Supplementary information

A cortex-wide self-consistent manifold for a body–environment reference frame

Fumiya Imamura[1], Hiroto Imamura[1], Reiko Hira[1], Yoshikazu Isomura[1,†], Riichiro Hira[1,2,†]

[1] *Department of Physiology and Cell Biology, Graduate School of Medical and Dental Sciences, Institute of Science Tokyo, Tokyo, Japan*
[2] *High Performance Artificial Intelligence System Research Team, Center for Computational Science, RIKEN, Saitama 351-0198, Japan*

† Corresponding authors.

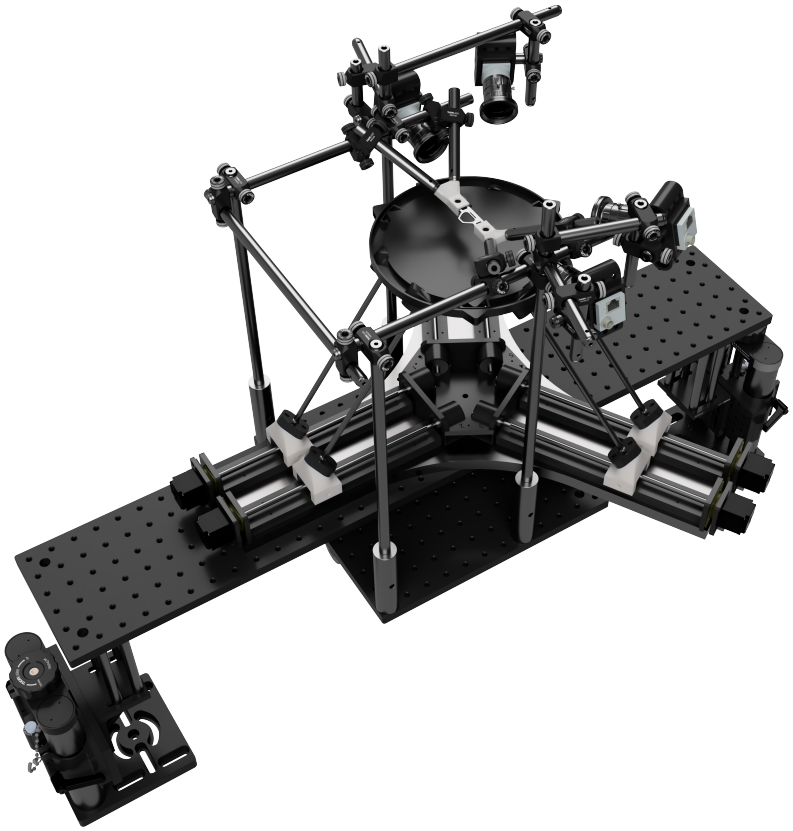
Correspondence:
Riichiro Hira, M.D., Ph.D. `rhira.phy2@tmd.ac.jp`
Department of Physiology and Cell Biology, Graduate School of Medical and Dental Sciences, Institute of Science Tokyo, 1-5-45 Yushima, Bunkyo-ku, Tokyo, 113-8510, Japan.
Yoshikazu Isomura, Ph.D. `isomura.phy2@tmd.ac.jp`
Department of Physiology and Cell Biology, Graduate School of Medical and Dental Sciences, Institute of Science Tokyo, 1-5-45 Yushima, Bunkyo-ku, Tokyo, 113-8510, Japan.

This supplementary document describes the design, implementation, and validation of the experimental and computational methods developed for this study. We constructed a custom six-degree-of-freedom motion platform (6AGM) capable of delivering precisely controlled whole-body tilt and translational stimuli to head-fixed mice while maintaining compatibility with large-scale neural recording techniques. The system integrates mechanical design, inverse kinematics, real-time control, encoder-based feedback, and modular software components implemented in LabVIEW and MATLAB, enabling accurate and reproducible generation of complex motion trajectories. Compared with commercially available platforms, the 6AGM provides an open-source and cost-effective solution optimized for neuroscience experiments requiring precise coordination between motion stimulation and neural measurement.
In addition, we developed a custom camera calibration pipeline tailored for high-precision three-dimensional reconstruction of mouse posture during motion stimulation. The calibration framework employs a radial distortion parameterization that improves geometric consistency across wide fields of view and supports accurate stereo reconstruction using checkerboard-based constraints. By refining intrinsic parameters, distortion coefficients, and stereo geometry through iterative model selection, the method enables reliable tracking of body-part positions in three dimensions. Together, the 6AGM platform and calibration pipeline provide an integrated experimental framework for quantitatively linking controlled whole-body motion to neural population dynamics across the dorsal cortex.

Contents

1. 6AGM

We developed a fully open-source six-degree-of-freedom arbitrary ground-motion platform (6AGM) for use in this study and with future integration into virtual-reality (VR) systems and brain–machine interfaces (BMIs) in mind. In the sections that follow, we provide a detailed account of the platform requirements and comparisons with existing devices, the mechanical design of 6AGM, the computational methods, and the hardware and software.



1.1. Platform development and comparison with existing 6-DOF platforms

We searched for a mechanism capable of rapidly moving a platform to arbitrary positions and orientations to enable an integrated understanding of movement, posture, and environmental cues using large-scale two-photon calcium imaging. For use with mice under the Diesel2p two-photon microscope, the static design constraints were an overall footprint $\leq$ 500 mm in diameter and $\leq$ 400 mm in height, with a platform disk 150–250 mm in diameter. The dynamic requirements were translational travel $\geq$ 60 mm in both mediolateral and anteroposterior directions, and rotational travel $\geq 10^\circ$ in roll, pitch, and yaw. Because the head-fixation apparatus, lick port, speakers, and visual-stimulus monitor together weigh $\sim$ 6 kg, the payload capacity needed to be $\geq$ 6 kg. While meeting these specifications, the system also had to be stable enough for two-photon imaging—specifically, under head fixation the brain's vertical (z-axis) motion had to be continuously suppressed to $\leq$ 2 μm—and we had to avoid imparting unwanted stimuli to the animal caused by jitter or chatter arising from feedback delays and related effects. We also required that arbitrary motion trajectories be programmable via custom scripts and that real-time motion be monitored with immediate response to real-time motion targets to enable future closed-loop control.

Supplementary Table 1 summarizes examples of commercially available 6-DOF platforms. While some products met individual requirements for size, payload, travel range, or speed, none satisfied all of them simultaneously. In particular, compact units had limited maximum speeds, whereas platforms that can move fast enough were too large to mount on an optical table. Several products offered precision far beyond our needs; however, devices that prioritized speed over precision were largely simulator rigs and were very large. Therefore, we built a platform that meets these requirements in-house.

Here, we developed a fully open-source system that satisfies the above specifications. Components can be custom-built, and we have released the corresponding CAD files. The hardware is based on low-cost, widely available controllers (Arduino, Arduino MEGA, and NI USB-6001), and the software centers on a LabVIEW-based control stack. All scripts are released as open source.

Supplementary Table 1: Comparison of commercially available 6-DOF platforms. The maximum velocity of platform motion, range of motion, size, mass, load capacity, and control software or interface are compared.

Product	**Maximum velocity**	**Range of motion**	**Precision (resolution / repeatability)**	**Size (height, base diameter)**	**Mass / load capacity**	**Interface / controller**
6AGMv1, F. Imamura et al.	290 mm/s (X,Y), 140 mm/s (Z), 90°/s (Rx), 105°/s (Ry), 157°/s (Rz)	165/165/115 mm / ±15°/ ± 10°/ ± 15°	Joint resolution 7.5 μm	Height 250 mm, ∅300 mm	6.7 kg / 6.5 kg*	Arduino MEGA + UNOs (encoders, pulse gen), TB6600 via NI USB-6001, MATLAB/LabVIEW
Symétrie SOLANO	30/20 mm/s (X,Y/Z), 15/20°/s (Rx,Ry/Rz)	±18/ ± 18/ ± 6.5 mm / ±10°/ ± 10°/ ± 21°	Resolution 0.08–0.1 μm, repeatability ±0.25 μm	Height 104 mm, ∅120 mm	1 kg / 5 kg	ALPHA+ controller, Ethernet
PI H-811 (Miniature)	20–25 mm/s (X,Y,Z), ∼ 28.6°/s	±17/ ± 16/ ± 6.5 mm / ±10°/ ± 10°/ ± 21°	MiM 0.08–0.2 μm / repeatability ±0.06–0.15 μm	Height 114 mm, ∅136 mm	2.2 kg / 5 kg	C-887 (EtherCAT / TCP-IP / RS-232)
Aerotech HEX150-125HL	30 mm/s (X,Y) / 8 mm/s (Z) / 10°/s (Rx,Ry), 30°/s (Rz)	42/44/17 mm / ±16° (Rx,Ry), ±42° (Rz)	MiM 15 nm / repeatability ±1.5 μm (X), ±0.4 μm (Z), ±3 arcsec	Height 125 mm, ∅150 mm	3 kg / 5 kg	HEX RC / Automation1
Newport / MKS HXP50	14/12/5 mm/s (X/Y/Z) / 6/6/15°/s (Rx/Ry/Rz)	±17/ ± 15/ ± 7 mm / ±9°/ ± 8.5°/ ± 18°	MiM 0.05–0.1 μm / repeatability 0.2 μm	Height 151 mm, ∅200 mm	2.2 kg / 5 kg	HXP50-ELEC controller (TCP/IP)
PI H-820	20 mm/s (X,Y,Z) / 11.5°/s	±50/±50/±25 mm / ±15°/ ± 15°/ ± 30°	MiM 5 μm / repeatability ±1.5 μm (X,Y), ±0.5 μm (Z), ±8/ ± 8/ ± 25 μrad	Height 308 mm, ∅350 mm	15 kg / 20 kg	C-887 6D controller (PI)

Product	**Maximum velocity**	**Range of motion**	**Precision (resolution / repeatability)**	**Size (height, base diameter)**	**Mass / load capacity**	**Interface / controller**
PI H-860	> 250 mm/s (X,Y,Z), 4 g, ∼ 25 Hz	±7.5 mm / ±4°	—	Height 319 mm, ∅407 mm	30 kg / 1 kg	PI 6D controller (TCP/IP, RS-232)
ACROME Stewart Platform	10 mm/s (12 kg) / 40 mm/s (35 kg) / 12–40°/s (roll/pitch)	±60–100 / ±50–103 mm / ±20–30°	Resolution 100–200 μm / repeatability ±50–100 μm	Height 406–717 mm, ∅450 mm	14 kg / 12 kg	TCP/UDP API (Ethernet, MATLAB/Simulink etc.)
Symétrie HEGOA	200 mm/s (X,Y) / 120 mm/s (Z) / 50°/s	±100/ ± 50 mm / ±23°/ ± 23°/ ± 30°	—	Height 420 mm, ∅500 mm	30 kg / 50 kg	Ethernet (ALPHA+)
Mikrolar P1500	1,244 mm/s (X,Y) / 254 mm/s (Z)	∅833 mm (XY) / 315 mm (Z) / ±25°/ ± 25°/ ± 15°	Repeatability 25 μm	Height 833 mm, ∅833 mm	36 kg / 150 kg	Software/UI (G-code)
Quanser Hexapod	0.67 m/s, 1 g, 0–10 Hz	±7.4/ ± 11/ ± 5.4 cm / ±17°/ ± 15°/ ± 27°	Joint resolution 0.1 μm	Height 0.75 m, ∅1.1 m	100 kg / 100 kg	DAQ/amplifier, USB, QUARC (MATLAB/Simulink)
DOF Reality H6 (HERO 6)	0.5 m/s / ∼ 86°/s	±17°	—	1.20 × 1.50 × H 0.60 m	90 kg	USB control unit (SimRacingStudio)

MiM: Minimum Incremental Motion.
* Load capacity depends on the stage height. The value shown corresponds to a stage height of 230 mm (the default configuration used in experiments). Increasing the stage height increases the load capacity.

1.2. Mechanical design and CAD components

Here, we describe the mechanical design of the 6AGM system. The main components of the 6AGM include the stage, on which the animal is placed, the actuators (NEMA11 2PH, Pilang), and the connecting shafts linking the stage to the actuators. The stage, together with the connectors between the shafts and actuators, was designed in 3D CAD software (Fusion 360, Autodesk) and fabricated with a 3D printer (Form 2, Formlabs). Ball joints were implemented using magnetic sockets and steel balls, which were mounted with custom parts produced by 3D printing. Unlike conventional Stewart platforms, where actuators are vertically arranged and must bear much of the load, our design places linear actuators on a flat surface. This geometry allows the weight of the stage and its payload to be distributed primarily through normal forces rather than the actuators themselves, improving load support and reducing mechanical stress. Below, we present the final assembly along with each individual component and its description.

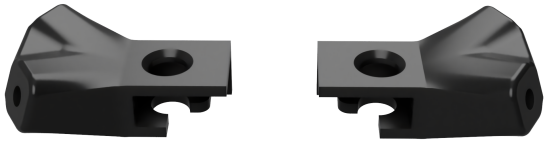
- **6dofPlatformAssembly.dxf**
  CAD model of the assembled 6AGM.

- **Stage.dxf**
  The stage, designed to accommodate head-fixed animals, is connected to six shafts via top joints. A circular window, 14 mm in diameter, provides optical access for imaging the animal's underside with cameras. To reduce surface reflections from the acrylic plate, a black infrared-absorbing sheet (IR Flock Sheet; The Black Market) was applied when necessary.

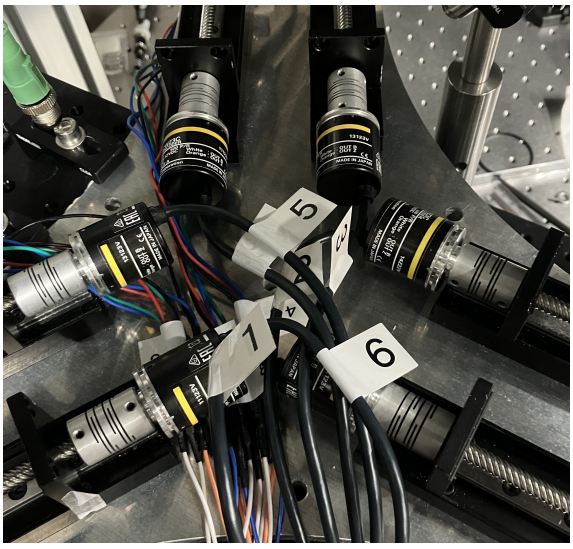
- **Base.dxf**
  A 10-mm-thick aluminum plate was used to mount the six stepper motors. For precise stage control, the relative positions of the motors had to remain rigidly fixed, thereby ensuring stable and reproducible kinematics of the 6AGM.

- **BallJointInterface.dxf**
  Ball joints connect each end of the shafts to the actuators and the stage. At the bottom, a magnetic socket allows the ball to roll smoothly, while the upper plate holds it in place without restricting motion. After 3D printing, the upper plate was trimmed to increase the range of motion. Oil (S4-T1500N, Sugiura Laboratory Inc.) was applied to all joints to minimize friction.

- **HeadPlateHolder.dxf**
  The head plate holder is fixed by rods attached at an angle to avoid interference with the platform below and the objective lens above.



- **Photograph of the rotary encoder**
  The inner ends of the linear guides are disassembled to connect rotary encoders (E6A2-CWZ3C) for tracking the rotations.



- **DamperAdapter.dxf**
  To prevent vibrations of the platform from propagating as severe artifacts during two-photon imaging, each stepper motor was mounted via a vibration-absorbing damper, attached with a custom adapter.

1.3. Inverse kinematics and actuator coordinate conversion

This section describes how a desired platform pose is converted into the commanded displacements of the six linear actuators. Specifically, given the platform orientation and translation, we solve the inverse kinematics of the mechanism to determine the actuator positions required to realize the prescribed stage motion.

Let the base coordinate system be fixed at the center of the aluminum base plate, and let the stage coordinate system be fixed at the center of the stage surface. The base-side ball-joint locations are denoted by $\mathbf{a}_i$, the stage-side ball-joint locations by $\mathbf{b}_i$, and the actuator guide directions by $\mathbf{v}_i$.

Because the mechanism has threefold rotational symmetry, these attachment points can be generated from reference coordinates using in-plane rotations about the z axis. Define

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

$$\theta_i = 0,\ 0,\ \frac{2\pi}{3},\ \frac{2\pi}{3},\ \frac{4\pi}{3},\ \frac{4\pi}{3}, \qquad i = 1, \ldots, 6, \tag{2}$$

and let

$$s_i = \begin{cases} -1, & i = 1, 3, 5, \\ 1, & i = 2, 4, 6. \end{cases} \tag{3}$$

Then the base-side attachment points are

$$\mathbf{a}_i = R_z(\theta_i) \begin{bmatrix} a_x \\ s_i a_y \\ a_z \end{bmatrix}, \qquad (a_x, a_y, a_z) = (197.2,\ 30.0,\ 47.7)\ \text{mm}, \tag{4}$$

and the actuator guide directions are

$$\mathbf{v}_i = \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \\ 0 \end{bmatrix}, \qquad i = 1, \ldots, 6. \tag{5}$$

Similarly, the stage-side attachment points are

$$\mathbf{b}_i = R_z(\theta_i) \begin{bmatrix} b_x \\ s_i b_y \\ b_z \end{bmatrix}, \qquad (b_x, b_y, b_z) = (80.5,\ 30.0,\ -25.5)\ \text{mm}. \tag{6}$$

For a given platform pose, the point $\mathbf{b}_i$ is mapped from the stage frame into the base frame as

$$R\mathbf{b}_i + \mathbf{p}, \tag{7}$$

where R is the platform rotation matrix and $\mathbf{p}$ is the platform translation vector.

The actuator carriage for leg i moves along the fixed base-frame direction $\mathbf{v}_i$, starting from the base point $\mathbf{a}_i$. Its position is therefore

$$\mathbf{a}_i + \lambda_i \mathbf{v}_i, \tag{8}$$

where λ_i denotes the actuator displacement. Because the connecting shaft has fixed length l, the distance between the transformed stage-side joint and the actuator carriage must satisfy

$$\|R\mathbf{b}_i + \mathbf{p} - \mathbf{a}_i - \lambda_i \mathbf{v}_i\|^2 = l^2. \tag{9}$$

Defining

$$\mathbf{c}_i = R\mathbf{b}_i + \mathbf{p} - \mathbf{a}_i, \tag{10}$$

the length constraint becomes

$$\|\mathbf{c}_i - \lambda_i \mathbf{v}_i\|^2 = l^2. \tag{11}$$

Since $\mathbf{v}_i$ is a unit vector, this expands to

$$\lambda_i^2 - 2\,\mathbf{c}_i^T \mathbf{v}_i\, \lambda_i + \mathbf{c}_i^T \mathbf{c}_i - l^2 = 0. \tag{12}$$

Solving this quadratic equation gives

$$\lambda_i = \mathbf{c}_i^T \mathbf{v}_i \pm \sqrt{\left(\mathbf{c}_i^T \mathbf{v}_i\right)^2 - \mathbf{c}_i^T \mathbf{c}_i + l^2}. \tag{13}$$

In practice, the physically admissible root consistent with the actuator mounting direction is selected; in our implementation this corresponds to the positive square-root branch.

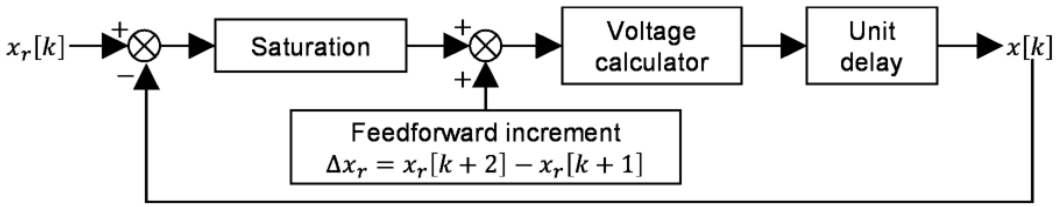
Applying this calculation to all six legs converts the desired platform pose $(R, \mathbf{p})$ into the six actuator displacements required by the motor controllers.

To reduce susceptibility to parallel singular configurations, the six linear actuators were arranged as three adjacent parallel pairs, each aligned with one side of the triangular base.

In preliminary open-loop operation, small actuator errors accumulated over time and led to positional drift. Therefore, a high-speed feedback control system was introduced to maintain positioning accuracy during long experimental sessions, as described in the next section.

1.4. Discrete-time feedback–feedforward control scheme

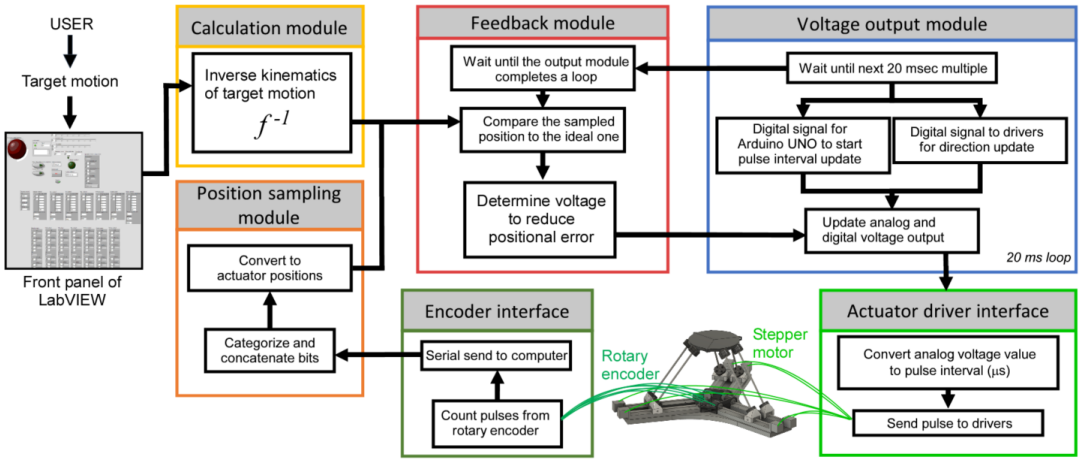
To ensure accurate trajectory tracking of the 6-DOF platform, we implemented a discrete-time feedback–feedforward control scheme (**Extended Data Fig. 1b**). At each control step, the current positions of the six stepper motors were read out and compared with the ideal actuator positions, yielding offset values. These offsets were clipped to a tunable value to limit excessive commands and to prevent mechanical instability or oscillations. The offsets were subsequently added to the next feedforward increment calculated from the inverse kinematics of the planned trajectory, which was then converted to appropriate motor drive signals. By combining bounded feedback with predictive feedforward control, the platform was able to execute rapid and precise trajectories, including arbitrary tilts, rotations, and translations, while maintaining stability across all six degrees of freedom.



Supplementary Figure 1: Schematic of the feedback control reproduced from Extended Data Fig. 1b for convenience. (i) the current motor positions are read and the error relative to the target pose is computed at each update; (ii) the error is clamped to a maximum value to avoid overcompensation; (iii) the combined feedback and feedforward commands yield fast, accurate convergence with minimal overshoot and smooth trajectories.

1.5. Control system implementation

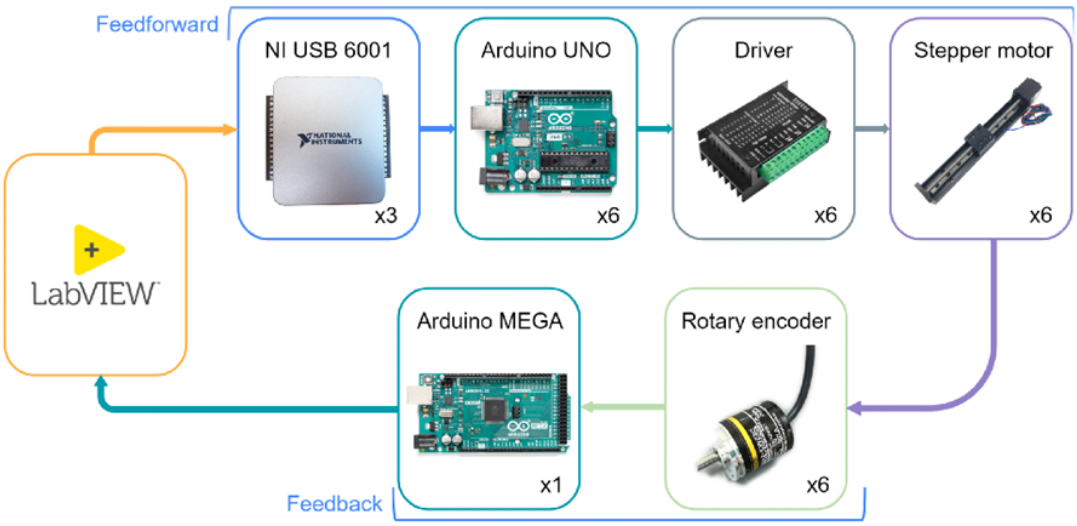
A schematic of the control system including real-time inverse kinematics and feedback control is shown in Extended Data Fig. 1a.



Supplementary Figure 2: Schematic of the platform control system reproduced from Extended Data Fig. 1a. Four modules (Position-Sampling Module, Feedback Module, Calculation Module, Voltage-Output Module) move the platform rapidly and sequentially toward a target pose. The Position sampling module receives precise stepper-motor positions via the encoder interface, and the Voltage output module drives the stepper motors via the actuator driver interface.

The control system comprised four functional modules: the calculation module, position sampling module, feedback module, and voltage output module. The calculation module determined the analog voltage output required to execute user-specified platform trajectories, including transitions between motions with different initial configurations, and transmitted this information to the voltage output module. The position sampling module received serial transmissions from the encoder interface and converted these signals into actuator positions. The feedback module compared the estimated actuator positions with the reference values, computed updated outputs to compensate for deviations in subsequent iterations, and sent these corrections to the voltage output module.

Feedback control was implemented using rotary encoders (E6A2-CWZ3C) and an Arduino MEGA (16 MHz, 54 channels), which computed rotation angles and transmitted the data serially to the control PC at approximately 640 Hz. The voltage output module was designed to provide both analog and digital voltage outputs to the actuator driver interface (TB6600), as well as digital voltage outputs to the drivers via an NI USB-6001. The actuator driver interface consisted of six Arduino Unos, which converted the analog signals into digital pulses to drive the stepper motors. The entire control program was developed in MATLAB (MathWorks) and LabVIEW (National Instruments).



Supplementary Figure 3: Schematic of the control architecture for the 6AGM. Schematic diagram of the control system architecture. Feedforward signals generated in LabVIEW were transmitted through NI USB-6001 devices (three units) to six Arduino UNOs, which provided digital control inputs to the actuator drivers (TB6600, six units). Each driver controlled a stepper motor (six units in total) for platform actuation. Feedback signals from six rotary encoders were collected by an Arduino MEGA, which transmitted the actuator position estimates to the control PC. This closed-loop configuration enabled precise trajectory execution through the combined feedforward and feedback pathways.

1.6. Encoder interface

The Arduino MEGA board received three inputs from each rotary encoder: phase A, phase B, and phase Z. An interrupt was triggered on the rising edge of phase A, which incremented or decremented the pulse count depending on the phase offset of phase B. At higher rotation speeds, this operation could occasionally result in missed or inverted counts. To correct such errors, the encoder interface utilized the phase Z signal to realign the pulse count once per revolution. On the first detection of a Z pulse, the current pulse count was stored as a reference. For every subsequent Z activation, the counter was adjusted so that its difference from the reference corresponded exactly to an integer multiple of the encoder's pulses per revolution. This correction greatly improved stability over long durations, substantially reduced long-term drift.

1.7. Actuator driver interface

The actuator driver interface consisted of six Arduino Uno boards, each of which generated pulse signals for one motor driver. Every 20 ms, the control computer sent a digital trigger to the interface, causing each Arduino to sample an analog input voltage. The measured voltage was then converted into a pulse interval, in microseconds, according to

$$\Delta\tau = \mathrm{round}\left(\frac{B}{V+A}\right), \tag{14}$$

where V denotes the 10-bit ADC value (0–1023 counts) corresponding to the analog input voltage, and

$$A = \frac{R\,t - r\,T}{T - t}, \tag{15}$$

$$B = \frac{T\,t\,(R - r)}{T - t}, \tag{16}$$

with parameter values

$$T = 10000, \qquad t = 100, \qquad R = 990, \qquad r = 0. \tag{17}$$

Here, T and t are the upper and lower bounds of the pulse interval, respectively, measured in microseconds, while R and r are the corresponding upper and lower bounds of the analog input range. Thus, increasing the input voltage decreases the pulse interval and therefore increases the pulse frequency sent to the motor driver.

To compensate for a consistent gain mismatch between the Arduino Uno analog-to-digital converter and the reference acquisition device (NI USB-6001), the upper bound of the analog input range was set to $R = 990$ rather than the nominal full-scale value of 1023. The analog input was digitized with 10-bit resolution, whereas pulse timing was generated using fast PWM with 16-bit resolution. The inverse mapping in Eq. (14) allocates finer timing resolution to higher actuator velocities, where smaller pulse intervals correspond to faster stepping.

An additional advantage of this mapping is that a fixed perturbation in the voltage reading produces an approximately speed-independent expected positional drift over time. This property improves the predictability of long-duration actuator behavior in the presence of measurement noise.

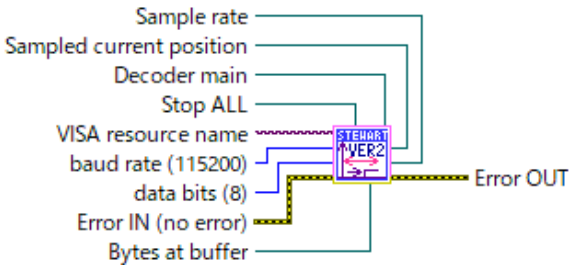
1.8. Motion generation

By integrating the modules described above, we achieved smooth and stable motion of the platform. The static and dynamic parameters of the system were as follows. By default, the platform was positioned 30 mm below the head plate of the head-fixed mouse, with the stage center located beneath the occipital region (5 mm posterior to bregma). The platform was capable of rotations of up to 15° about the anterior–posterior axis (roll), 10° about the medio–lateral axis (pitch), and 15° about the vertical axis (yaw). Translational movements reached up to 165 mm horizontally and 115 mm vertically.

1.9. LabVIEW software modules (Sub-VIs)

The LabVIEW program was divided into modules (Sub-VIs) to improve overall visibility and enable reuse of individual modules. The inputs, outputs, and functions of each Sub-VI are described below.

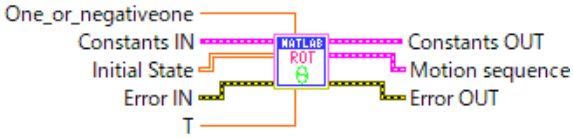
- **Position_sampling.vi**
  Reads data from the serial port. This function decodes the received data into positions of each actuator. Once all actuator positions are updated, the serial buffer is flushed to read the newest values.



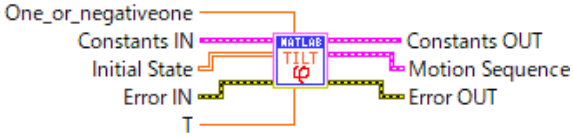
- **Calculate_motion.vi**
  Calculates motion based on a time series of six parameters. This function computes the inverse kinematics of the platform and outputs time series of voltage values and actuator positions to be used by `Output.vi` and `Feedback.vi`, respectively.

- **Initialization.vi**
  Initializes parameters used for inverse kinematics calculation.

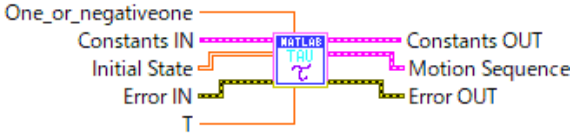
- **Motion_TRot.vi**
  Generates a time series of six parameters for tilted rotation motion.



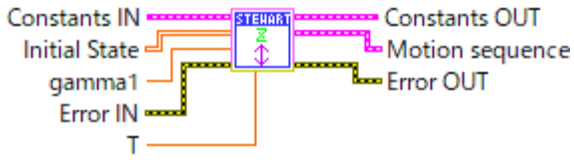
- **Motion_Roll.vi**
  Generates a time series of six parameters for roll motion.



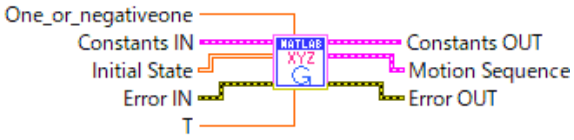
- **Motion_Yaw.vi**
  Generates a time series of six parameters for yaw motion.



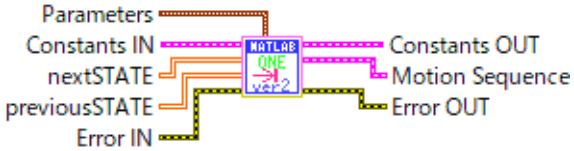
- **Motion_Vert.vi**
  Generates a time series of six parameters for vertical translation motion.



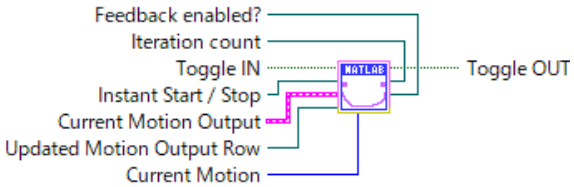
- **Motion_CircularHor.vi**
  Generates a time series of six parameters for horizontal circular motion.



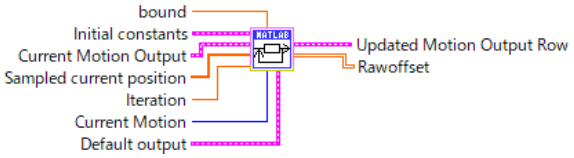
- **Motion_Transition.vi**
  Generates time series of six parameters for transition motion. Transition motions are calculated by linearly interpolating the final state of the previous motion and the initial state of the next motion.



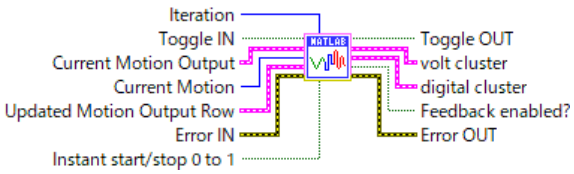
- **Output.vi**
  Iteratively updates analog and digital voltage applied from NI USB-6001. Every 20 ms, this VI updates the analog voltage applied to the actuator driver interface and the digital voltage applied to the motor drivers indicating the direction of motor rotation. Subsequently, it toggles its digital output to trigger an interrupt in the actuator driver interface that starts analog voltage acquisition. This VI compares the current iteration value and the iteration value passed from the feedback module and selects the updated value if the two indices match, and default values from the calculation module otherwise.



- **Feedback.vi**
  Compares the sampled position from `Position_sampling.vi` with the target trajectory from `Calculate_motion.vi`, and updates the voltage time series to compensate for the calculated offset. The feedback algorithm subtracts the observed actuator position from the reference position and clips it to a predetermined value. It then updates the feedforward increment by adding the clipped value to compensate for this offset in the next iteration output. The updated value goes through the voltage calculator and is used during the subsequent execution in `Output.vi`.



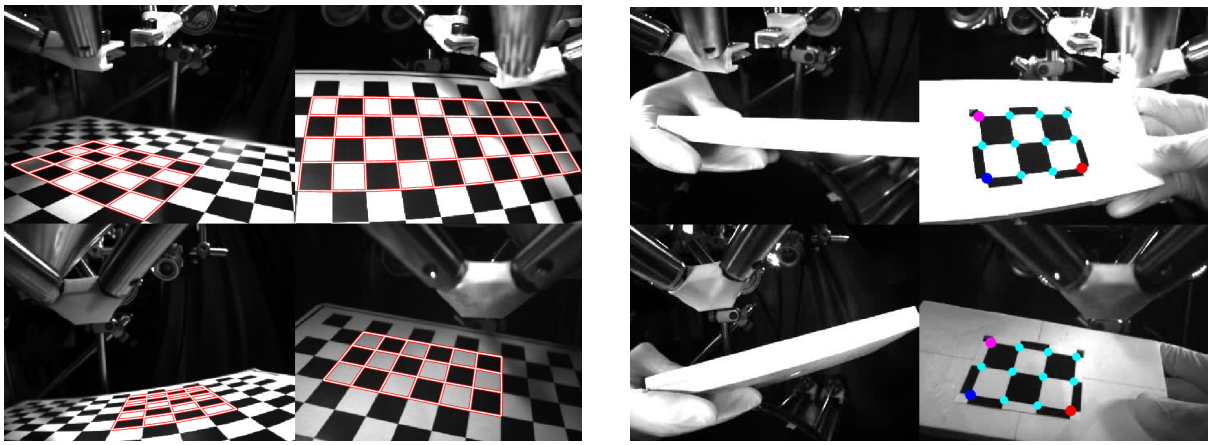
- **Voltage_output_selector.vi**
  This VI resides inside `Output.vi` and selects the appropriate output. If the iteration number tagged to the updated voltage from `Feedback.vi` matches the current iteration number of `Output.vi`, this VI forwards the updated voltage value. Otherwise, this VI routes the voltage provided by `Calculate_motion.vi`. The selected voltage is sent out over NI USB-6001.



2. Camera calibration

2.1. Calibration workflow and overview

In our experiments, we observed the posture of an animal using four machine vision cameras. From these images, we identified the positions of multiple body parts using DeepLabCut; however, these positions are in two-dimensional camera sensor coordinates. Therefore, it is necessary to reconstruct the three-dimensional coordinates for each body part. In this section, we describe the procedure for calibrating the camera model and its pose with respect to real-world coordinates using a checkerboard. We used a planar checkerboard calibration target consisting of a 10×14 grid with square size 19.25 mm for single-camera calibration. A total of 200 images were randomly selected from a calibration video for each of the four cameras. For stereo calibration, we used a separate checkerboard consisting of a 2×3 grid with square size 16 mm; similarly, 200 synchronized image pairs were randomly selected from calibration videos. Checkerboard corners were detected automatically using a publicly available MATLAB implementation provided by Geiger et al. [1]. Representative examples of checkerboard detection used for single-camera and stereo calibration are shown in Supplementary Fig. 4. Using the calibrated intrinsic and extrinsic parameters, corresponding 2D keypoints detected by DeepLabCut across cameras were triangulated to recover 3D coordinates.



a **b**

Supplementary Figure 4: Representative checkerboard detections used for camera calibration reproduced from Extended Data Fig. 1d, e.
a. Example of checkerboard corner detection for single-camera calibration.
b. Example of corresponding checkerboard corner detections used for stereo calibration.

2.2. Single-camera calibration

2.2.1 Linear pinhole parameter estimation

We estimated the linear pinhole camera parameters from multiple views of a planar checkerboard using a homography-based formulation related to the method of Zhang [2]. For a planar checkerboard ($Z = 0$), image points satisfy

$$s_j \tilde{\mathbf{q}}_j = \mathbf{K} \begin{bmatrix} \boldsymbol{\rho}_1 & \boldsymbol{\rho}_2 & \mathbf{t} \end{bmatrix} \tilde{\mathbf{Q}}_j = \mathbf{H} \tilde{\mathbf{Q}}_j, \tag{18}$$

where

$$\tilde{\mathbf{q}}_j = \begin{bmatrix} u_j \\ v_j \\ 1 \end{bmatrix}, \qquad \tilde{\mathbf{Q}}_j = \begin{bmatrix} X_j \\ Y_j \\ 1 \end{bmatrix}, \qquad \mathbf{K} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Let $\mathbf{H} = [\mathbf{h}_1 \ \mathbf{h}_2 \ \mathbf{h}_3]$. Instead of estimating the full homography, we estimate $\mathbf{h}_1$ and $\mathbf{h}_2$ directly from pairwise constraints between checkerboard points. For two image points, define the line

$$\boldsymbol{\ell}_{ij} = \tilde{\mathbf{q}}_i \times \tilde{\mathbf{q}}_j. \tag{19}$$

Using planar projection and subtracting two correspondences gives

$$\boldsymbol{\ell}_{ij}^{\top} (\mathbf{h}_1 \Delta X_{ij} + \mathbf{h}_2 \Delta Y_{ij}) = 0. \tag{20}$$

Stacking all such constraints yields

$$\mathbf{A}\mathbf{u} = 0, \qquad \mathbf{u} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{bmatrix}, \tag{21}$$

which is solved by SVD.

The intrinsic matrix is then recovered from the orthonormality of the rotation columns. Defining

$$\mathbf{B} = \mathbf{K}^{-\top}\mathbf{K}^{-1}, \tag{22}$$

we obtain

$$\mathbf{h}_1^{\top}\mathbf{B}\mathbf{h}_2 = 0, \tag{23}$$

$$\mathbf{h}_1^{\top}\mathbf{B}\mathbf{h}_1 = \mathbf{h}_2^{\top}\mathbf{B}\mathbf{h}_2. \tag{24}$$

These constraints are linear in the six independent entries of $\mathbf{B}$. After stacking constraints from all views, $\mathbf{B}$ is estimated from the null space of the resulting system, and $\mathbf{K}$ is recovered by Cholesky factorization.

For each view, extrinsic parameters are obtained as

$$\boldsymbol{\rho}_1 = \lambda\mathbf{K}^{-1}\mathbf{h}_1, \qquad \boldsymbol{\rho}_2 = \lambda\mathbf{K}^{-1}\mathbf{h}_2, \qquad \boldsymbol{\rho}_3 = \boldsymbol{\rho}_1 \times \boldsymbol{\rho}_2, \tag{25}$$

followed by projection onto $SO(3)$. The translation vector is estimated by least squares.

2.2.2 Radial distortion estimation (division model)

After estimating the linear intrinsic parameters, image points are converted to normalized camera coordinates,

$$\mathbf{q} = (x, y)^{\top}. \tag{26}$$

Radial distortion is modeled using the two-parameter division model

$$\mathbf{q}_u = \mathbf{d} + \frac{\mathbf{q} - \mathbf{d}}{1 + k_1 r^2 + k_2 r^4}, \qquad r^2 = \|\mathbf{q} - \mathbf{d}\|^2, \tag{27}$$

where $\mathbf{d} = (d_x, d_y)^{\top}$ is the distortion center and (k_1, k_2) are radial distortion coefficients.

The key constraint is that checkerboard rows and columns correspond to straight lines in the scene, and therefore should also be straight after correct undistortion. We use this property

to estimate the distortion parameters before final nonlinear bundle refinement.

One-parameter initialization. This stage corresponds to EstimateDivisionModel1 in Algorithm 1.

We first obtain an initial estimate of the distortion center and a coarse undistortion using the one-parameter division model

$$\mathbf{q}_u^{(0)} = \mathbf{d} + \frac{\mathbf{q} - \mathbf{d}}{1 + kr^2}. \tag{28}$$

For each checkerboard row or column, distorted image points are fit by the conic model

$$\alpha(x^2 + y^2) + \beta x + \gamma y + \delta \approx 0, \tag{29}$$

which is the image of a straight line under the one-parameter division model. Candidate values of $(k, \mathbf{d})$ are obtained from these line constraints, and the best candidate is selected by evaluating how straight the corresponding undistorted checkerboard lines become. Straightness is quantified by fitting a line to each undistorted row or column and measuring the orthogonal residual variance.

Two-parameter refinement from line straightness. This stage corresponds to EstimateDivisionModel2 in Algorithm 1.

After initialization, coordinates are recentered at the estimated distortion center,

$$\mathbf{p} = \mathbf{q} - \mathbf{d}, \qquad \mathbf{p}_u = \frac{\mathbf{p}}{1 + k_1 r^2 + k_2 r^4}. \tag{30}$$

For each checkerboard row or column, the ideal undistorted points satisfy a line equation

$$Ap_{ux} + Bp_{uy} + C = 0. \tag{31}$$

Substituting the division model gives

$$AX + BY + C(\mathbf{1} + k_1 R_1 + k_2 R_2) = 0, \tag{32}$$

where X and Y collect the centered distorted coordinates of the points on the line, and

$$R_1 = (r_1^2, \ldots, r_n^2)^\top, \qquad R_2 = (r_1^4, \ldots, r_n^4)^\top.$$

For each checkerboard line, the offset C is first estimated by fitting a line to the one-parameter undistorted points. In the subsequent two-parameter fit, the orientation parameters (A, B) are eliminated analytically by projection onto the orthogonal complement of $\mathrm{span}\{X, Y\}$. Specifically, let $\mathbf{W}$ denote the orthogonal projector onto the complement of $\mathrm{span}\{X, Y\}$, so that $\mathbf{W}X = 0$ and $\mathbf{W}Y = 0$. Premultiplying the stacked line constraint by $\mathbf{W}$ removes the nuisance orientation parameters, yielding a linear constraint in (k_1, k_2) while C is treated as fixed. This gives

$$k_1 \mathbf{r}_1 + k_2 \mathbf{r}_2 + \mathbf{s} = 0, \tag{33}$$

with

$$\mathbf{r}_1 = C\,\mathbf{W}R_1, \qquad \mathbf{r}_2 = C\,\mathbf{W}R_2, \qquad \mathbf{s} = C\,\mathbf{W}\mathbf{1}. \tag{34}$$

Stacking these constraints over all checkerboard rows and columns from all images gives the global least-squares problem

$$\min_{k_1,k_2} \sum_{\ell} \|k_1 \mathbf{r}_{1,\ell} + k_2 \mathbf{r}_{2,\ell} + \mathbf{s}_\ell\|^2 , \tag{35}$$

which is solved in closed form. In this way, the distortion parameters are estimated directly from the requirement that undistorted checkerboard lines be straight.

The resulting distortion parameters are then used to initialize a full nonlinear reprojection-error minimization together with the intrinsic and extrinsic parameters. All parameters are refined by minimizing the total reprojection error,

$$\min_{\boldsymbol{\theta}} \sum_{k,j} \|\mathbf{q}_{kj} - \pi(\mathbf{Q}_j; \boldsymbol{\theta})\|^2 , \tag{36}$$

where $\pi(\cdot)$ denotes the full projection model including distortion.

Algorithm 1 summarizes the practical estimation procedure corresponding to the mathematical formulation above, including robust subset selection, distortion estimation, inlier refinement, and final nonlinear optimization.

Algorithm 1 Single-camera calibration

Require: Calibration video $video_{\text{cam}}$, checkerboard geometry
Ensure: Refined camera model $(\mathbf{K}, \mathbf{D}, \text{Extr})$

Sample frames from $video_{\text{cam}}$, cluster them, and select a diverse subset
Detect checkerboards in selected images and collect valid boards $\mathcal{B}$
$bestScore \leftarrow +\infty$
for $iter = 1$ to N_{iter} **do**
    $\mathcal{F} \leftarrow \text{RandomSubset}(\mathcal{B}, N_{\text{fit}})$
    $(\mathbf{K}_0, \text{Extr}_0) \leftarrow \text{EstimatePinholeParams}(\mathcal{F})$
    $\mathbf{q}_{\text{norm}} \leftarrow \text{NormalizeCoordinates}(\mathcal{F}, \mathbf{K}_0, \text{Extr}_0)$
    $(k_1, \mathbf{d}) \leftarrow \text{EstimateDivisionModel1}(\mathbf{q}_{\text{norm}})$
    $(k_1, k_2) \leftarrow \text{EstimateDivisionModel2}(\mathbf{q}_{\text{norm}}, \mathbf{d})$
    $\mathbf{D} \leftarrow (k_1, k_2, \mathbf{d})$
    $\mathbf{q}_{\text{undist}} \leftarrow \text{Undistort}(\mathbf{q}_{\text{norm}}, \mathbf{D})$
    $\mathbf{x}_{\text{undist}} \leftarrow \text{DenormalizeToPixels}(\mathbf{q}_{\text{undist}}, \mathbf{K}_0)$
    $(\mathbf{K}_1, \text{Extr}_1) \leftarrow \text{EstimatePinholeParams}(\mathcal{F}, \mathbf{x}_{\text{undist}})$
    Define inliers $\mathcal{I}_{\text{in}} \subset \mathcal{B}$ using reprojection error threshold θ_{reproj}
    $\mathbf{x}_{\text{in}} \leftarrow \text{UndistortAndDenormalize}(\mathcal{I}_{\text{in}}, \mathbf{K}_1, \mathbf{D})$
    $(\mathbf{K}_2, \text{Extr}_2) \leftarrow \text{EstimatePinholeParams}(\mathcal{I}_{\text{in}}, \mathbf{x}_{\text{in}})$
    $score \leftarrow \text{MeanReprojectionError}(\mathcal{I}_{\text{in}}, \mathbf{K}_2, \mathbf{D})$
    if $score < bestScore$ **then**
        $bestScore \leftarrow score$
        $(\mathbf{K}, \mathbf{D}, \text{Extr}) \leftarrow (\mathbf{K}_2, \mathbf{D}, \text{Extr}_2)$
        $bestInliers \leftarrow \mathcal{I}_{\text{in}}$
        $bestCenter \leftarrow \mathbf{d}$
    end if
end for
for $s = 1$ to N_{stab} **do**
    $\mathbf{q}_{\text{norm}} \leftarrow \text{NormalizeCoordinates}(bestInliers, \mathbf{K}, \text{Extr})$
    $(k_1, k_2) \leftarrow \text{EstimateDivisionModel2}(\mathbf{q}_{\text{norm}}, bestCenter)$
    $\mathbf{D} \leftarrow (k_1, k_2, bestCenter)$
    $\mathbf{q}_{\text{undist}} \leftarrow \text{Undistort}(\mathbf{q}_{\text{norm}}, \mathbf{D})$
    $\mathbf{x}_{\text{undist}} \leftarrow \text{DenormalizeToPixels}(\mathbf{q}_{\text{undist}}, \mathbf{K})$
    $(\mathbf{K}, \text{Extr}) \leftarrow \text{EstimatePinholeParams}(bestInliers, \mathbf{x}_{\text{undist}})$
    if Converged$(\mathbf{K}, \mathbf{D})$ **then**
        break
    end if
end for
$(\mathbf{K}, \mathbf{D}, \text{Extr}) \leftarrow \text{NonlinearReprojectionMinimization}(bestInliers, \mathbf{K}, \mathbf{D}, \text{Extr})$
return $(\mathbf{K}, \mathbf{D}, \text{Extr})$

2.3. Stereo calibration

This stage estimates the relative pose between two cameras from corresponding checkerboard detections after intrinsic calibration and radial distortion correction. Let $\mathbf{q}_{1,i}$ and $\mathbf{q}_{2,i}$ denote corresponding normalized image points in the two cameras. The relative geometry is described

by the essential matrix $\mathbf{E}$, which satisfies

$$\mathbf{q}_{2,i}^{\top}\mathbf{E}\mathbf{q}_{1,i} = 0. \tag{37}$$

2.3.1 Essential matrix estimation

The essential matrix was estimated from normalized correspondences using the standard eight-point algorithm. The linear estimate was then projected onto the essential-matrix manifold by singular-value decomposition, enforcing rank 2 with equal nonzero singular values.

2.3.2 Pose recovery

The relative rotation $\mathbf{R}$ and translation direction $\mathbf{t}$ were recovered from the essential matrix using the standard decomposition. Among the candidate solutions, the physically valid pose was selected by requiring triangulated checkerboard points to lie in front of both cameras.

2.3.3 Triangulation and scale recovery

Given the recovered relative pose, checkerboard corners were triangulated from the stereo correspondences. Because the essential matrix determines translation only up to scale, the metric scale was recovered using the known checkerboard geometry. This yielded the full stereo transformation

$$\mathbf{T}_{2\leftarrow 1} = (\mathbf{R}, \mathbf{t}). \tag{38}$$

2.3.4 Diagnostics

Calibration quality was evaluated from triangulation consistency and geometric error after reconstruction. These measures were also used for robust model selection during RANSAC-based estimation.

Algorithm 2 summarizes the practical stereo-calibration procedure corresponding to the formulation above, including essential-matrix estimation, pose recovery, triangulation-based inlier selection, and final model selection.

Algorithm 2 Stereo calibration

Require: Synchronized calibration videos ($video_{\text{cam1}}, video_{\text{cam2}}$), checkerboard geometry, refined single-camera models $(\mathbf{K}_1, \mathbf{D}_1)$ and $(\mathbf{K}_2, \mathbf{D}_2)$

Ensure: Stereo transform $\mathbf{T}_{2\leftarrow 1}$

Sample synchronized frame pairs and collect valid stereo board pairs $\mathcal{P}$
$bestScore \leftarrow +\infty$
for $iter = 1$ to N_{iter} **do**
    $\mathcal{F} \leftarrow \textsc{RandomSubset}(\mathcal{P}, N_{\text{fit}})$
    $\mathbf{q}^{(1)} \leftarrow \textsc{NormalizeAndUndistort}(\mathcal{F}, \mathbf{K}_1, \mathbf{D}_1)$
    $\mathbf{q}^{(2)} \leftarrow \textsc{NormalizeAndUndistort}(\mathcal{F}, \mathbf{K}_2, \mathbf{D}_2)$
    $\mathbf{T}^{(0)}_{2\leftarrow 1} \leftarrow \textsc{EstimateStereoTransform}(\mathbf{q}^{(1)}, \mathbf{q}^{(2)}, \mathcal{F})$
    $e_{\text{tri}} \leftarrow \textsc{TriangulationError}(\mathbf{q}^{(1)}, \mathbf{q}^{(2)}, \mathbf{T}^{(0)}_{2\leftarrow 1})$
    $\mathcal{I}_{\text{in}} \leftarrow \textsc{SelectInliers}(\mathcal{P}, \mathbf{T}^{(0)}_{2\leftarrow 1}, \theta_{\text{tri}})$
    $\mathbf{q}^{(1)}_{\text{in}} \leftarrow \textsc{NormalizeAndUndistort}(\mathcal{I}_{\text{in}}, \mathbf{K}_1, \mathbf{D}_1)$
    $\mathbf{q}^{(2)}_{\text{in}} \leftarrow \textsc{NormalizeAndUndistort}(\mathcal{I}_{\text{in}}, \mathbf{K}_2, \mathbf{D}_2)$
    $\mathbf{T}_{2\leftarrow 1} \leftarrow \textsc{EstimateStereoTransform}(\mathbf{q}^{(1)}_{\text{in}}, \mathbf{q}^{(2)}_{\text{in}}, \mathcal{I}_{\text{in}})$
    $e_{\text{tri}} \leftarrow \textsc{TriangulationError}(\mathbf{q}^{(1)}_{\text{in}}, \mathbf{q}^{(2)}_{\text{in}}, \mathbf{T}_{2\leftarrow 1})$
    $score \leftarrow \textsc{MeanTriangulationError}(e_{\text{tri}})$
    if $score < bestScore$ **then**
        $bestScore \leftarrow score$
        $bestTransform \leftarrow \mathbf{T}_{2\leftarrow 1}$
    end if
end for
return $bestTransform$

References

[1] Andreas Geiger, Frank Moosmann, Oemer Car, and Bernhard Schuster. Automatic calibration of range and camera sensors using a single shot. In *International Conference on Robotics and Automation (ICRA)*, 2012.

[2] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.