be calculated very efficiently on GPU[20]. Specifically, to calculate the distance from any point $\bar{p}$ in the point-cloud, we just center the points on the center of an ellipsoid, and – for example – calculate:

$$\bar{p}' = \bar{p} - \bar{c}_{hip}$$

$$\tilde{d} = \left|1 - \|\bar{p}'\|_{Q_{hip}}^{-1}\right| \cdot \|\bar{p}'\| \quad \text{where} \quad \|\bar{p}'\|_{Q_{hip}} = \sqrt{\langle \bar{p}', \bar{p}' \rangle}_{Q_{hip}} = \sqrt{(\bar{p}')^{\mathrm{T}} Q_{hip}\, \bar{p}'}$$

In fitting the model, we used the following constants.

Loss function evaluation and tracking

Joint position of the two mice is represented as a particle in 17-dimensional space. For each data frame, we start with a proposal particle (leftmost green block, based on previous frames), from which we generate 200 particles by pseudo-random perturbation within a search space (next green block). For each proposal particle, we calculate three types of weighted loss contributions: loss associated with the distance from the point-cloud to the surface of the mouse body models (top path, green color), loss associated with body key-points (middle path, key-point colors as in and loss associated with overlap of the two mouse body models (bottom path, purple color). We broadcast the results in a way, which allows us to consider all 200x200 = 40.000 possible joint postures of the two mice. After calculation, we pick the top 200 joint postures with the lowest overall

loss, and anneal the search space, or – if converged – continue to the next frame. When we continue to a new frame, we add the fitted frame to a KRLS-T filter bank (online adaptive filter for prediction), which proposes the next position of the particle for the next frame, based on previous frame. All loss function calculations, and KRLS-T predictions as pure tensor algebra, that can be fully vectorized and executed on a GPU.

State space filtering of raw tracking data

After tracking, the coordinates of the skeleton points (c_{hip}, c_{nose}, etc.) were smoothed with a 3D kinematic Kalman filter tracking both the 3D position (p), velocity (v) and (constant) acceleration (a). For example, for the center of the hip coordinate:

$$\bar{x} = \left[p_x, v_x, a_x, p_y, v_y, a_y, p_z, v_z, a_z\right]$$

$$\bar{z} = \left[c_{hip,x}, c_{hip,y}, c_{hip,z}\right]$$

$$\mathbf{F} = \begin{bmatrix} \mathbf{F'} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{F'} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{F'} \end{bmatrix}, \text{where } \mathbf{F'} = \begin{bmatrix} 1 & dt & \frac{1}{2}dt^2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{P} = \mathbf{1}_{9\times 9} \cdot \sigma_{cov}^2$$

$$\mathbf{R} = \mathbf{I}_{3\times 3} \cdot \sigma_{measurement}^2$$

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q'} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q'} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q'} \end{bmatrix} \cdot \sigma_{process}^2$$

where $\mathbf{Q}$' is the Q matrix for a discrete constant white noise model $\mathbf{Q'} = \begin{bmatrix} \frac{1}{4}dt^4 & \frac{1}{2}dt^3 & \frac{1}{2}dt^2 \\ \frac{1}{2}dt^3 & dt^2 & dt \\ \frac{1}{2}dt^2 & dt & 1 \end{bmatrix}$ and $\sigma_{measurement} = 0.015$ m, $\sigma_{process} = 0.01$ m, $\sigma_{cov}^2 = 0.0011$ m^2. The σ's were the same for all points, except the slightly more noisy estimate of the center of the implant, where we used. $\sigma_{measurement} =$

0.02 m, $\sigma_{process} = 0.01$ m, $\sigma_{cov}^2 = 0.0011$ m^2 From the frame rate (60 fps), $dt = \frac{1}{60}$ s. The maximum-likelihood trajectory was estimated with the Rauch-Tung-Striebel method[30] with a fixed lag of 16 frames. The filter and smoother was implemented using the 'filterpy' package (https://github.com/rlabbe/filterpy). The spine scaling, *s*, was smoothed with a similar filter in 1D, except that we did not model acceleration, only *s* and a (constant) *s* 'velocity', with $\sigma_{measurement} = 0.3, \sigma_{process} = 0.05$ m, $\sigma_{cov}^2 = 0.0011$.

After filtering the trajectories of the skeleton points, we recalculated the 3D rotation matrices of the hip and head ellipsoid by the vectors pointing from c_{hip} to c_{mid} (from the middle of the hip ellipsoid to the neck joint), and from c_{hip} to c_{nose} (from the neck joint to the middle of the nose ellipsoid). We then converted the 3D rotation matrixes to unit quaternions, and smoothed the 3D rotations by smoothing the quaternions with an 10-frame boxcar filter, essentially averaging the quaternions by finding the largest eigenvalue of a matrix composed of the quaternions within the boxcar [32]. After smoothing the ellipsoid rotations, we re-calculated the coordinates of the tip of the nose ellipsoid (c_{tip}) and the posterior end of the hip ellipsoid (c_{tail}) from the smoothed central coordinates, rotations, and – for c_{tail} – the smoothed spine scaling. A walkthrough of the state space filtering pipeline is shown in **Supplementary Fig. 6**.

Template matching

To detect social events, we calculated three social distances, from three instantaneous 'social distances', defined as the 3D distance between the tip of each animal's noses ('nose-to-nose'), and from the tip of each animal's nose to the posterior end of the conspecific's hip ellipsoid ('nose-to-tail'; Fig. 6c). From these social distances, we could automatically detect when the mouse bodies were in a nose-to-nose (if the nose-to-nose distance was < 2 cm and the nose-to-tail distance was > 6 cm) and in a nose-to-tail configuration (if the nose-to-nose distance was > 6 cm and the nose-to-tail distance was > 2 cm). The events were detected by the logic conditions, and then single threshold crossings due to noise were removed by binary opening with a 3-frame kernel, followed by binary closing with a 30-frame kernel.

State space modeling of mouse behavior

State space modeling of the locomotion behavior was performed in Pyro[35] a GPU-accelerated probabilistic programming language built on top of Pytorch[25]. We modeled the (centered and whitened) locomotion behavior as a hidden Markov model with discrete latent states, z, and and associated transition matrix, $\mathbf{T}$.

$$z(t+1) = \text{Categorical}(e_{z(t)}^{T} \cdot \mathbf{T})$$

$$\mathbf{T} = \begin{bmatrix} p_{ij} & \cdots \\ \vdots & \ddots \end{bmatrix}$$

To make the model 'sticky' (discourage fast switching between latent states) we draw the transition probabilities, p_{ij} from a Dirichlet prior with a high mass near the 'edges' and initialize $\mathbf{T}_{init} = (1-\eta)\mathbf{I} + \eta/n_{\text{states}}$ where $\eta = 0.05$.

$$p \sim \text{Diriclet}(0.5)$$

Each state emits a forward speed and a left speed, drawn from a two-dimensional gaussian distribution with a full covariance matrix.

$$\begin{bmatrix} v_{\text{fwd}} \\ v_{\text{left}} \end{bmatrix} \sim \text{MVNormal}(\mu, \mathbf{S})$$

We draw the mean of the states from a normal distribution and use a LKJ Cholesky prior for the covariance:

$$\mu \sim \text{Normal}(0,1)$$

$$\mathbf{S} = \begin{bmatrix} \sigma_{\text{fwd}} & 0 \\ 0 & \sigma_{\text{left}} \end{bmatrix} \mathbf{L} \begin{bmatrix} \sigma_{\text{fwd}} & 0 \\ 0 & \sigma_{\text{left}} \end{bmatrix}$$

$$\sigma \sim \text{LogNormal}(-1,1)$$

$$\mathbf{L} \sim \text{LKJcorr}(2)$$

The up speed was modeled in a similar way, except that the latent states were just a one-dimensional normal distribution. The means and variances for the latent states was initialized by kmeans clustering of the locomotion speeds. The model was fit in parallel to 600-frame snippets of a subset of the data by stochastic variational inference[62]. We used an automatic delta guide function ('AutoDelta') and an evidence lower bound

(ELBO) loss function. The model was fitted by stochastic gradient descent with a learning rate of 0.0005. After model fitting, we generated the ethograms by assigning latent states by maximum a posteriori probability with a Viterbi algorithm.

3D head direction estimation

We use the 3D position of the ear key-points to determine the 3d head direction of the partner animal. We assign the ear key-points to a mouse body model by calculating the distance from each key-point to the center of the nose ellipsoid of both animals (cutoff: closest to one mouse and < 3cm from the center of the head ellipsoid, Supplementary Fig 10a). To estimate the 3D head direction, we calculate the unit rejection (v_{rej}) between a unit vector along the nose ellipsoid (v_{nose}) and a unit vector from the neck joint (c_{mid}) to the average 3D position of the ear key-points that are associated with that mouse (v_ear_direction, Supplementary Fig. 10b). If no ear key-points were detected in a frame, we linearly interpolate the average 3D position. To average out jitter, the estimates of the average ear coordinates and the center of the nose coordinate were smoothed with a Gaussian ($\sigma = 3$ frames). The final head direction vector was also smoothed with a Gaussian ($\sigma = 10$ frames).

Supplementary material to:

Automatic tracking of mouse social posture dynamics by 3D videography, deep learning and GPU-accelerated robust optimization

Christian L. Ebbesen[1,2,*] & Robert C. Froemke[1,2,3,*]

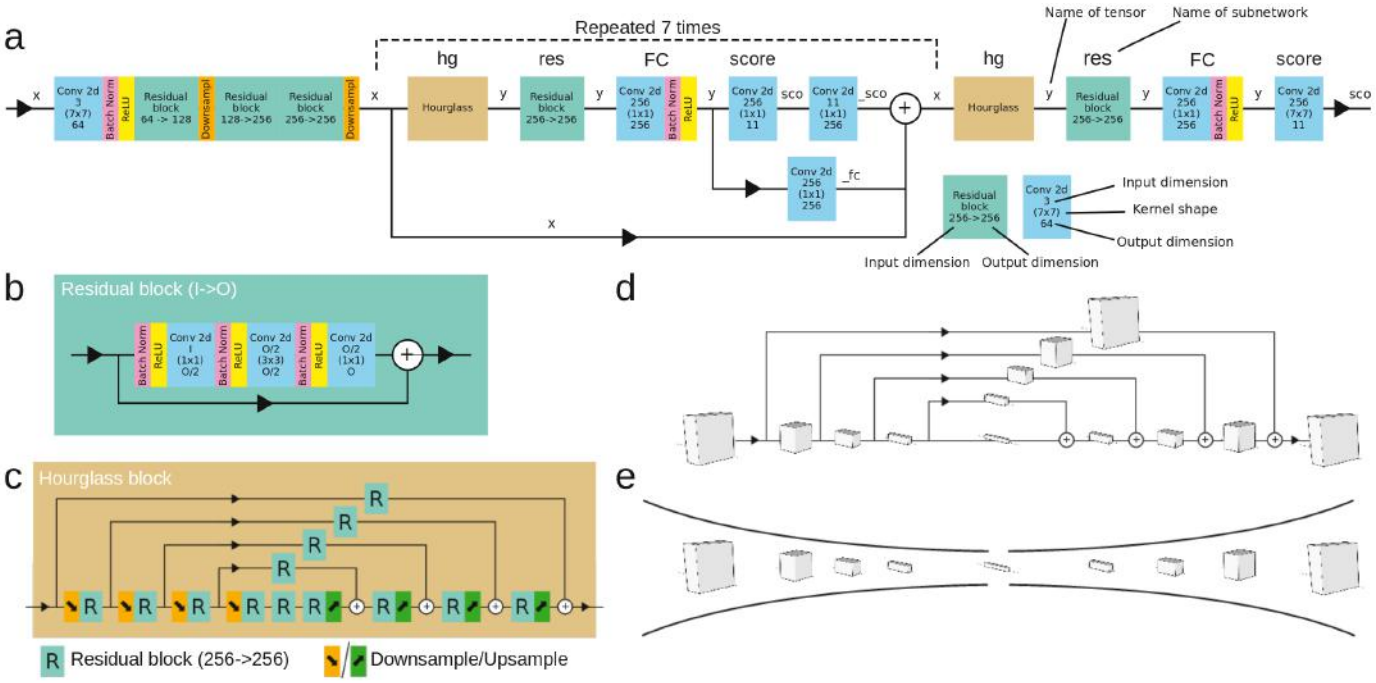
[1] Skirball Institute of Biomolecular Medicine, Neuroscience Institute, Departments of Otolaryngology, Neuroscience and Physiology, New York University School of Medicine, New York, NY, 10016, USA.

[2] Center for Neural Science, New York University, New York, NY, 10003, USA.

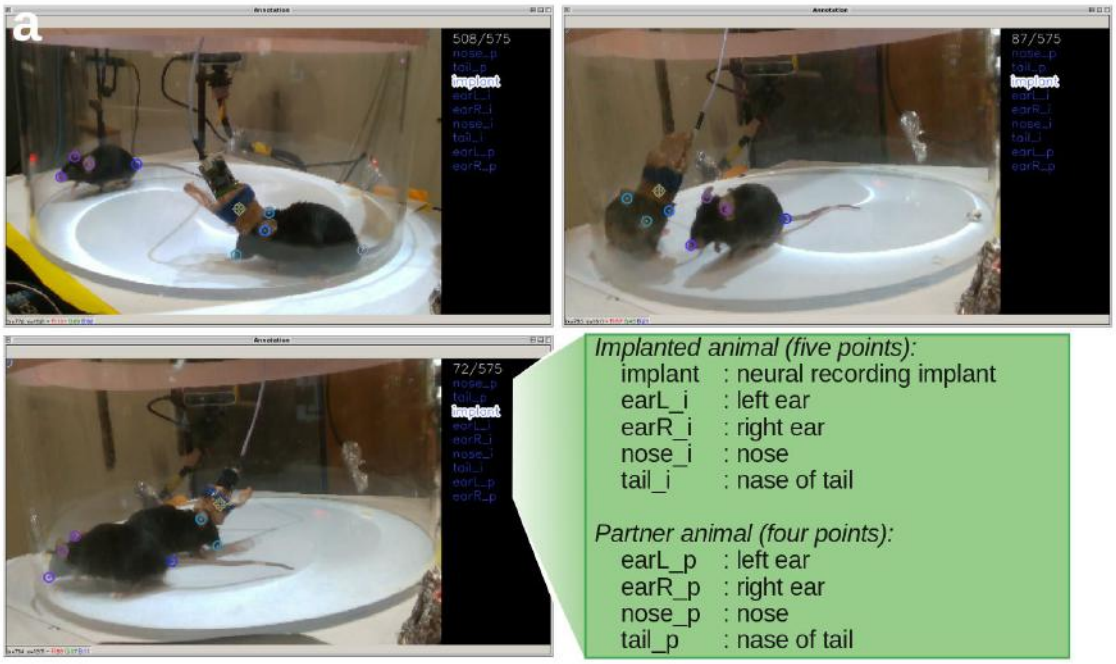
[3] Howard Hughes Medical Institute Faculty Scholar.

* Correspondence to: C.L.E. (christian.ebbesen@nyumc.org) or R.C.F. (robert.froemke@med.nyu.edu)

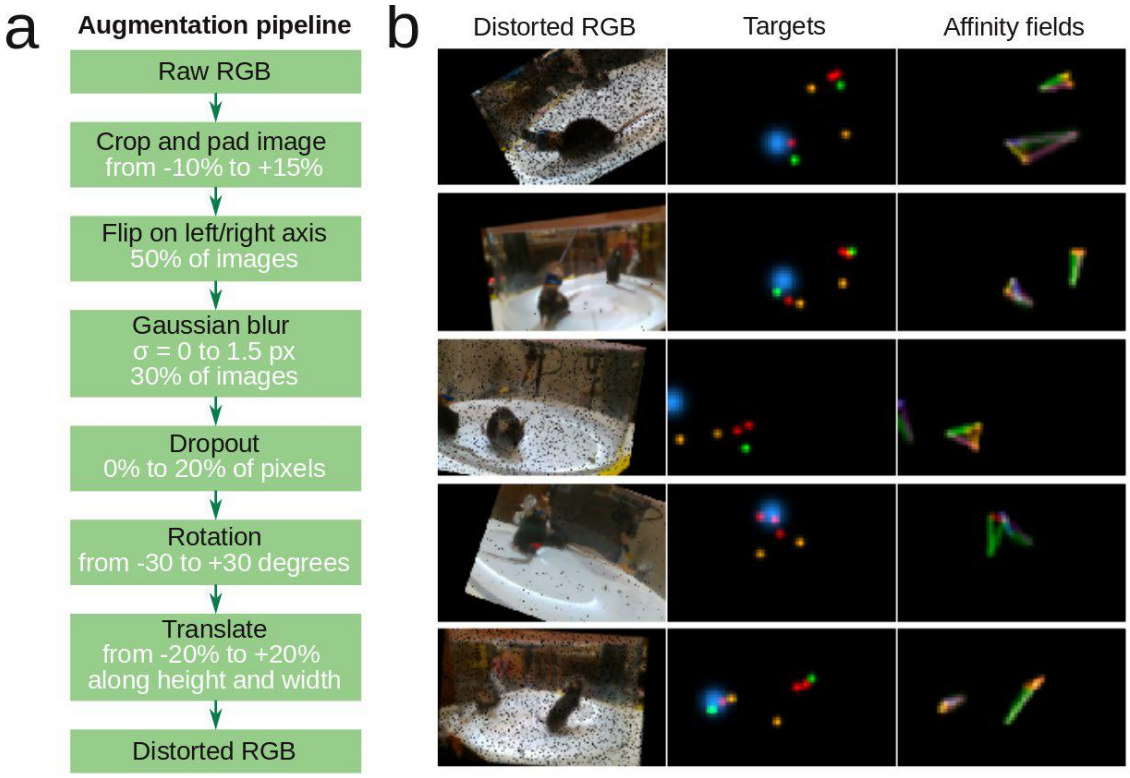
SUPPLEMENTARY FIGURES



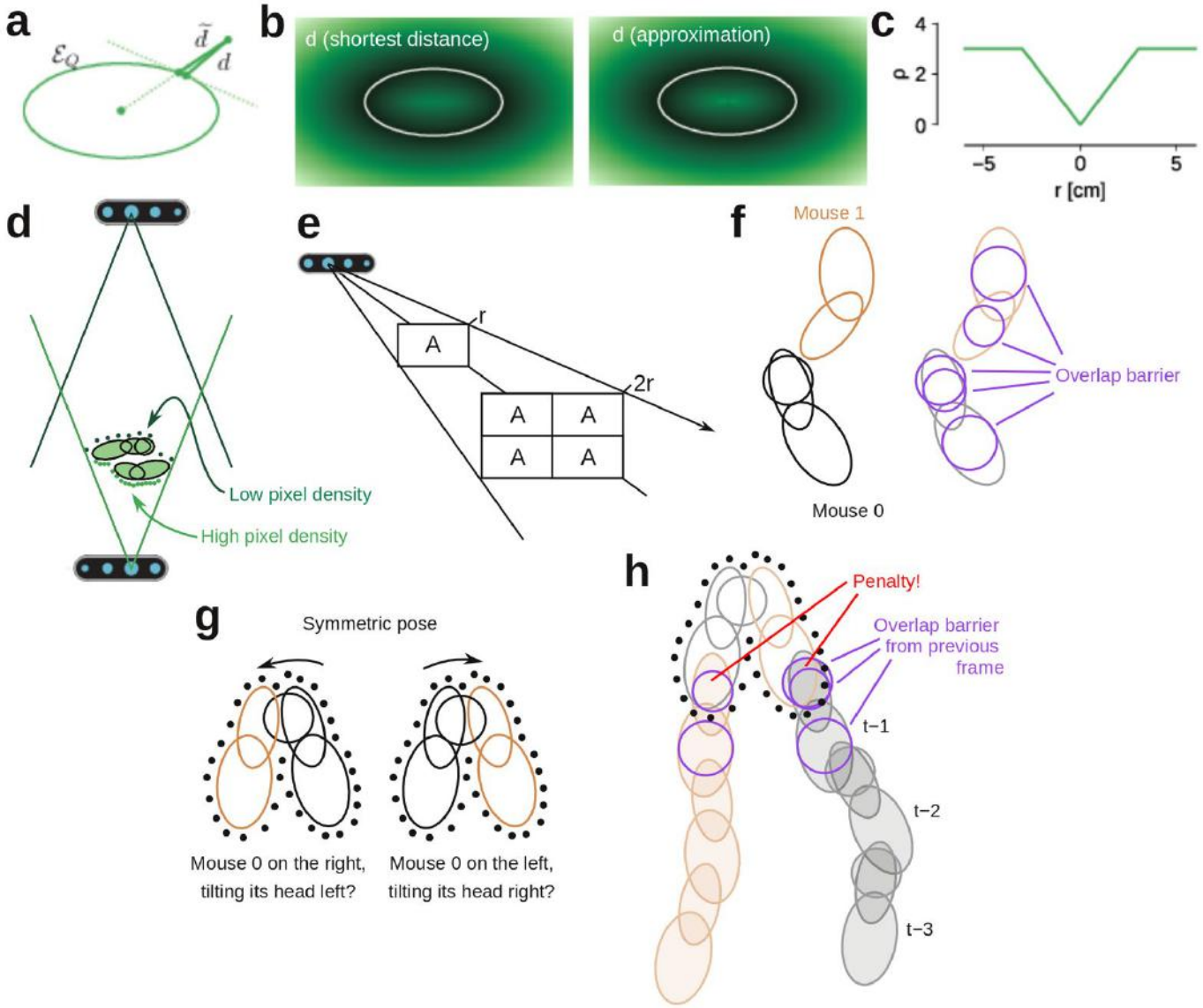
Supplementary Figure 1. Deep convolutional neural network architecture. a, Schematic of flow of tensors through deep convolutional neural network. Convolutional blocks show kernel shapes and input/output dimensions of feature dimension, starting from 3 (RGB image), expanding to 256 during hourglass blocks, and ending in 11 for intermediate and final outputs (4 body point targets, 7 part affinity fields). Full implementation details (e.g., including stride, padding, bias, etc.) are included in supplementary code. **b**, Schematic of a single residual block. **c**, Schematic of a single hourglass block. Upsampling (green, nearest neighbor) and downsampling (orange, by max pooling, both a factor of 2) happens along 2D image space (height/width). **d**, Shapes of tensors flowing through hourglass block. Along bottom path, feature dimension stays constant, but image dimensions (height/width) are increasingly downsampled, and then upsampled again. After each upsample, tensors are merged with skip connections (paths above). **e**, The hourglass-like shape that gives name to the network architecture.



Supplementary Figure 2. GUI for labeling of training data for the neural network. a, For training the network to recognize body parts, we must generate labeled frames by manual annotation. For each frame, 1-5 body parts are labeled on the implanted animal and 1-4 body parts on the partner animal. This can be done with any annotation software; we used a modified version of the free 'DeepPoseKit-Annotator' (Graving et al., 2019) (https://github.com/jgraving/DeepPoseKit-Annotator/) included in the supplementary code. This software allows easy labeling of the necessary points, and pre-packages training data for use in our training pipeline. Body parts are indexed by i/p for implanted/partner animal ('nose_p' is the nose of the partner animal, for example).

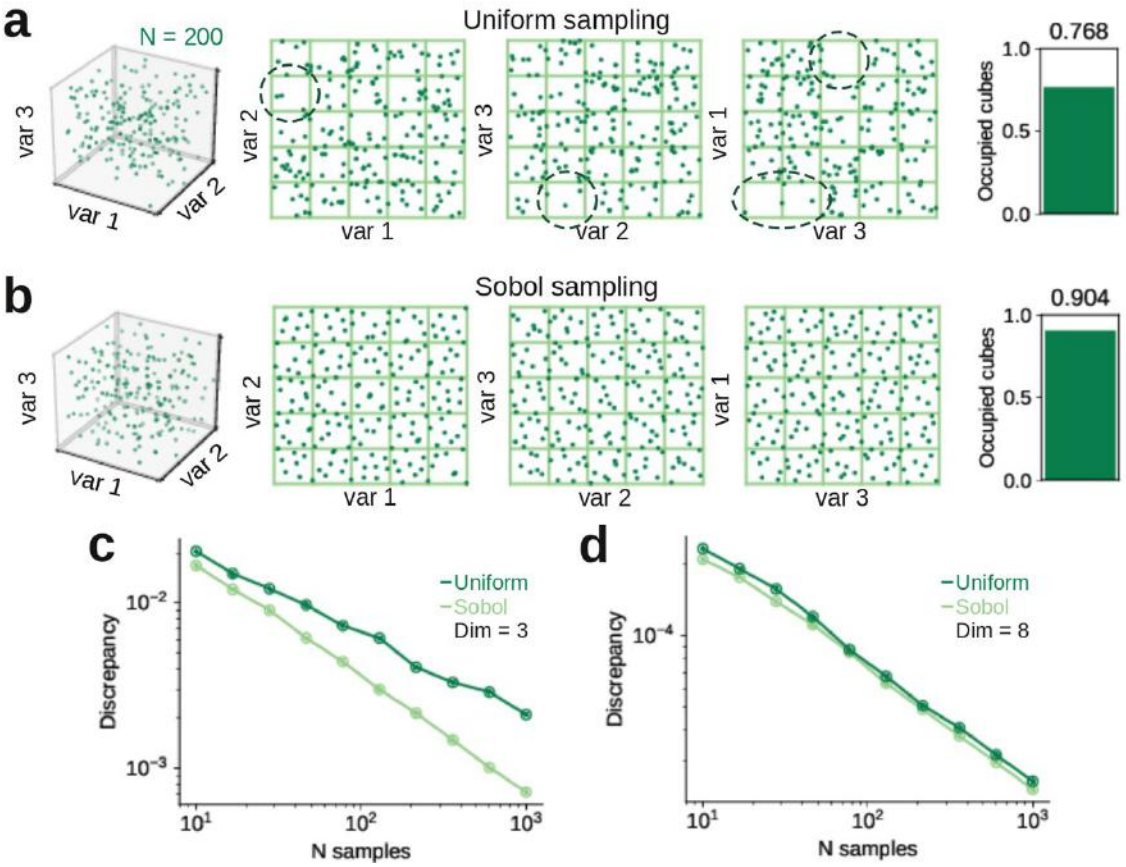


Supplementary Figure 3. Augmentation pipeline for network training. a, Flowchart of augmentation pipeline used to generate distorted frames during network training. **b**, Examples of distorted labeling frames generated by augmentation pipeline, as well as corresponding body part targets and affinity fields used during training.

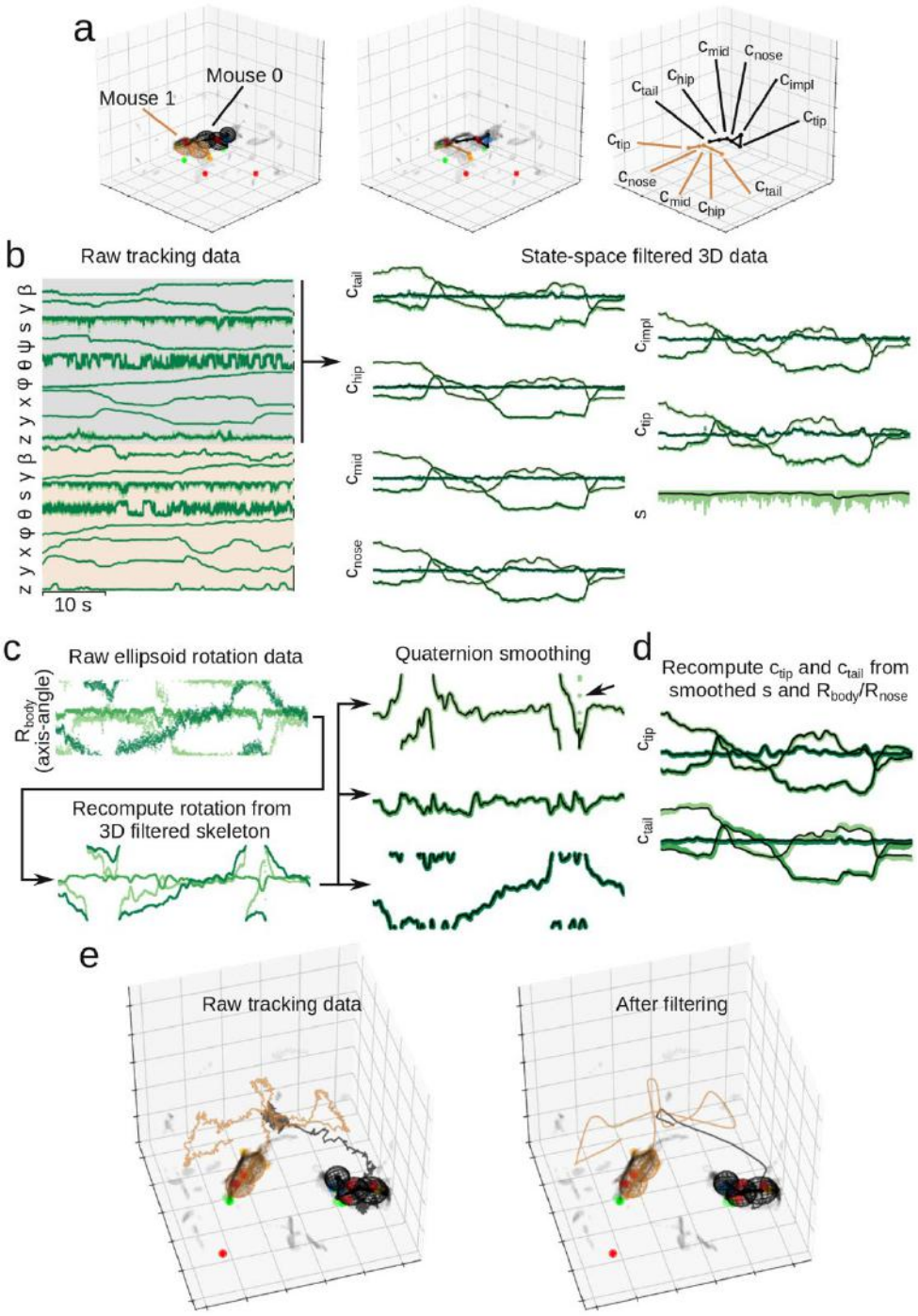


Supplementary Figure 4. Loss function calculation details. a, Shortest distance to surface of an ellipsoid, d, and our approximation, $\tilde{d}$. **b**, $\tilde{d}$ is a good approximation to d. Color map, value of $d/\tilde{d}$. White line, ellipsoid surface. **c**, The loss function, ρ, associated with the pointcloud is the mean absolute error of the distance estimate, truncated at +/- 3.0 cm **d**, Pixel density of the point-cloud depends on distance to the fixed-resolution depth cameras. **e**, Pixel density is inversely proportional to the square of the distance to depth camera. **f**, Overlap barrier spheres (implant sphere and spheres centered on the body ellipsoids with a radius equal to the minor axis). **g**, Example of mirror symmetric body position (side-by-side, facing same direction), resulting in ambiguity in animal identity if only one frame is considered. **h**, To include the context of previous frames, we

add an overlap loss penalty (similar to **f**) between each mouse and the position of the interaction partner in the previous frame. In panel **g**, right, we would add a penalty term to the particle representing joint body pose. In contrast, in panel **g**, left, this penalty is zero as there is no overlap with the position of the conspecific in the previous frame.

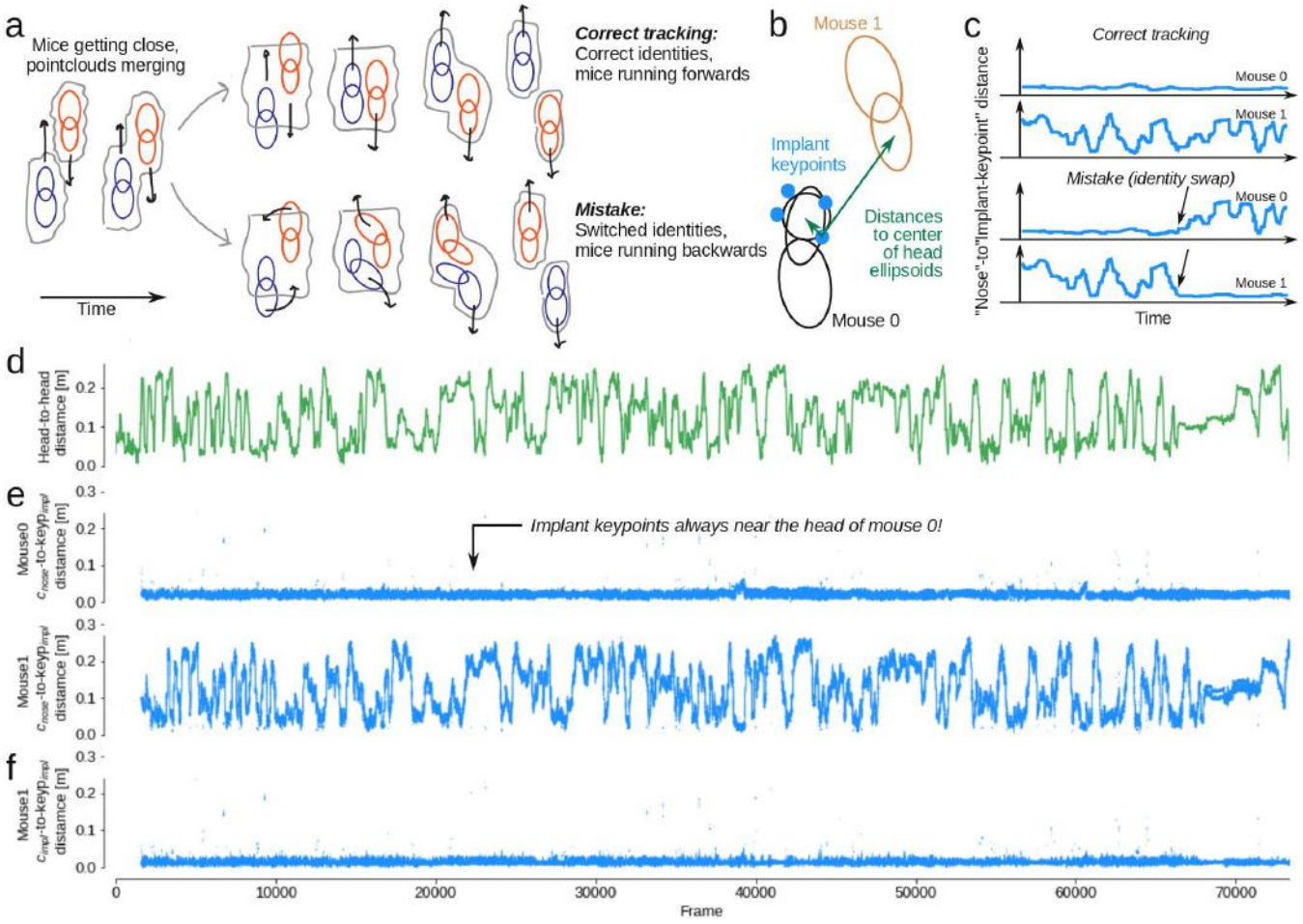


Supplementary Figure 5. Quasi-random particle filter exploration strategy. a, Left, 3D plot. Middle, 2D projection plots of three random variables, drawn from independent uniform distributions. Points in 3D space do not fill space well; in the 2D projections, there are squares (i.e., full rows, columns and pipes) of the 3D space not sampled at all (dashed lines). Right, partitioning space in 20%-cubes (green lines), only 76.8 % of cubes are occupied. **b**, Same as **a**, but variables are drawn from quasi-random Sobol sequence (Sobol, 1967). Points are more evenly dispersed in space, and 90.4% of all 20%-cubes are sampled. **c**, Mean discrepancy as function of sample number, for 3-dimensional (like panels **a,b**) uniform random sequence and a Sobol sequence, calculated across 100 random sequences. The Sobol sequences have a lower discrepancy, i.e. sample more regions of space. **d**, Same as **c**, but for 17-dimensional variables (like our joint body posture particles).



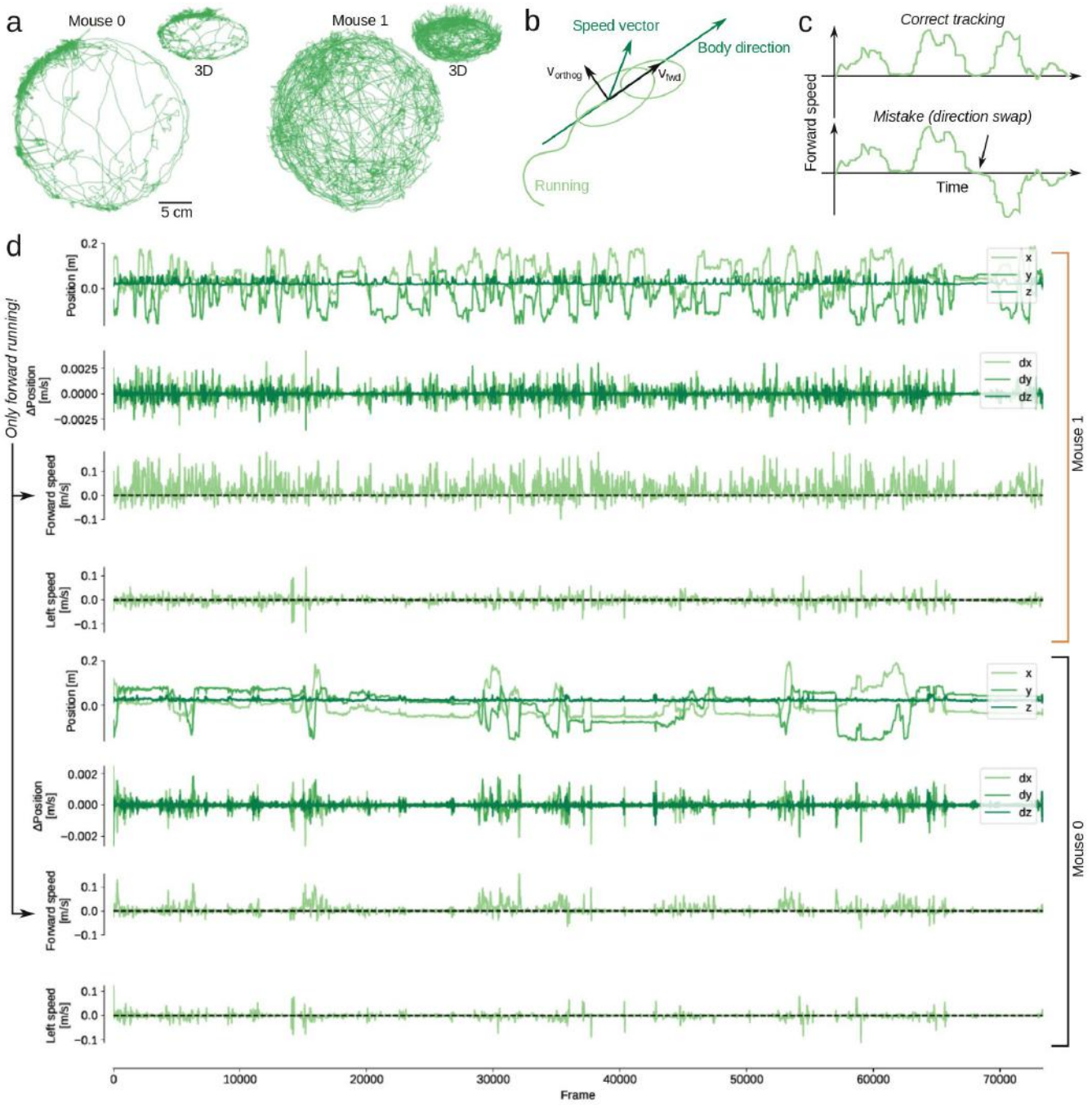
Supplementary Figure 6. State space filtering of tracked body models. a, Estimated 3D locations of body model surfaces (wireframes, left) and skeletons (dots and lines, right) for an example frame. **b**, Fitted joint pose parameters for the two mouse body models (left, 100 s snippet) and corresponding 3D coordinates of the body skeleton points, and the spine scaling, *s*, for the implanted mouse (right, same 100 s). **c**, Raw 3D rotation angle of the nose ellipsoid of the implanted animal (axis-angle representation), recalculated 3D rotation angles

from the filtered skeleton points, and final 3D rotation angles after quaternion smoothing (note the smoothing our of noise, indicated by arrow). **d**, Recalculated c_nose and c_tail from the smoothed 3D rotations and smoothed spine scaling. **e**, Example frame before (left) and after state space filtering of the tracked data (right).



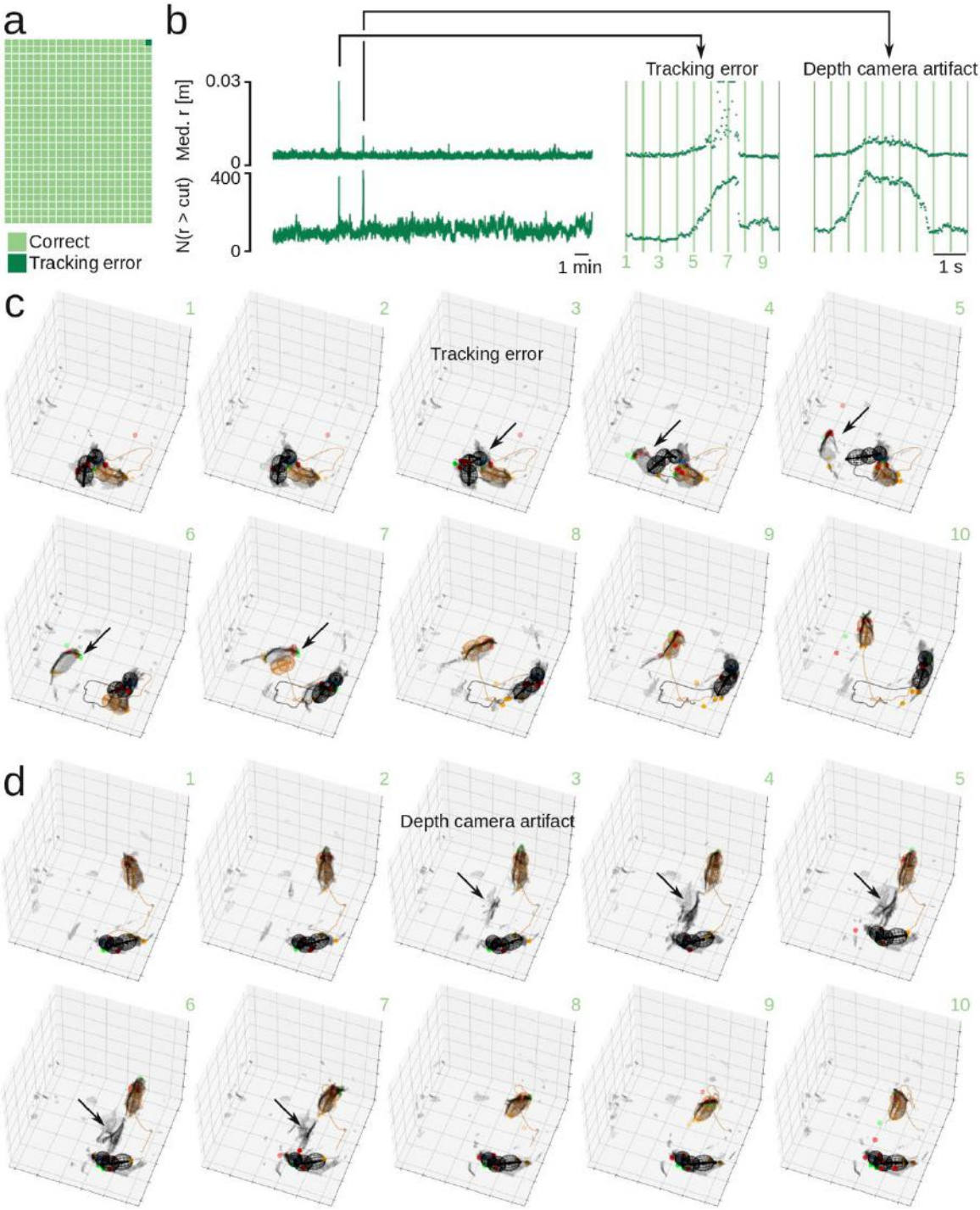
Supplementary Figure 7. Implant-to-nose distance demonstrates that there are no swapped identities. **a**, Schematic showing two common errors in tracking algorithms: Swapped identities and swapped directions. When the mice approach each other, their point clouds will merge. Because resolution and frame rates are limited, it can be hard to estimate body postures in this configuration. For example, if the tracking algorithm is not properly spatiotemporally regularized, the algorithm might mistakenly swap mouse identities, such that the mice appear to be running backwards (shown in bottom row). Direction swaps and identity swaps can also happen independently. For example, when mice are allogrooming, or passing over/under each other, identities might swap, but both mice can still appear to run normally with no apparent errors. Conversely, when a mouse is self-grooming, their point-cloud essentially resembles a ball, and when they start moving again, it may not be clear if they are 'really' moving forward or backwards. **b**, For all frames, we calculated the distance between implant key-points and the centroid of both nose ellipsoids. **c**, If there is an identity swap of the mice,

this is will be evident in the distance between the implant key-points and the head of both mice. In correct tracking (top row), implant body model always follows the same mouse. In tracking with mistakes (bottom row), implant will switch from being close to one mouse, to being close to the other mouse. **d**, The head-to-head (nose-centroid-to-nose-centroid) distance for the two mice, across the session. The mice often closely interact (low head-to-head distance), allowing for potential identity swaps. **e**, Distance between implant key-points and the nose centroid for both mice, across the session. The implant key-points are always near mouse0 and there are no identity swaps. **f**, The actual implant-key-point to implant-skeleton-point distance for mouse 0, across the session, is lower than the distance to the centroid of the nose ellipsoid.



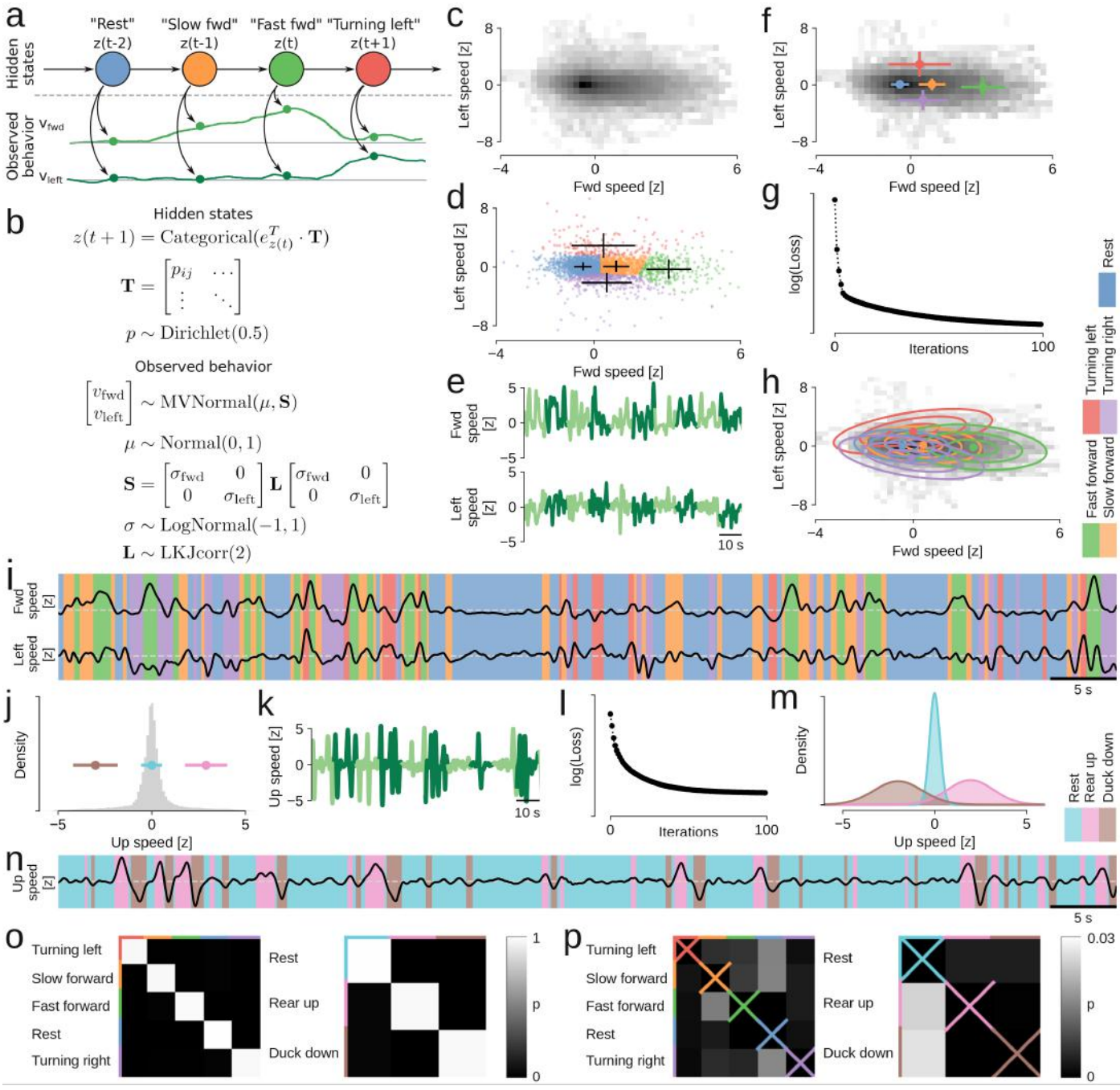
Supplementary Figure 8. Calculation of movement speed in egocentric coordinates. a, Running behavior of the two mice (centroid of the hip ellipsoid) across the behavioral session, shown in 2D (top-down view) and 3D. **b**, Running speed decomposed into two components, 'forward speed' (v_fwd, projected onto the orientation of the hip ellipsoid) and 'left speed' (v_orthog, the orthogonal component). **c**, In correct tracking (top row), running bouts will have positive forward speed. If there is a mistake in the tracking (bottom row), such that the mouse body model has switched direction, the mouse will appear to be 'running

backwards'. **d**, Top to bottom: The x,y,z-coordinates of the position (c_hip) of the mouse at each tracked frame, the change in position between frames, the forward speed, and the left speed. The four rows are repeated for both mice. There are no direction swap mistakes, and across the whole session, both mice only displayed bouts of forward running (confirmed by visual inspection of raw video).



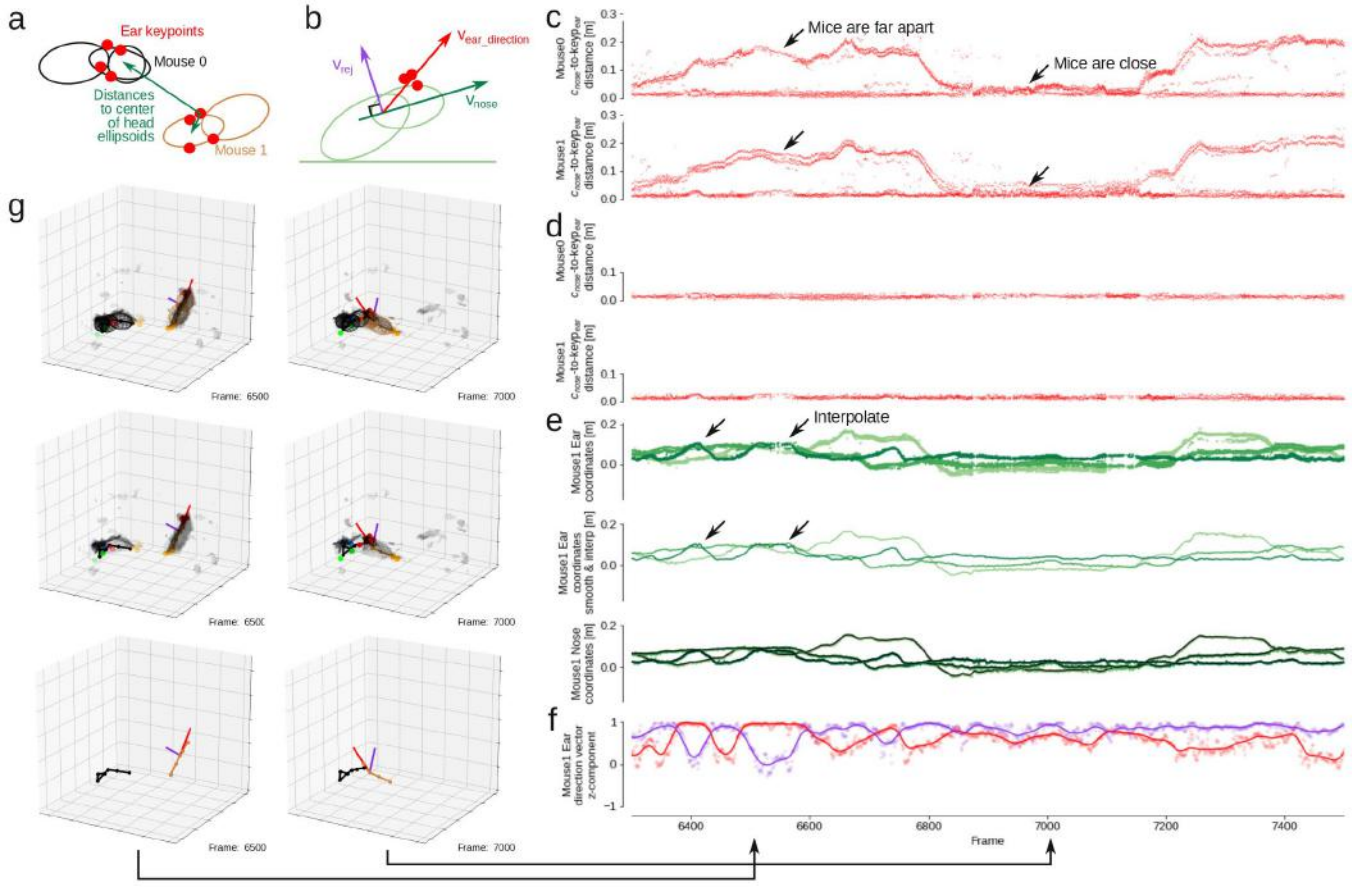
Supplementary Figure 9. Manual error checking. a, By manual inspection of 500 frames, we detected one tracking error. **b,** Median point-cloud residual (top) and number of point-cloud points with a residual larger than the cutoff (bottom, cut = 0.03 m) across an example 21 min recording. These traces show two anomalies: One tracking error (around frame 17000, the error we also detected by manual inspection of the 500 frames) and one depth camera artifact (tracking was fine, but a ghostly artifact showed up in the point-cloud for few

a seconds. Due to the of the robust loss function, tracking was not distorted by the artifact). **c,** Ten example frames showing the tracking error (0.5 s between frames, indicated by vertical lines in panel b). Note that after the error, the particle filter quickly recovers to correct tracking again. **d,** Ten example frames showing the depth camera artifact.



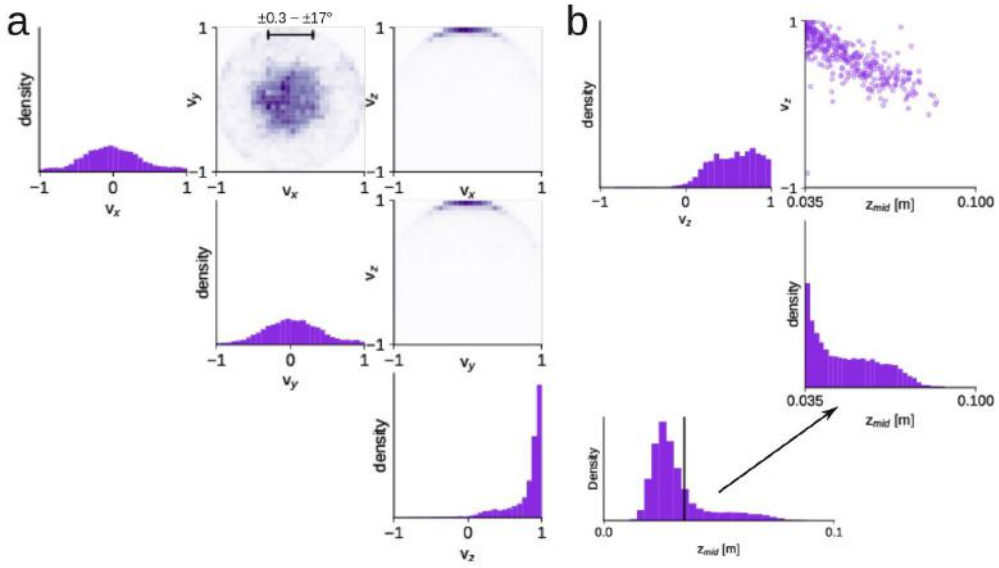
Supplementary Figure 10. Bayesian modeling and automatic classification of behavioral states. a, Generative model fit to the running behavior to automatically classify behavioral states. The model is a hidden Markov model with discrete latent states (circles), and each state emits a forward speed and a left speed, drawn from a two-dimensional gaussian distribution with a full covariance matrix. **b**, The generative model expressed as equations, showing Bayesian priors for estimating the parameters. **c**, Joint distribution (on a log-scale) of the forward speed and left speed, for both mice, across an entire behavioral session. **d**, Initial position

for the variation inference, for the model of forward and left speed. Crosses, cluster centers and standard deviations (calculated independently for fwd/left speed) for clusters assigned by k-means clustering into 5 clusters. Dots, individual samples of fwd/left speed (colors indicate clusters, every 50th sample is show). **e**, Bayesian model was fitted to a subset of the data (5 mins), split and run in parallel on 10-s sequences. The plot shows example 10-s sequences. **f**, Joint distribution (on a log-scale) of the forward speed and left speed, for the training data, overlayed with the cluster centers and standard deviations from all data (i.e., from **d**). Training data cover same velocity space as the whole session (compare with **a**). **g**, Convergence plot showing the decrease in model loss (increased evidence lower bound) across iterations for the training data. **h**, Locations and covariance ellipsoids (indicating three standard deviations) for the gaussian emission distributions associated with the five latent states, after model fitting. The five clusters are easily interpretable, and the labels are shown on the right. **j**, Initial position for the variation inference for the up speed. Distribution of the up speed (grey bars), as well as the center and standard deviation of three clusters (colored bars and dots), assigned by k-means clustering. **i**, Automatically assigned states (by maximum a posterior probability) to an example sequence of forward and left speed. **k,l,m,n**, same as **e,g,h,i**, but for the model fitting of the emission gaussians (in one dimension) of the up speed. **o**, Transition probabilities between latent states, for both forward/left speed and up speed models. The sample rate is 60 frames/s, so – since behavioral states are longer than that – the self-transition probabilities (diagonals) are very high. **p**, As **o**, but without showing the self-transition probabilities (the diagonals, crossed out). These matrices have understandable structure. For example, in the left matrix, the most likely transition from "rest" is to "slow forward". From "slow forward", the mouse is likely to transition to "turning left", "fast forward" or "turning right". It is very unlikely to transition directly from "fast forward" to "rest" or from "turning left" to "turning right". From the right matrix, we can see that it is unlikely to transition directly from "rear up" to "rear down", it is more probable to have a period of "rest" in between.



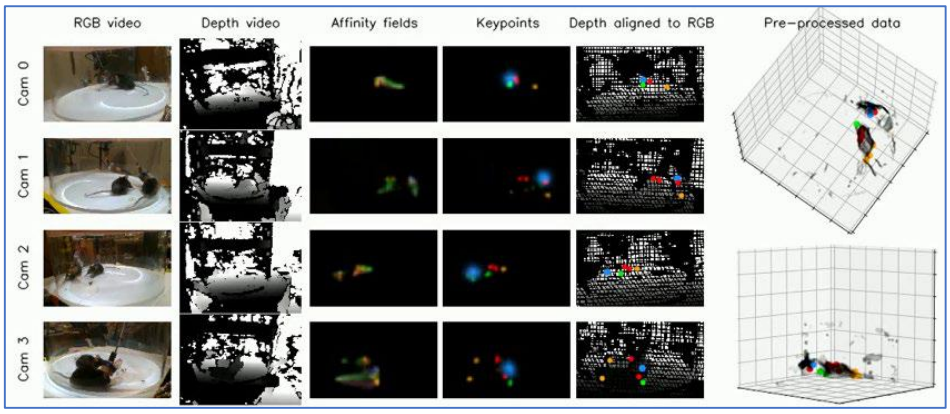
Supplementary Figure 11. Estimation of 3D heading direction in the partner animal, part I. a, We use the 3D position of the ear keypoints to determine the 3d head direction of the partner animal. We assign the ear keypoints to a mouse body model by caculating the distance from each keypoint to the center of the nose ellipsoid of both animals. **b**, To estimate the 3D head direction, we calculate the unit rejection (v_rej) between a unit vector along the nose ellipsoid (v_nose) and a unit vector from the neck joint (c_mid) to the average 3D position of the ear keypoints that are associated with that mouse (v_ear_direction). **c**, The distance from all ear keypoints to the center of the nose ellpsoid, for both mice, for an example portion of the recording session. **d**, The distance from ear keypoints to the center of the nose ellpsoid, only showing the keypoints that we estimate to be associated with each mouse. **e**, Estimated mean 3D position of the ear keypoints associated with the partner animal ('Mouse 1').Top to bottom: Raw 3D position of all keypoints, mean position using linear interpolation, smoothed with a Gaussian kernel (\sigma = XX frames). **f**, The z-component of v_ear_direction and v_rej. The z-component is high, indicating that the ears are on the dorsal side of the head ellipsoid.

When the mouse is running on the groung, both v_ear_direction and v_rej have high z-components (marked with rightmost arrow), but when the mouse is rearing and tilting the head backwards, v_rej will be more in the xy-plane, and have a low z-component (marked with leftmost arrow). **g**, The 3D body positions, of the frames indicated by arrows in panel **f**.

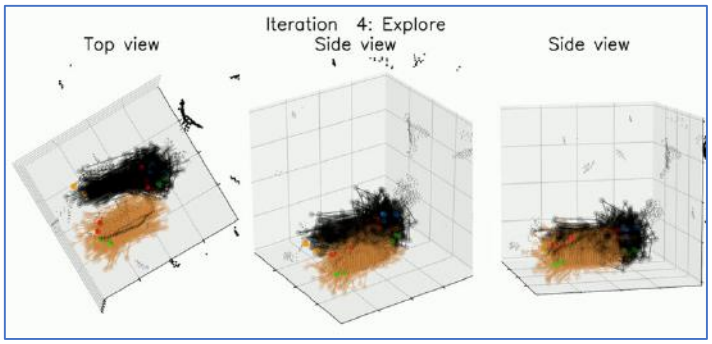


Supplementary Figure 12. Estimation of 3D heading direction in the partner animal, part II. a, The joint distributions of the components of v_rej (Supplementary Fig. 9b) shows that mouse mostly keeps the ears horizontal, rarely tilting the head more than 17 degrees towards the left or right. As also shown in Supplementary Fig 9f-g, the z-component is mostly close to 1 (pointing straight up), but sometimes smaller, closer to 0 (meaning that the nose is pointing up towards the sky). **b**, We can dig in to the details of the 3D head direction behavior. For example, we can decide to only look at the head direction, when the z-coordinate of the neck (z_mid) is high (i.e. when the mouse is rearing). Here we find a clear negative correlation between the z-component of v_rej and z_mid, which matches the visual inspection of the videos: When the mouse rears up or climbs up against the walls of the transparent social arena, the head tilts back to extend the nose upwards.

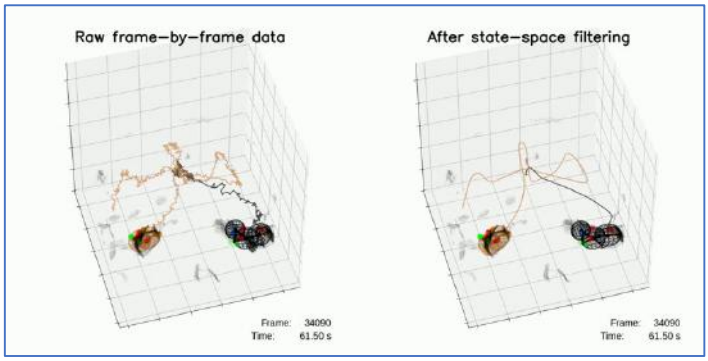
SUPPLEMENTARY VIDEOS



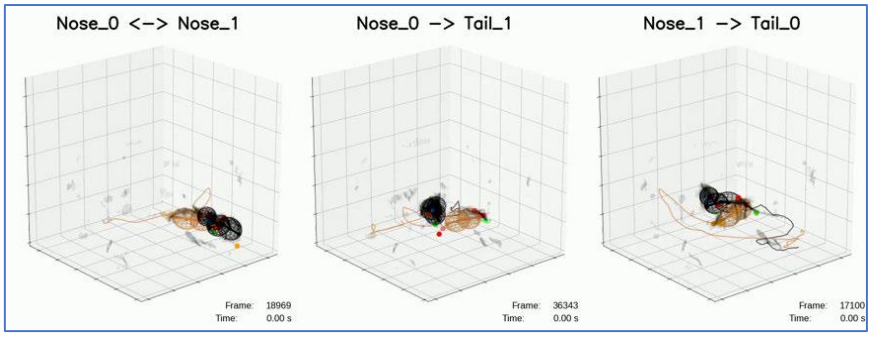
Supplementary Video 1. Pre-processing pipeline.



Supplementary Video 2. Particle filter behavior.



Supplementary Video 3. State-space filtering.



Supplementary Video 4. Social events.