Journal of Neuroscience Methods 327 (2019) 108403

Contents lists available at ScienceDirect

Journal of Neuroscience Methods

journal homepage: www.elsevier.com/locate/jneumeth

Virtual reality for animal navigation with camera-based optical flow tracking

Ivan Vishniakou[a], Paul G. Plöger[b], Johannes D. Seelig[a,*]

[a] Center of Advanced European Studies and Research (caesar), Bonn, Germany
[b] Department of Computer Science, Hochschule Bonn-Rhein-Sieg, Sankt Augustin, Germany

ARTICLE INFO

Keywords:
Virtual reality
Navigation
Optical flow
Spherical treadmill
Ball tracking
Drosophila
Real-time image processing

ABSTRACT

Background: Virtual reality combined with a spherical treadmill is used across species for studying neural circuits underlying navigation and learning.
New method: We developed an optical flow-based method for tracking treadmill ball motion in real time using a single high-resolution camera.
Results: Tracking accuracy and timing were determined using calibration data. Ball tracking was performed at 500 Hz and integrated with an open source game engine for virtual reality projection. The projection was updated at 120 Hz with a latency with respect to ball motion of 30 ± 8 ms. The system was tested for behavior with fruit flies. The application and source code are available at https://github.com/ivan-vishniakou/neural-circuits-vr.
Comparison with existing method(s): Optical flow-based tracking of treadmill motion is typically achieved using optical mice. The camera-based optical flow tracking system developed here is based on off-the-shelf components and offers control over the image acquisition and processing parameters. This results in flexibility with respect to tracking conditions – such as ball surface texture, lighting conditions, or ball size – as well as camera alignment and calibration.
Conclusions: A fast system for rotational ball motion tracking suitable for virtual reality behavior with fruit flies was developed and characterized.

1. Introduction

Virtual reality (VR) is used across species for studying neural circuits underlying behavior (Dombeck and Reiser, 2012). In many implementations, animals navigate through virtual realities on a spherical treadmill – a ball which can be freely rotated around its center of mass (Dombeck and Reiser, 2012; Mason et al., 2001; Lott et al., 2007; Dombeck et al., 2007; Seelig et al., 2010).

Tracking of ball rotation is typically accomplished using optical mice which are based on low-resolution, high-speed cameras integrated with a light source for measuring displacements when moving across a surface. Movement across the surface results in optical flow – the displacement of features across the camera sensor. Such features can for example be speckle-like reflections from surface roughness; comparing these speckle images between different frames then allows computing the displacement using hardware-integrated image processing.

Optical mice come however with some limitations for measuring ball rotation. First, a single optical mouse measures displacements only in two directions and therefore two mice are required for tracking all three degrees of freedom of ball motion. Secondly, limited or no control over the onboard processing algorithms as well as camera settings requires careful calibration. In particular, if the mouse sensors cannot be placed in direct proximity of the ball surface, accurate alignment of the two sensors as well as calibration with respect to surface properties and lighting conditions is necessary (Seelig et al., 2010). As an approach that overcomes some of these limitations, real-time tracking was developed with a single high-resolution camera for situations where a uniquely patterned ball can be used (Moore et al., 2014). In that case, ball orientation was calculated by matching each recorded frame to a map of the entire ball surface pattern. Using a high-resolution camera allows control over all recording parameters and offers the freedom to choose a custom algorithms and test its performance with simulated data (Moore et al., 2014).

Here, we combine the flexibility of optical flow-based tracking with the advantages of using a high-resolution camera. The developed system tracks optical flow of rotational ball movement at 500 Hz using

* Corresponding author.
E-mail address: johannes.seelig@caesar.de (P.G. Plöger).

https://doi.org/10.1016/j.jneumeth.2019.108403
Received 19 March 2019; Received in revised form 19 August 2019; Accepted 20 August 2019
Available online 23 August 2019

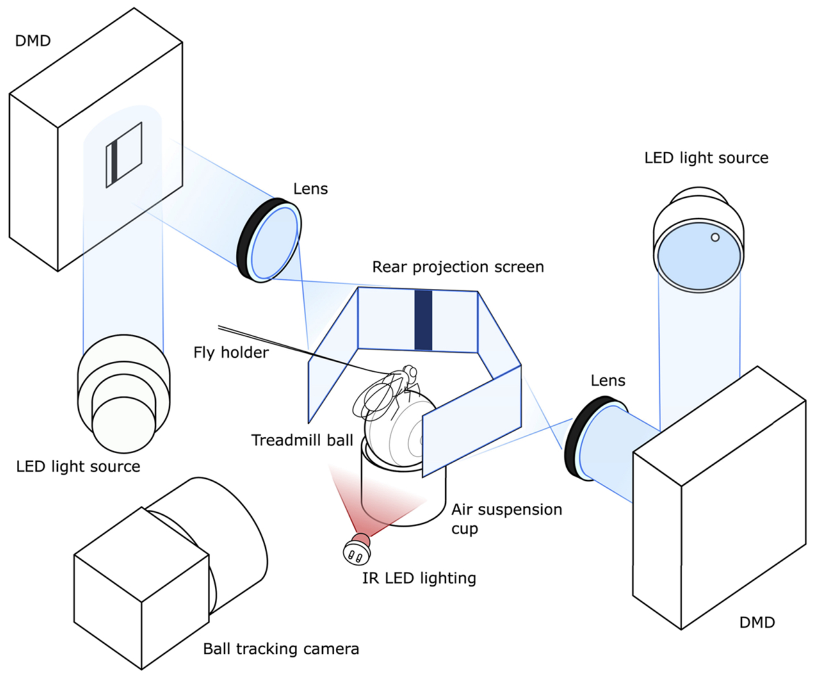


Fig. 1. A schematic of behavior setup for *Drosophila melanogaster*. A 6-mm polyurethane ball is air-suspended in a holder and serves as an omnidirectional treadmill. The fly is glued to a thin metal wire. A rear projection screen is used for displaying the virtual reality (VR) in the fly's field of view. Two DMDs, LED light sources, and two projection lenses are combined to project from two sides onto the screen. A single tracking camera is set up to capture ball images at the rate of 500 fps. Infrared LED lighting invisible to the fly is used for illuminating the ball.

a single camera and is integrated with an open source virtual reality environment and two projectors (Haberkern et al., 2018) with a refresh rate of 120 Hz (Turner-Evans et al., 2017) and overall latency of 30 ± 8 ms – from detecting ball movement to projecting the updated virtual reality image. This is achieved using off-the-shelf hardware components and open source software. The system is easy to align and calibrate, can be used under different conditions and at different scales, and can be integrated with two-photon imaging or electrophysiology. The system was tested for behavior with fruit flies, which showed a robust closed-loop response.

2. Materials and methods

2.1. Virtual reality setup components

A schematic of the setup is shown in Fig. 1 and an actual setup in Fig. 19. Two digital micromirror devices (DMD, DLP LightCrafter 6500 by Texas Instruments) were used for projecting the VR onto an angled screen from two sides (Turner-Evans et al., 2017; Haberkern et al., 2018). The projectors are FullHD digital micromirror devices with HDMI and DisplayPort interfaces allowing them to display images the same way as standard monitors. In our setup, the DMD was set to use DisplayPort input and display the frames at a maximum frame rate of 120 Hz (see Section A.5 in the Appendix for details). This frame rate can only be achieved when frames are displayed with bit depth 1 (binary images). To display different gray levels, a dithering technique is used and the density of white pixels is varied proportionally to the brightness level of the displayed region with an ordered Bayer matrix (Bayer, 1973) (see Section A.5 in the Appendix for details). Two collimated blue LEDs (M470L3, Thorlabs) were used as light sources. Light reflected off the DMD was projected from the outside (seen from the fly's perspective) onto a screen made from black paper.

The spherical treadmill used for the current experiments is described in Seelig et al. (2010). Briefly, a 6 mm diameter polyurethane foam ball is held in a cup-shaped holder and suspended in an air stream. The ball is illuminated with two IR LEDs (Thorlabs LED850LN – 850 nm LED, TO-39, and Mightex LED Driver, SLA series) (Turner-Evans et al., 2017) and monitored with a camera (Basler acA640-750um). The camera uses a USB 3 interface to communicate with the PC, and the bandwidth of 350 MB/s allows transfer of full-frame images (640 × 480 × 8 bpp mono color) at a rate of 751 frames per second (fps). For tracking, a frame rate of 500 fps was used, with an exposure time of 100 μs under infrared illumination. The resolution was 224 × 140 pixels, which results from cropping and 2 × 2 binning of the full frame. Binning (summing the signal from adjacent pixels into a single one) increased signal-to-noise ratio and reduced the image readout and transfer time. A Computar camera objective M2514-MP2 F1.4, f = 25 mm together with a Computar EX2C extender were used for imaging the ball (Seelig et al., 2010).

2.2. Integration with virtual reality

For virtual reality applications a 3D rendering game engine (Urho3D, an open-source C++ game engine (Urho3D contributors, 2019a)) was used and the *x*, *y*- and *z*-rotations of the ball were mapped to planar motion (forward-, sidestepping- and turning components) of the animal in the virtual reality (this was done similarly to Moore et al. (2014), see Section A.4 in the Appendix for details).

The application is designed to run ball tracking and environment rendering in two separate execution threads. The image processing thread is a loop acquiring camera frames and calculating ball displacements, tuned to run one iteration in under 2 ms, resulting in a tracking frequency of 500 fps. The rendering thread runs the game engine, which models a virtual environment in 3D. See Fig. 18 for a screenshot of the runnign software.

Setting up the virtual environment requires specifying the arena geometry as well as lighting and rendering properties. Further, the virtual cameras have to be configured such that the virtual environment is rendered from the angles corresponding to the position of the screen. Additional parameters are required to describe the mapping of ball rotations to movement in the virtual environment; these transformations take into account the size of the ball and the orientation of the tracking camera relative to the forward direction of the animal. This is all done with the in-built Urho3D engine's scripting language which also describes each environment in a separate script file, containing instructions about which objects are to be created in the scene. In each update of the game engine the ball tracking signal is interpreted as the displacement of the animal, and the virtual camera positions and orientations are updated before being rendered and displayed on the screen.

Other game engine capabilities used in the application are dithering shader to output binary frames suitable for the DMD and networking to broadcast the state of the VR to any client, for example for monitoring the VR or a script that triggers any other external stimuli. The application can also be used without visual output for tracking the animal's activity. The configuration of the virtual environment is described in detail in Section A.4 in the Appendix as well as in the source code.

The tracking data is saved in a text log file: for each camera frame a new line is added with the timestamp of the frame, *x*-, *y*-, *z*-ball displacements and *x*-, *y*-, *z*-position of the animal in the virtual environment. Additionally, the tracking camera is set to output a frame trigger pulse with each recorded frame, which can be used for synchronization with other applications, such as two-photon imaging.

All experiments were performed on a PC with an Intel Xeon E5-1620 v3 @ 3.50 GHz (8 cores) CPU, 32 Gb DDR 4 @ 2993 MHz of RAM, a NVIDIA Quadro K620, 2 Gb DDR3 RAM, 384 CUDA cores GPU and Microsoft Windows 8.1 Enterprise edition operating system. OpenCV 3.4.1 and opencv-contrib module were compiled using Microsoft Visual Studio Toolkit 14.0 (VS2015) in release configuration with CUDA 9.1 for tests of the GPU-accelerated algorithms.

3. Theory and algorithm

As seen in Fig. 2a–c, rotations around the three different camera frame axes produce recognizably different velocity field distributions. To distinguish them, a ring-shaped region of interest is selected around the center of the ball (Fig. 2d–f). For each point in the ROI, two principal directions are selected: radial (i.e., orthogonal) and tangential to the ROI circle at this point, directed counterclockwise. The optical flow vector is then projected onto these directions to find its radial and tangential components with respect to the ROI (Fig. 2g–i). These radial and tangential components of the optical flow build distinct distributions over the ROI, which allows to separate the motion components (Fig. 2j–l).

We found empirically that these distributions can be fitted well by the following two functions:

$$\begin{cases} f_{\text{rad}}(\varphi) = & c_{xy\ \text{rad}} \cdot \omega_{xy} \cdot \Delta t \cdot \sin(\varphi + \phi) \\ f_{\tan}(\varphi) = & c_{xy\ \tan} \cdot \omega_{xy} \cdot \Delta t \cdot \cos(\varphi + \phi) + c_z \cdot \omega_z \cdot \Delta t. \end{cases} \tag{1}$$

Here, φ is the azimuth of a point in the ROI represented in polar coordinates, $\omega_{xy} \cdot \Delta t$ and $\omega_z \cdot \Delta t$ are the angular displacements about a rotational axis lying in the *xy*-plane or on the *z*-axis, respectively, ϕ is the orientation of the axis of rotation in the *xy*-plane inducing the corresponding phase offset of the optical flow distributions. $c_{xy\ rad}$, $c_{xy\ tan}$, and c_z (px/rad) are calibration factors relating the angular displacements $\omega_{xy} \cdot \Delta t$ and $\omega_z \cdot \Delta t$ to the angular velocity vectors lying in the *xy*-plane and *z*-plane, respectively (see Section A.9 in the Appendix for details). Note, that radial and tangential components of optical flow induced by ω_{xy} rotation have different coefficients, since the respective optical flow distributions are different, while the ω_z rotation induces only tangential optical flow, and a single coefficient c_z is sufficient. The calibration factors subsume all setup parameters, such as camera focal length, and can be determined as described below and in Section A.9 in the Appendix.

3.1. Finding calibration factors by function fitting

Calibration factors were determined experimentally by recording a sequence of frames with two cameras pointed at the ball center and positioned orthogonally to each other (see Fig. 16 in the Appendix). The z-component is a planar motion (that is, in the camera focal plane) which can be measured accurately with optical flow (as was verified using simulated as well as motor actuated ground truth data) and the $c_{xy\ rad}$ and $c_{xy\ tan}$ coefficients can be found by regression between the estimates of the ball motion based on the two cameras. This is described in more detail in Section A.9.3 of the Appendix and the corresponding Python script for running the calibration is available at https://github.com/ivan-vishniakou/neural-circuits-vr.

3.2. Tracking algorithm

The ball tracking algorithm takes two consecutive frames of ball motion and calculates the three independent rotational displacements of the ball, i.e. the *x*-, *y*- and *z*-axis angular displacements, by fitting their projections onto the radial and tangential directions with function (1) in a circular ROI. Since the distribution of optical flow in the circular ROI is noisy the optical flow is averaged over a band (limited by minimal inner and maximal outer rings, see Section A.8 in the Appendix for how the optimal ROI size was determined) in the radial direction to increase robustness. This results in the following algorithm:

```
   Result: Angular velocity vector ($\omega_x$, $\omega_y$, $\omega_z$)
   Input: Two grayscale images of the ball video: $frame_1$, $frame_2$
1  polar transform $frame_1$, $frame_2$ with respect to the center of the ROI
2  crop the polar-transformed $frame_1$, $frame_2$ to keep only the ROI
3  $flow_u$, $flow_v \longleftarrow$ compute_optical_flow($frame_1(cropped)$, $frame_2(cropped)$)
4  $flowROI_{rad}$, $flowROI_{tan} \longleftarrow$ average $flow_u$, $flow_v$ over radius
5  fit $flowROI_{rad}$, $flowROI_{tan}$ with function (1) by finding $(\omega_{xy} \cdot \Delta t)$, $\phi$, $(\omega_z \cdot \Delta t)$
6  $(\omega_x, \omega_y, \omega_z) \longleftarrow \omega_{xy}\sin(\phi), \omega_{xy}\cos(\phi), \omega_z$
   return ($\omega_x$, $\omega_y$, $\omega_z$)
```

Data from the algorithm's intermediate steps are shown in Fig. 3. The polar transform (Fig. 3b) is calculated before computing optical flow, allowing the following optimizations: first, the ring-shaped ROI transforms into a rectangular image section, where one dimension corresponds to the azimuthal angle of the ROI, φ, and the other to the radius, ρ. This means that the ROI can be isolated by cropping the image into a rectangular strip, and optical flow can be calculated for a small fraction of the full frame. Secondly, in the polar-transformed ROI, the optical flow *u* and *v* components correspond to the radial and tangential motions directly. This avoids having to calculate these flows from optical flow obtained in Cartesian coordinates. The dependence of the optical flow calculation on the ROI position which results from the polar transform (see Fig. 3b) is taken into account in the experimentally determined calibration coefficients.

Other optimizations included passing over the polar-transformed and cropped frame to the next iteration to save on those operations, and using the previous iteration's function fitting result as a prior for the solver in the next frame. Function fitting was done with Downhill Solver included in OpenCV, which implements the Nelder-Mead downhill simplex method (Nelder and Mead, 1965).

For the source code or a compiled executable of the VR application please refer to the repository at https://github.com/ivan-vishniakou/neural-circuits-vr or send an email to ivan.vishniakou@caesar.de or johannes.seelig@caesar.de.

4. Results

4.1. Tracking algorithm implementation

For real-time applications the optical flow calculation was implemented in a C++ program using OpenCV 3.4.1. Best performance in terms of processing speed and tracking accuracy was achieved with the Farneback optical flow algorithm on a 60 × 140 ROI scaled down to 30 × 140 (see Section A.7 in the Appendix for a comparison of a range of optical flow algorithms and for selecting ROI parameters). Preprocessing (polar transform, cropping and resizing) together with optical flow estimation took 1.45 ms (standard deviation s = 0.54 ms), the function fitting by downhill solver was capped at 30 iterations to keep its run-time under 0.4 ms. These parameters allow to achieve real-time tracking at 500 frames per second with the current PC configuration. All time measurements were performed with the cv::TickMeter

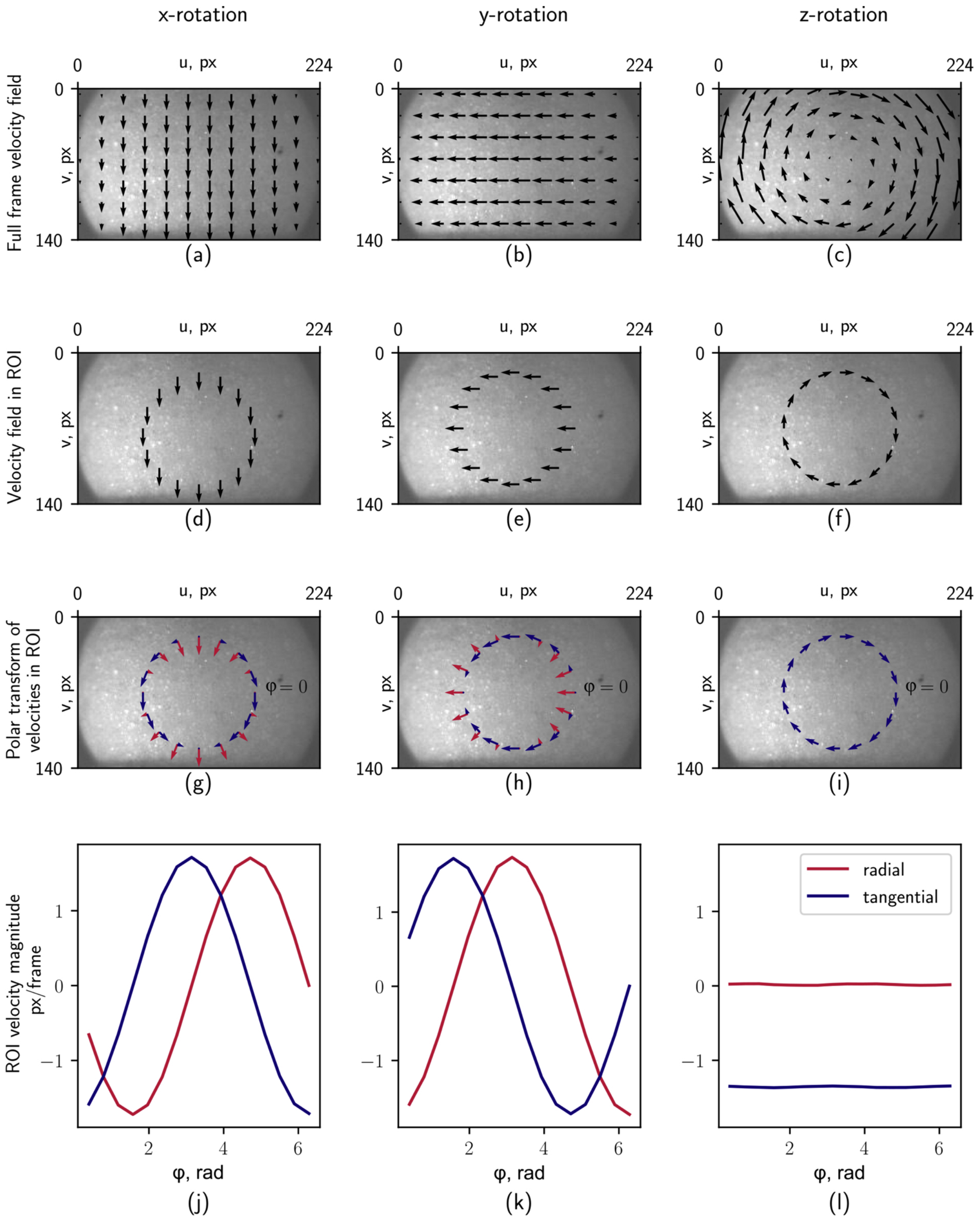


Fig. 2. Estimated optical flow produced by rotations of the ball around the respective three independent axes of rotation.

class.

We noticed that occasionally frames were dropped, for example when starting the program or due to other operating system-related processes. Such frame drops can be detected in the saved data file by looking for time stamps with a separation of more than 2 ms. If a single or (very rarely) multiple frames are dropped, the next displacement is calculated with the latest available frame before the frame drop over the corresponding longer time interval. As long as the increased ball displacements stay within the tracking operating range (the window over which the correlation is calculated), the accuracy is not affected.

4.2. Evaluation of tracking accuracy

The evaluation of tracking accuracy is based on comparing ground truth $\boldsymbol{\omega}_{\mathbf{GT}}$ and estimated angular velocity $\boldsymbol{\omega}_{\mathbf{est}}$. Three metrics are selected for this comparison: the absolute error (in degrees per frame)

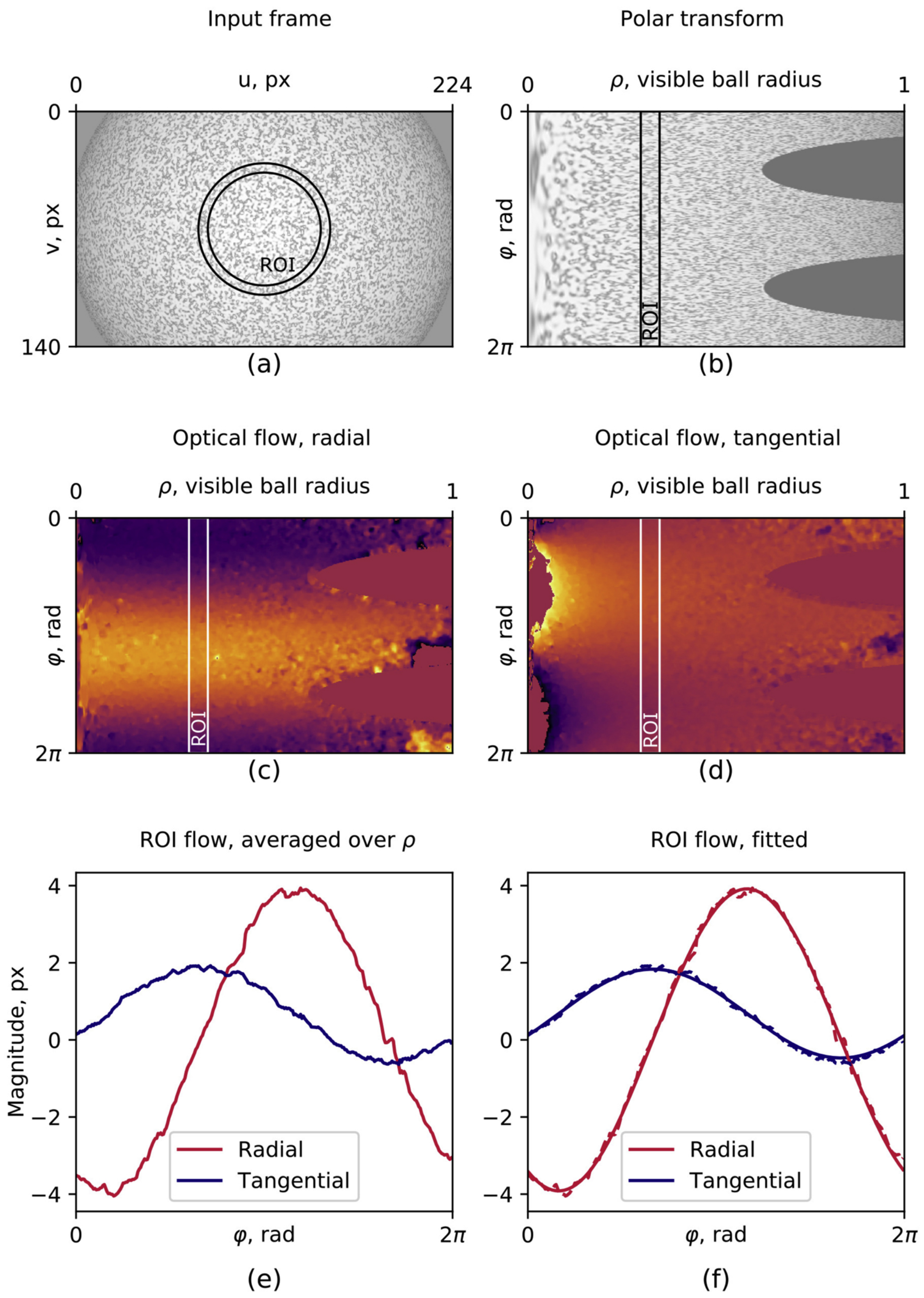


Fig. 3. Ball rotation evaluation from optical flow. Input frame (a) is polar-transformed (b) and optical flow (c, d) is extracted by comparing it to the previous frame. The distribution of optical flow in the ROI (e) is then fitted with function (1) (f).

$$\varepsilon_{\mathrm{magn\_abs}} = \mathrm{abs}(\|\omega_{\mathbf{est}}\| - \|\omega_{\mathbf{GT}}\|),$$

the relative (percentage) magnitude error

$$\varepsilon_{\mathrm{magn\_relative}} = \frac{\varepsilon_{\mathrm{magn\_abs}}}{\|\omega_{\mathbf{GT}}\|}\cdot 100\%,$$

and the orientation error (in degrees), which is the angle between the ground truth and estimated rotation vectors in the plane spanned by these two vectors,

$$\varepsilon_{\mathrm{orient}} = \cos^{-1}\left(\frac{\omega_{\mathbf{est}}\cdot\omega_{\mathbf{GT}}}{\|\omega_{\mathbf{est}}\|\|\omega_{\mathbf{GT}}\|}\right).$$

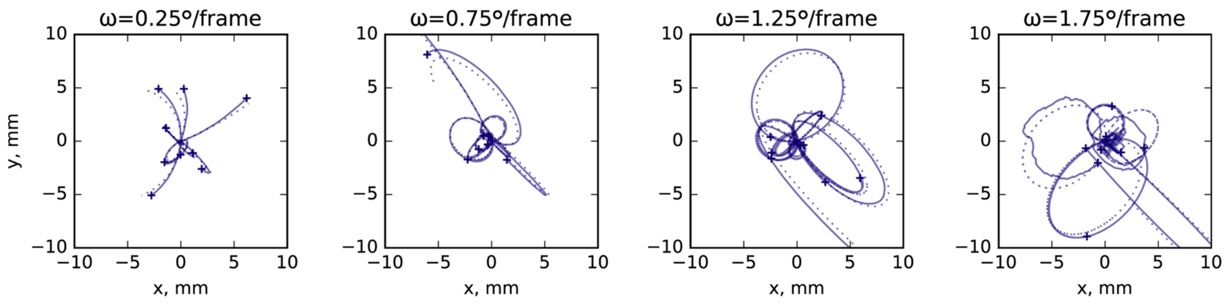


Fig. 4. Walking trajectories reconstructed from simulated ball rotation tracking (solid) and corresponding ground truth (dashed) at different angular velocities ω. Each trajectory's duration is 1 s (500 frames) starting at the origin (0, 0); the endpoint is marked with a cross mark. Projection of ball rotations into the two-dimensional walking trajectory was done as in Moore et al. (2014).

Two ground truth datasets were generated for testing the tracking performance (see Section A.6 Appendix for details). One consisted of simulated data, which offered control of all rotation parameters, while however not fully replicating surface texture or lighting and camera conditions encountered in the actual experiment. A second dataset was generated by rotating balls of three different sizes with a stepper motor.

Using the simulated data with accelerated rotations the operation range of the tracking algorithm was determined. For each of the three rotation axes and rotation speeds in this dataset the tracking errors were determined (see Fig. 17). The tracking error depends on the rotation speed of the ball (for the ROI and parameters used here, see Fig. 17 in the Appendix). The average tracking error stays under 10% (magnitude) and 7.5° (orientation) for rotations up to 1.70° per frame, which corresponds to 45 mm/s linear speed for a 6 mm diameter ball. Figs. 4 and 5 show the comparison of the integrated tracked trajectories with ground truth data.

Using a stepper-motor a ball was rotated at constant speed around a single axis (see Section A.6 in the Appendix for details). The integrated rotation magnitude error for the recorded sequences are shown in Table 5.

4.3. VR system timing

The run-time of the tracking algorithm was measured using internal timing. The total latency of the combined tracking and VR setup was estimated using a high-speed camera filming the VR display: the display was triggered to change state, and the state change was detected with the tracking camera. This sequence contains all the latency components for one cycle of a closed loop VR experiment.

Table 1
Latency measurement of a camera – virtual reality – display system.

Display mode	Median latency, ms
120 Hz	25 ± 8 ms
120 Hz, V-sync	33 ± 8 ms
60 Hz	50 ± 16 ms
60 Hz, V-sync	50 ± 16 ms

The main component of the latency is the display device input lag. It depends strongly on the selected video mode (Table 1) and the lowest values were found at 120 Hz update rate with V-sync disabled.

4.4. Behavior experiments in VR

For testing the setup for fly behavior, we used a simple VR with a single bright stripe of 10° width and 45° height projected onto a two-part screen similar to Haberkern et al. (2018) of 9 mm height and 2 times 18 mm width made out of black paper. The fly's head was fixed to its thorax with UV glue, and the wings were glued to the tethering pin

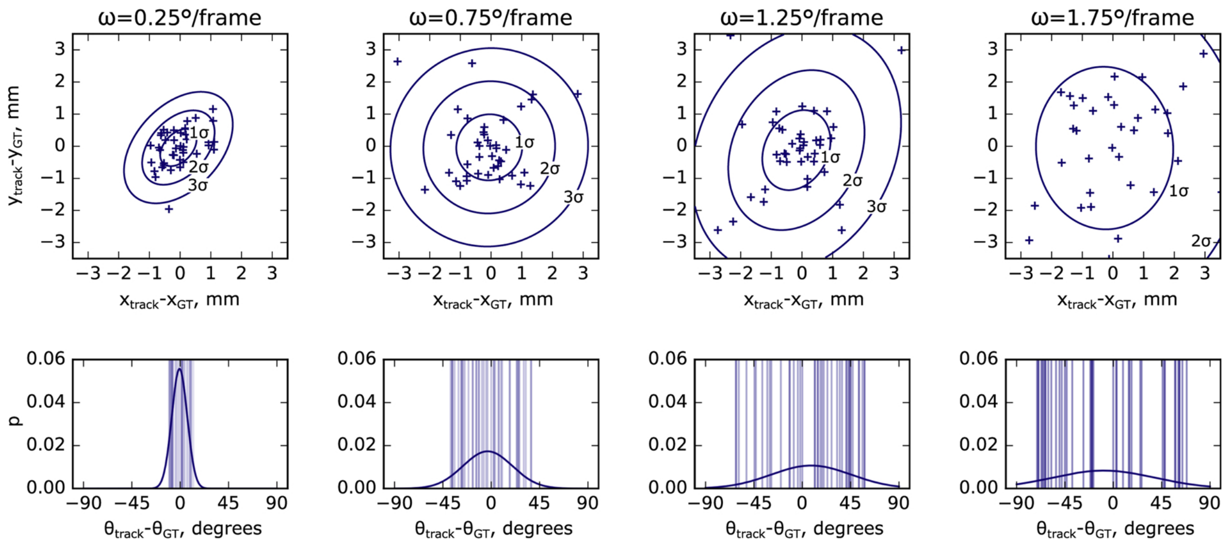


Fig. 5. Top row: Distance tracking error. i In the simulated ball rotation trajectories at different rotation speeds ω. For each ω, 40 trajectories were rendered, each as a result of a 1 s long (500 frames) rotation of the ball about an axis with a randomly selected orientation. The resulting endpoints of the trajectories on a virtual 2D plane is compared to the ground truth and the difference is shown as a cross mark. The resulting error distributions is fitted with a 2D Gaussian (top). Bottom row: Orientation tracking error. The difference in orientation at the endpoint of the trajectory in the 2D plane is taken between the ground truth and the measured trajectory; for each trajectory the difference is marked with a line, and the resulting distribution is fit with a Gaussian.

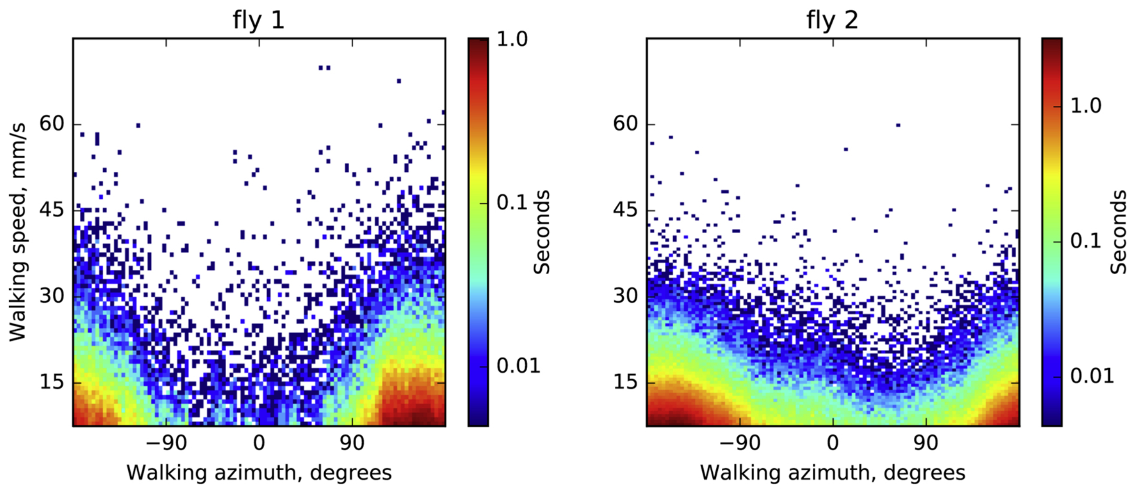
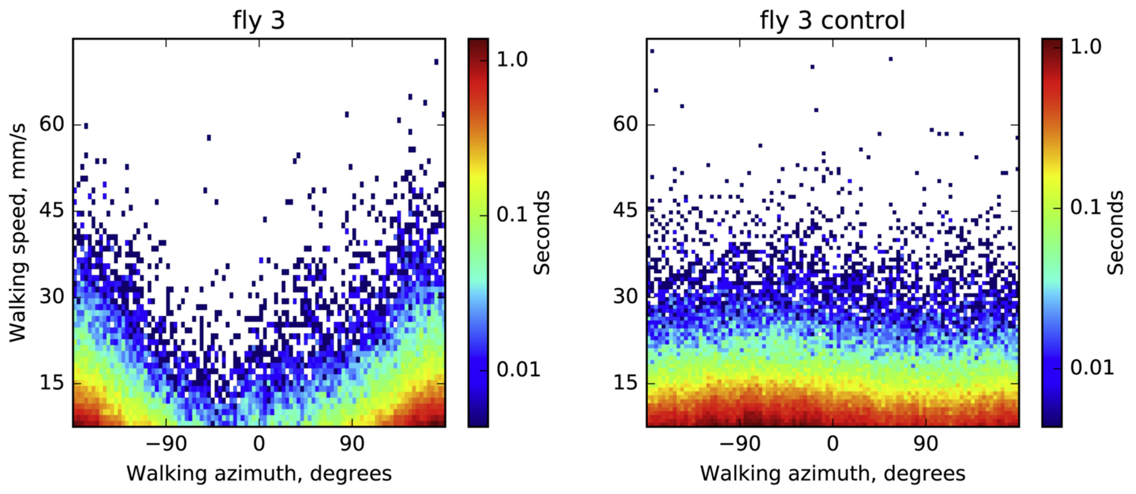



Fig. 6. Speed and direction of three flies walking in a closed loop with a bright stripe on a black screen (see text for details). Color indicates histogram bins (binned with a grid of 100 × 100 pixels) of time spent in each state for a total trial length between 15 and 25 min (depending on the fly)). In darkness the flies do not show a pronounced orientation preference (as shown for fly 3, control). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

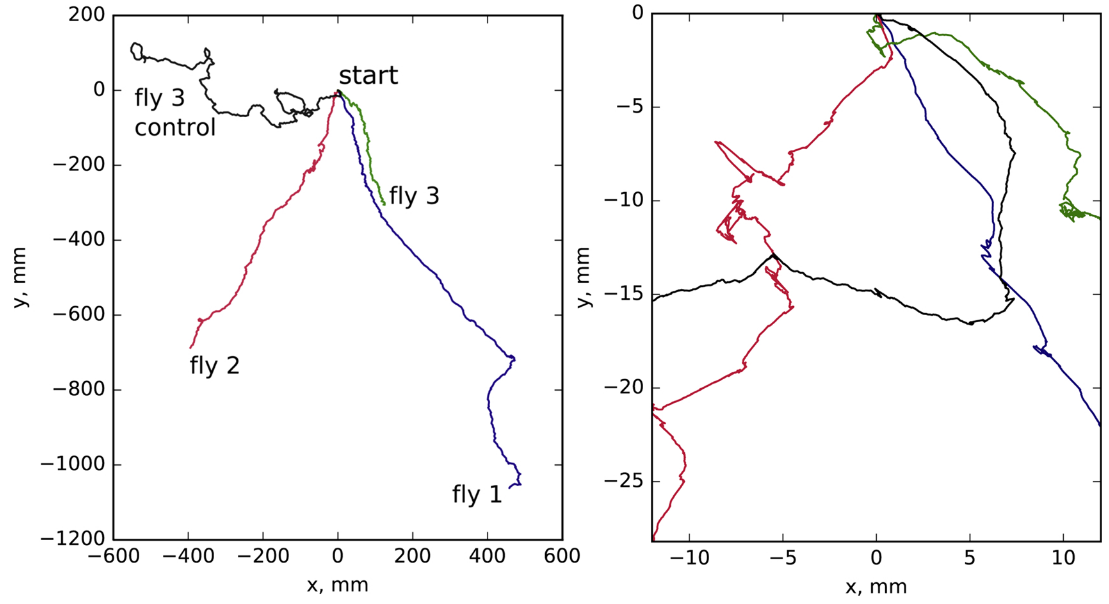


Fig. 7. Walking trajectories recorded for 5 min projected onto the plane (Fig. 6). The flies have a preferential walking direction in the negative *y* direction away from the stimulus, except for the control (waking in darkness). The left side shows the entire trajectories, the right side shows a close-up around the starting position.

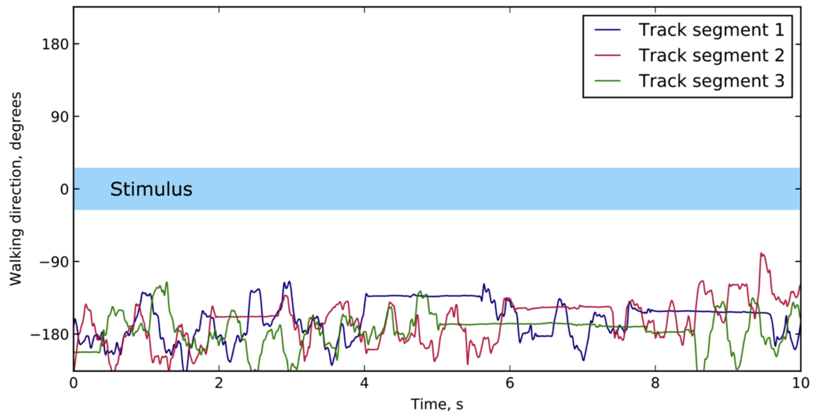


Fig. 8. Fly walking direction over time, in a closed loop with a single bright stripe. The walking direction of 180° points away from the stimulus.

using UV glue to encourage walking behavior (Grabowska et al., 2018). The stripe was visible for 4/5th of the flies virtual surroundings and disappeared for 1/5th behind the fly. This resulted in robust orientation behavior with the fly walking away from the stripe (see Figs. 6–8).

5. Discussion

An optical flow-based method for tracking of spherical treadmill motion using a single high-resolution camera was developed and integrated with virtual reality projection. Treadmill motion was monitored at 500 Hz, the display update rate was 120 Hz, and the total latency for an entire VR cycle was 30 ± 8 ms (from reading out ball movement to projecting the updated VR frame on the display). The system combines the flexibility of optical flow-based tracking with the convenience of using a single high resolution camera, off-the-shelf components, and open source software.

Compared with an optical mouse based system the approach developed here offers independent control over the different components of the system, such as camera settings and optical flow processing algorithm. Image processing is therefore not limited to a preset and unknown algorithm, but a suitable approach can be selected and its parameters can be tuned (see Section A.7 in the Appendix for a comparison of all optical flow algorithms tested). Additionally, the resulting tracking accuracy and the operation range (which for all optical flow-based methods depends on the combination of field of view, movement speeds and processing algorithm) can be tested using simulated data. Different from approaches using optical mice, tracking of all three degrees of freedom of ball rotation is achieved with a single camera which can be positioned flexibly and can easily aligned and calibrated. This also facilitates adapting the system to a variety of experimental conditions and scales.

As an alternative to optical flow approaches, a solution relying on a unique patterning of the ball surface was implemented in Moore et al. (2014). While this pattern matching approach avoids integration errors, optical flow can on the other hand track any surface with sufficient speckle contrast under a variety of lighting conditions. For small balls, such as the ones used for *Drosophila* behavior, this has the advantage that high contrast and unique ball patterning that could interfere with visual stimulation is not required. The method can easily be adapted to different ball sizes but requires sufficient optical contrast of the ball surface, such as a random patterning as used in the motor actuated calibration experiments.

Compared to cameras, optical mouse sensors typically have a higher frame rate (at the cost of lower pixel resolution) and time jitter and latencies are shorter on a dedicated processing chip. For example, Lott et al. (2007) measured the temporal accuracy of a customized optical mouse sensor and found a tracking latency of less than 500 µs, compared to 2 ms here. However, as the latency time breakdown shows (see Table 2), the delay is mostly caused by the standard display interface (DisplayPort). Since the camera tracking latencies and jitter are small compared to the display latencies and jitter, this should have a very limited impact on VR behavior. The overall latency of the closed-loop virtual reality system was 30 ± 8 ms on average, comparable to optical mice-based solutions and about three times faster than another camera-based pattern matching approach (Moore et al., 2014). The display update rate of typical virtual reality setups with data transfer through the display port is generally limited to 60 to 120 Hz. Typical latencies found with such a PC and camera-based solutions are between 50-90 ms (Stowers et al., 2017; Moore et al., 2014) and have been shown to be acceptable for closed-loop behavior experiments with a variety of species (Dombeck and Reiser, 2012; Stowers et al., 2017). We tested the system for behavioral performance in fruit flies and found a robust avoidance response of a bright stripe under the display conditions tested.

Table 2
Latency time breakdown for the closed-loop virtual reality setup.

Component	Time, ms
Camera exposure	0.1
Sensor readout	1.2
Transfer to PC	< 2
Tracking algorithm	2
VR rendering + Transfer to DMD Over DisplayPort + display latency	25 ± 8
Total	30 ± 8

Overall, a VR system based on optical flow measurements with a high-resolution camera was developed with short latencies suitable for VR experiments in fruit flies and easily adaptable to different scales.

Acknowledgements

We would like to thank Santosh Thoduka for suggestions on camera calibration and helpful discussions, Fiona Ross and Stephanie Miceli for initial behavioral tests, Andres Flores for help with behavior analysis, setting up the behavior setup shown in Figure 19, and designing hardware for stimulus projection, Bernd Scheiding for electronics support, and Dan Turner-Evans for suggesting the components for IR illumination. This work was supported by the Max Planck Society and caesar.

Appendix A

A.1 Modeling of optical flow camera projections

In this section a model is developed that describes how optical flow generated by ball rotations around different axes is projected onto the camera sensor. Predictions of this model will be used as fit functions to extract ball rotation parameters from measured optical flow distributions.

Ball motion is described by specifying an axis of rotation and an angular velocity (axis-angle representation (Palais and Palais, 2007)). Assuming a Cartesian coordinate system O with the origin at the center of the ball and a corresponding ISO-conventional (Weisstein, 2005) spherical coordinate system S as shown in Fig. 9, a point p on the surface of the ball can be described with a vector $\mathbf{p}_S = (r_{ball}, \theta_p, \varphi_p)$. Here, θ_p and φ_p are the polar and azimuth angle of a point on the surface of the ball and r_{ball} is the constant radius. If O is selected so that its z-axis coincides with the axis of ball rotation, it becomes a convenient parametrization since only the azimuthal angle is varying with time if the ball rotates around the polar axis with angular velocity ω:

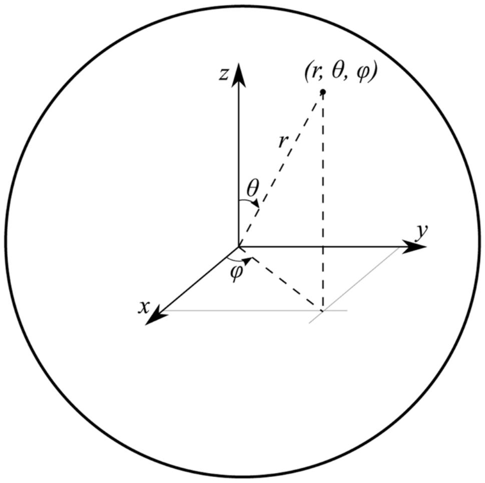


Fig. 9. Spherical coordinate system complying with the ISO 31-11 convention: a point is addressed by radius r, polar angle θ and azimuthal angle φ.

$$\mathbf{p}_S(t) = \begin{bmatrix} r_{\text{ball}} \\ \theta_p \\ \varphi_p + \omega t \end{bmatrix}. \tag{2}$$

In the Cartesian coordinate system O the same trajectory is expressed as

$$\mathbf{p}_O(t) = \begin{bmatrix} r_{\text{ball}} \sin(\theta_p)\cos(\varphi_p + \omega t) \\ r_{\text{ball}} \sin(\theta_p)\sin(\varphi_p + \omega t) \\ r_{\text{ball}} \cos(\theta_p) \end{bmatrix}. \tag{3}$$

Using a pinhole camera model (Sturm, 2014) we can express the projection of points on the ball surface onto the camera sensor. We here follow the same naming and frame orientation convention as the OpenCV library (Bradski, 2000, 2016), shown in Fig. 10. The trajectory of a point on the ball surface in the camera frame C is

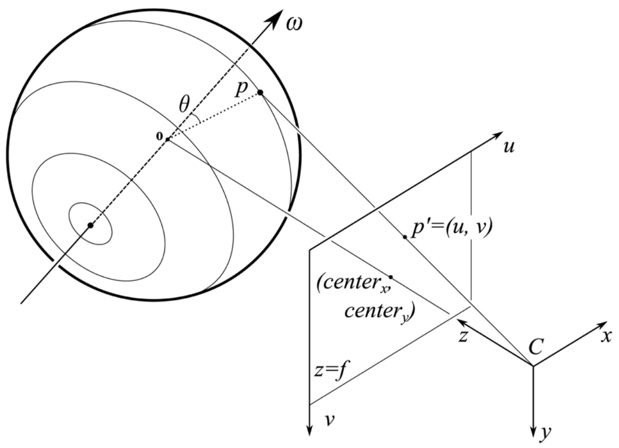


Fig. 10. Projection of points on the ball with a pinhole camera model. Point p on the ball surface can be described with coordinates (r, θ, φ) in a polar coordinate system aligned with the ball's instant axis of rotation. According to expression (4) it can be expressed in Cartesian coordinates (x, y, z) in the cameras reference frame C and projected onto the point $p' = (u, v)$ in the image plane according to Eq. (5).

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = R \begin{bmatrix} r_{\text{ball}} \sin(\theta_p)\cos(\varphi_p + \omega t) \\ r_{\text{ball}} \sin(\theta_p)\sin(\varphi_p + \omega t) \\ r_{\text{ball}} \cos(\theta_p) \end{bmatrix} + T, \tag{4}$$

where R is the rotation matrix representing the orientation of the ball's axis of rotation and T is the position of the ball's reference frame origin in the camera frame C. The image plane coordinates (u, v) of the projected point p' are expressed as

$$\begin{aligned} u &= f_x \frac{x_c}{z_c} + \text{center}_x \\ v &= f_y \frac{y_c}{z_c} + \text{center}_y \end{aligned} \tag{5}$$

where f_x and f_y are the focal lengths expressed in pixel units and $(center_x, center_y)$ is the image center. The ball is centered in the camera's field of view and therefore $T = [0, 0, d]$, where d is the distance between the center of the ball and the camera aperture.

By calculating the ball point projection between two consecutive frames at time t and $t + \Delta t$, with Δt equal to the frame period, one can calculate the displacement of the point projection and estimate optical flow. If all camera parameters as well as ball location and ball size are known, the model can be used to calculate the optical flow distribution on the camera. The model was implemented using the symbolic math library Sympy.

The angular velocity vector can be expressed as a sum of three orthogonal angular velocity components. For a camera-centered ball, it is convenient to choose the axes of a frame aligned with the camera frame with its origin at the ball's center. This gives three distinct rotations which can be registered by the camera as x-, y- and z-rotations (see Fig. 10).

Filling in the matrix R in expression (4) to obtain ball rotations around the x, y, and z camera frame axes, respectively, and combining it with (5) yields the trajectories of the ball's point projections on the camera (for rotation around the specified axis):

$$x: \begin{cases} u = \text{center}_x - \dfrac{f_x r_{\text{ball}} \cos(\theta_p)}{d + r_{\text{ball}} \sin(\theta_p)\cos(\omega t + \varphi_p)} \\ v = \text{center}_y + \dfrac{f_y r_{\text{ball}} \sin(\theta_p)\cos(\omega t + \varphi_p)}{d + r_{\text{ball}} \sin(\theta_p)\cos(\omega t + \varphi_p)} \end{cases} \tag{6}$$

$$y: \begin{cases} u = \text{center}_x + \dfrac{f_x r_{\text{ball}} \sin(\theta_p)\cos(\omega t + \varphi_p)}{d - r_{\text{ball}} \sin(\theta_p)\sin(\omega t + \varphi_p)} \\ v = \text{center}_y + \dfrac{f_y r_{\text{ball}} \sin(\theta_p)\cos(\omega t + \varphi_p)}{d - r_{\text{ball}} \sin(\theta_p)\sin(\omega t + \varphi_p)} \end{cases} \tag{7}$$

$$z: \begin{cases} u = \text{center}_x + \dfrac{f_x r_{\text{ball}} \sin(\theta_p)\cos(\omega t + \varphi_p)}{d + r_{\text{ball}} \cos(\theta_p)} \\ v = \text{center}_y + \dfrac{f_y r_{\text{ball}} \sin(\theta_p)\cos(\omega t + \varphi_p)}{d + r_{\text{ball}} \cos(\theta_p)} \end{cases} \tag{8}$$

A.2 Tracking settings

Tracking-specific settings are stored in the `Config\ tracking.cfg` file and include the path to the camera configuration file, the name of the camera to be used for tracking, as well as calibration coefficients as explained in Section A.1. Those parameters can be found in the corresponding sections of the config file:

```
[camera settings]
LoadCameraSettings=true
CameraSettingsFile=Config\camera_settings.pfs
TrackingCameraName=ball_cam
[tracking settings]
BallCenterX=112.0
BallCenterY=70.0
BallRadius=116.0
CxyRad=100.31
CxyTan=76.85
Cz=20.63
```

The camera was set to capture images at 500 fps, and the exposure time was set to 100 µs to prevent motion blurring (and required using sufficient infrared intensity). The resolution was set to 224 × 140, the highest resolution that was handled by our system in the available 2 ms per frame processing time. The settings of the camera can be saved in the Pylon configuration file and set to be loaded in the tracking configuration.

A.3 DMD settings

In order to use the LightCrafter 6500 DMDs at a 120 Hz display rate, they need to be connected through the DisplayPort interface and have to be configured to use Dual pixel clock mode, run in Video Pattern mode: this is an operation mode where each frame arriving through the video interface (RGB 8bpp) is regarded as 24 independent bit planes and each can be independently selected to be displayed with an arbitrary timing diagram. In this mode the most significant bit of the green channel is taken and displayed for 7 ms, resulting in 120 fps display rate. Additionally, it might be necessary to enable 120 Hz refresh rate in the Windows display adapter settings for both DMDs, which are visible to the operating system as normal displays.

The most convenient way to automate the DMD configuration is by saving these settings in a batch file (sequence of commands sent to the DMD) and upload it to the DMD as a startup script (this is a feature supported by LightCrafter 6500).

A.4 Virtual environment settings

Virtual reality has several configuration parameters which are specified in the `Config\vr.cfg`. It has game-engine specific parameters in the corresponding section, like the position of the VR window in screen coordinates and its size. Since we used two DMDs, it is spanned over twice the resolution of the DMD in width and located to the right of the main screen. `VSync` is disabled, since it is otherwise introduces additional latency to the display. `minFps` is set equal to the refresh rate of the displays; `maxFps` and `maxInactiveFps` are capping the update rate of the game engine. `maxInactiveFps` is the frame rate limit for a window while it is not in focus, i.e. when the user is working with other applications. This is almost always the case, since the game window is not visible for the computer user and other applications, for example for monitoring the VR, are used. The default value is 60 fps, and therefore needs to be increased in order to use the full capacity of the 120 Hz DMD displays.

The “transforms” section of the configuration defines how the ball rotation maps to motion in the virtual environment. The tracking signal is calibrated so that it presents angular displacements of the ball (in radians) about the *x*-, *y*-, and *z*-axes of the camera. Depending on how the camera is oriented relative to forward walking direction of the animal, the mapping varies. `ballXYZtoArenaXYZ` is a 3×3 rotation matrix, and

arena_displacement = ballXYZtoArenaXYZ·ball_displacement.

Similarly, the turning in the virtual environment is calculated from the ball displacement as a dot product with `ballXYZtoArenaYaw`:

arena_yaw_displacement = ballXYZtoArenaYaw·ball_displacement.

In the provided example the tracking camera is placed behind the ball and is aligned with the forward walking direction of the fly:

```
[transforms]
ballXYZtoArenaXYZ=0 0 -3.0 0 0 0 3.0 0 0
ballXYZtoArenaYaw=0 -57.3248 0
```

The 3 mm radius of the ball is incorporated in the motion transform, and the yaw rotation contains a conversion from radians to degrees.

A.5 Arena scripts

Although the Urho3D game engine allows saving and loading 3D environments in XML format, a more readable and convenient way is to declare environment properties in a script. Urho3D supports a compiled scripting language called AngelScript (Jönsson and Contributors, 2018), which is object-oriented, and features syntax similar to C+ +. The scene is represented in the game engine as a hierarchical tree structure: each entity of the 3D environment is a node that can be attached to either other nodes or to the root node of the scene. The coordinates of the nodes are always interpreted in the coordinate frame of their parent nodes.

In-detail information is provided in the game engine's manual (Urho3D contributors, 2019b), and sample arenas are documented. Dithering, which is required for displaying grayscale images with a DMD operating in binary display mode is done with a postprocessing shader, which is added to the rendering pipeline of the game engine. The configuration of the VR display is also shown in a documented example script.

A.6 Ground truth datasets

The datasets for evaluation of the tracking accuracy and for calibration were created using two different methods. On the one hand we used simulations, where a 3D scene was reconstructed with Blender 3D including a camera with focal parameters identical to the actual tracking setup, a light source and a ball with grainy surface texture. The model of the ball was rotated about a selected axis with a set angular velocity and rendered using Blender's API and in-built Python interpreter. This resulted in footage of the rotating 3D model of the ball with known rotation parameters. This data allowed to evaluate both tracking errors in magnitude ε_{magn} and in rotation axis orientation ε_{orient}.

On the other hand we used a stepper motor to rotate a ball. Unlike in simulations, it is difficult to arbitrarily control the axis of rotation, so that the axis of rotation was set at the beginning and was constant during the experiment.

Using simulation, several rotation sequences were generated:

- Rotations with constant angular velocity of 1° per frame (corresponding to about half of the maximal walking speed of the fly) with 30 different rotation axes, 100 frames for each axis;
- 30 rotations about various axes, but with exponentially increasing angular velocity from 0 to 2° per frame;
- 40 1-s long (500 frames) trajectories of a ball spinning about a random axis of rotation for each of the 4 selected angular velocities of 0.25, 0.75, 1.25 and 1.75° per frame.

With a stepper motor, controlled using Grbl firmware for Arduino boards (Simen Svale Skogsrud, 2009), following footage was recorded:

- Sequences of 1000 frames of rotation at 0.281, 0.562, 1.124 and 1.686° per frame, in two orientations corresponding to *y*- and *z*-rotations of a 6 mm polyurethane ball (same as used as a treadmill for the fly);
- Identical sequences of rotations of a 60 mm Styrofoam ball (scale model);
- Rotations of a 400 mm Styrofoam ball about *y*-axis at 0.12, 0.23 and 0.47° per frame, 6000 frames each.

An actual tracking camera image is shown side-to-side with a simulated image in Figs. 11 and 12: the real image has out-of-focus areas, some bright speckles from direct reflection of the lighting, and an overall uneven brightness distribution.

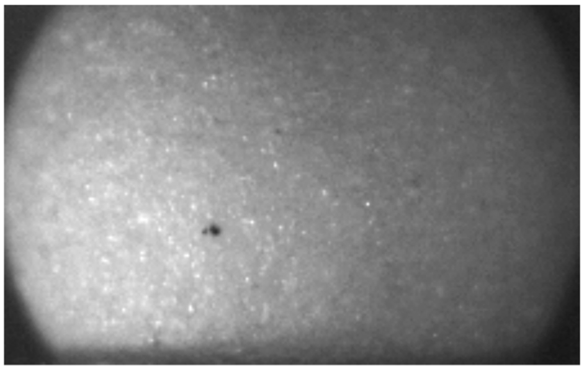


Fig. 11. Frame captured by tracking camera in VR setup.

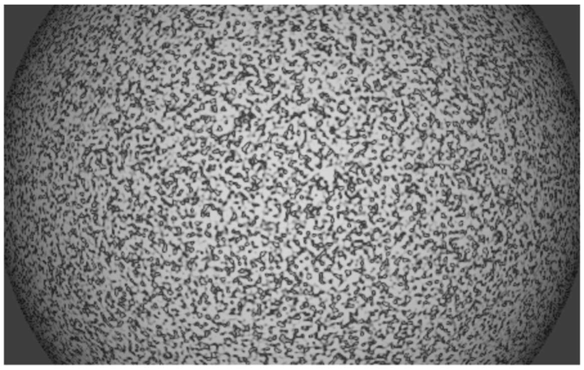


Fig. 12. Frame from simulated ball rotation sequence.

A.7 Optical flow algorithms

Algorithms for computing optical flow vary significantly in computational complexity, run-time and resulting accuracy. The OpenCV library features several implementations of the optical flow estimation algorithms (as of version 3.4.1) and the algorithms tested here were: Lucas-Kanade sparse feature tracking (Bouguet, 2001), Gunnar Farneback's algorithm of dense optical flow (Farnebäck, 2003), the optical flow algorithm by Brox et al. (2004), Dual TV-L$_1$ (Zach et al., 2007), Dense Inverse Search (DIS) (Kroeger et al., 2016), SimpleFlow (Tao et al., 2012), DeepFlow (Weinzaepfel et al., 2013) and PCAFlow (Wulff and Black, 2015) (see Table 3).

The speedup through reducing the size of the ROI varies depending on the algorithm, which motivated a comparison of the run-times depending on the input size (Fig. 14). To achieve a tracking frequency of 500 frames per second, the run-time should not exceed 2 ms, preferably be less to accommodate the frame preprocessing and ball motion estimation calculations. The algorithms relying on sparse features pre-selection fail in smaller ROIs (Sparse-to-dense, PCAFlow). Farneback, although being slow with default parameters, could be sped up significantly by reducing the number of scale levels and by limiting the internal iterations. It turned out to be the algorithm most suitable for the application, since it can handle a ROI of up to 60 × 140 px in under 2 ms when tuned for speed (Fig. 13).

A.8 Selecting ROI parameters

The ROI is a small part of the full frame used to identify the rotation direction of the ball from optical flow. The ROI is limited by two concentric rings with their centers coinciding with the center of the frame, which is also selected to agree with the center of the tracked ball. By specifying the radius of the ROI ρ_{ROI} (px) and its width $\Delta\rho_{ROI}$ (px), one can vary the run-times of the fitting algorithm and its accuracy. The ROI parameters are chosen by running the tracking algorithm on simulated datasets (constant 1° per frame angular velocity) and finding the variance in the estimated angular velocity magnitude and orientation compared to ground truth. Fig. 15 shows the summed magnitude and orientation error (normalized) depending on ρ_{ROI}, using $\Delta\rho_{ROI} = 10$ px. The lowest value was achieved for $\rho_{ROI} = 0.27$, $\rho_{max} = 60$ px.

A.9 Calibration coefficients

The coefficients $c_{xy\ rad}$, $c_{xy\ tan}$ and c_z in expression (1) are required for relating the optical flow in pixels (as determined with the optical flow algorithm) to the actual ball displacement in radians. Apart from a unit conversion factor these coefficients depend on the camera magnification, the size of the ball, and the position of the ROI on the ball. These factors can be determined in any of the following ways: by correlating optical flow induced by known ball rotations (with a ground truth dataset); by using modelled optical flow with the pinhole camera model as in expressions ((6)–(8), provided the optical parameters of the camera are known); or using a second synchronized camera. These methods are described in more detail in the following subsections.

A.9.1 Calibration with ground truth dataset

The optical flow in the tracking ROI was calculated on simulated data and its distributions are fitted with function (1). Then the calibration

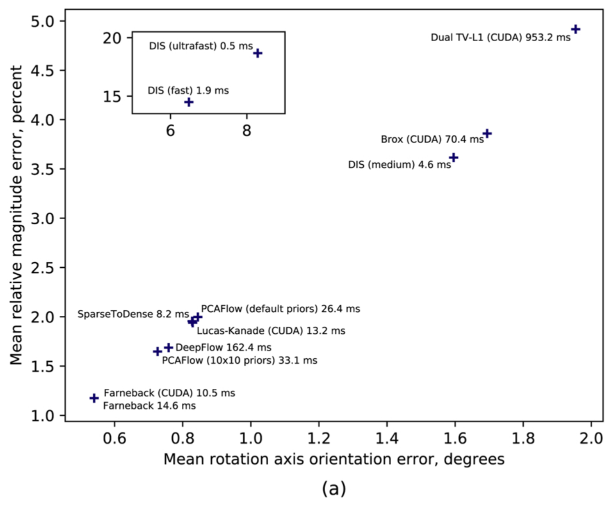
Table 3
Tracking performance on the simulated rotation dataset, using full frame optical flow. Error 1 is $\varepsilon_{magn\_r\ elative}$ as defined in Section 4.2 on accuracy evaluation, Error 2 is the mean absolute value of $\varepsilon_{magn\_r\ elative}$, Error 3 is ε_{orient}.

Algorithm	Error 1, % ± SD	Error 2, %	Error 3, degrees ± SD	Time, ms
Dual TV-L1 (CUDA)	0.2 ± 5.3	2.8	2.0 ± 1.6	953.2 ± 382.4
DeepFlow	−0.5 ± 1.0	0.9	0.8 ± 0.7	162.4 ± 8.4
Brox (CUDA)	−0.1 ± 2.5	2.1	1.7 ± 1.2	70.4 ± 12.8
PCAFlow (10x10 priors)	−0.6 ± 0.8	0.9	0.7 ± 0.5	33.1 ± 27.7
PCAFlow (default priors)	−0.5 ± 1.4	1.2	0.8 ± 0.8	26.4 ± 17.4
Farneback	−0.2 ± 0.7	0.6	0.5 ± 0.4	14.6 ± 4.4
Lucas-Kanade (CUDA)	0.1 ± 2.4	1.9	0.8 ± 0.9	13.2 ± 14.1
Farneback (CUDA)	−0.2 ± 0.7	0.6	0.5 ± 0.4	10.5 ± 1.4
SparseToDense	−0.5 ± 1.3	1.2	0.8 ± 0.8	8.2 ± 1.2
DIS (medium)	0.5 ± 2.6	1.9	1.6 ± 1.1	4.6 ± 0.7
DIS (fast)	2.9 ± 8.6	7.1	6.5 ± 4.3	1.9 ± 0.3
DIS (ultrafast)	4.6 ± 11.0	9.2	8.3 ± 5.9	0.5 ± 0.2

constants can be found as the ratios:

$$\begin{aligned} c_z &= \frac{\omega_{z\ \mathrm{GroundTruth}}}{\omega_{z\ \mathrm{Fitted}}} \\ c_{xy\ \mathrm{rad}} &= \frac{\omega_{xy\ \mathrm{GroundTruth}}}{\omega_{xy\ \mathrm{rad\ Fitted}}} \\ c_{xy\ \mathrm{tan}} &= \frac{\omega_{xy\ \mathrm{GroundTruth}}}{\omega_{xy\ \mathrm{tan\ Fitted}}}, \end{aligned} \tag{9}$$

where $\omega_{xy\ GroundTruth}$, $\omega_{z\ GroundTruth}$ are the known rotations, and $\omega_{xy\ rad\ Fitted}$, $\omega_{xy\ tan\ Fitted}$ are the results of fitting radial and tangential optical flow



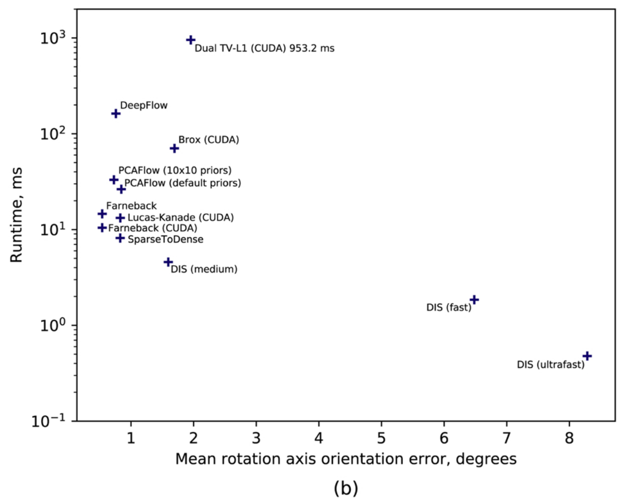


Fig. 13. Ball rotation estimation errors (a) and run-time of the optical flow methods on full frame of the simulated dataset (b).

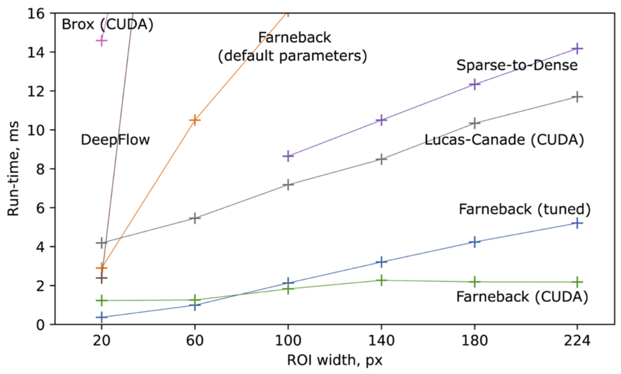


Fig. 14. Run-time of optical flow algorithms depending on input size (ROI width ×140 px). Measured as average over 3000 frames with simulated dataset with accelerating rotations.

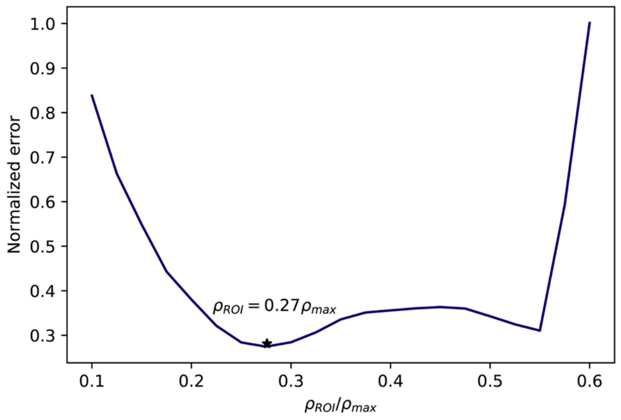


Fig. 15. Summed tracking orientation and rotation errors (normalized), obtained by running the tracking algorithm with $\Delta\rho_{ROI} = 10$ px. ρ_{max} is the visual radius of the ball (116 px).

Table 4
Measured parameters of the tracking setup scale model.

Parameter	Value
Distance to the ball	1400 mm
Ball radius	30 mm
Image resolution	224 × 140 px
Ball image radius	116 px

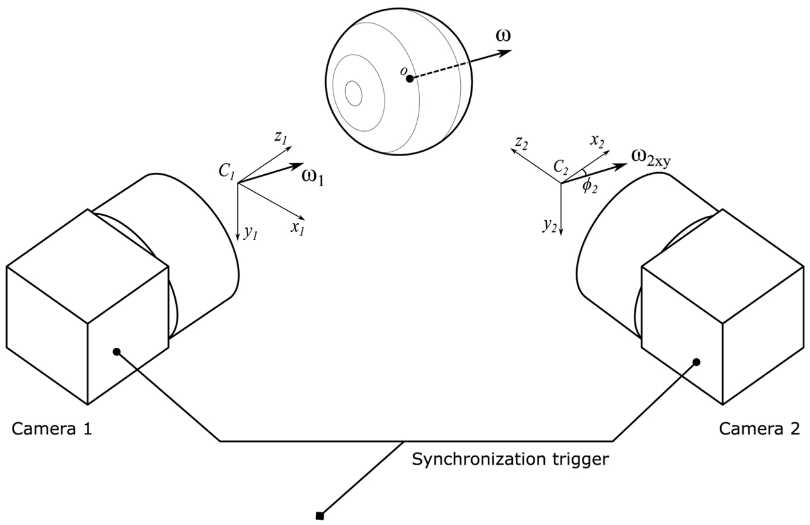


Fig. 16. Calibration procedure using two cameras. The *z*-component of angular velocity of the ball in the reference frame of camera 1 is projected onto the *x*-axis in the reference frame of camera 2. Calibration is performed by correlating those components to find c_{xy}.

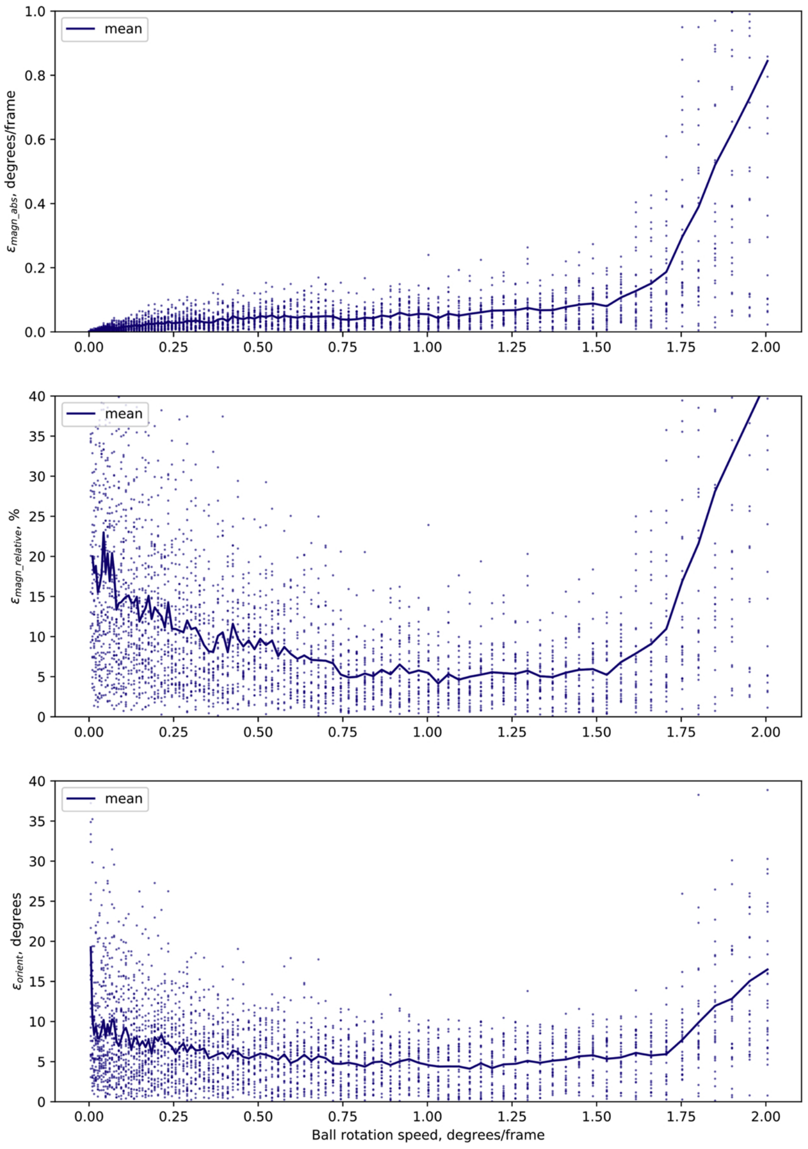


Fig. 17. Tracking errors for measured on simulated accelerated rotations dataset. Each point represents the error for a rotation at the corresponding angular velocity. The mean line shows average error between rotations about all axes of rotation of the dataset.

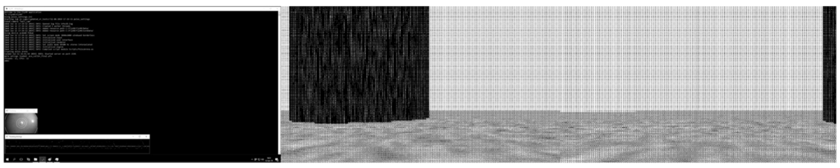


Fig. 18. Screenshot of the virtual reality application during an experiment. The left side is displayed on the monitor visible to the user, showing the status of the application; the right side is displayed on two projection screens seen by the fly.

distributions in the ROI assuming $c_{xy\ rad} = c_{xy\ tan} = c_z = 1.0$.

Using the rendered ball rotation dataset with constant rotations, the following calibration factors were found:

$$c_{\text{xy rad}} = 100.31\,\text{px/rad} = 1.75\text{px/}^\circ,$$

$$c_{\text{xy tan}} = 76.85\,\text{px/rad} = 1.34\text{px/}^\circ,$$

$$c_z = 20.63\,\text{px/rad} = 0.36\text{px/}^\circ.$$

A.9.2 Calculation of the calibration factors from the scene model

Using the ball projection model (Eqs. (4) and (5)) with substituted camera and scene parameters, optical flow distribution in the tracking ROI can be estimated and used for calibration similar to the previous method. With the help of a scaled-up calibration setup, the camera and scene parameters were measured as listed in Table 4.

Using these measurements, and the pinhole camera model (5) the focal length of the camera can be calculated (assuming the image projection

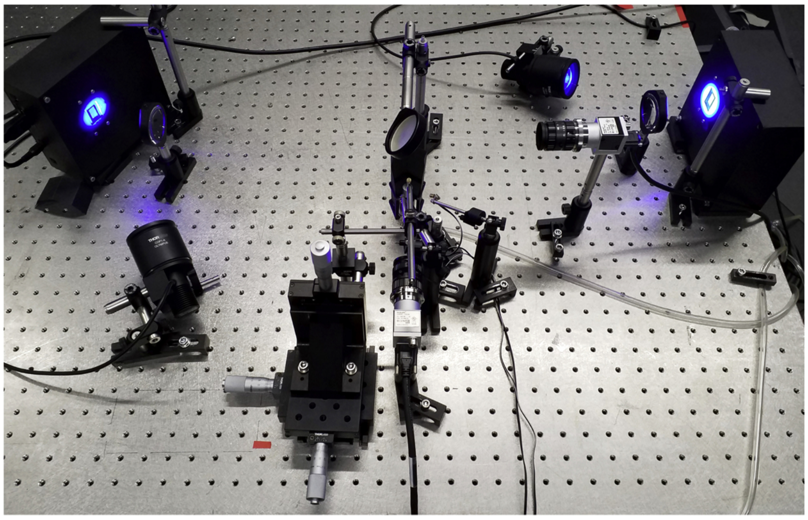


Fig. 19. Virtual reality setup corresponding to the schematic in Fig. 1. Two digital micromirror devices (DLP LightCrafter 6500, Texas Instruments) illuminated by collimated blue LEDs (M470L3, Thorlabs) are projecting rendered VR through achromatic lenses (Thorlabs) onto a 4-segment screen made from black paper. A 6 mm polyurethane foam ball is held in a cup-shaped holder and suspended in an air stream. It is illuminated with two IR LEDs (Thorlabs LED850LN – 850 nm LED, TO-39, and Mightex LED Driver, SLA series) and monitored with a camera (Basler acA640-750um). The fly holder is mounted onto a xyz-translation stage. An additional camera (Basler acA640-750um) provides the top view of the fly through the mirror for alignment and monitoring. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 5
Tracking validation on a single run of stepper-motor actuated ball rotation. "Ground truth" is the actual ball displacement, "Measured" refers to the rotation determined from the video sequence using the tracking system. Tracking was done using calibration factors obtained through two-camera calibration procedure.

Rotation axis, speed	Ground truth, degrees	Measured, degrees	Error, %
60 mm styrofoam ball			
y-, 0.281°/frame	281	300	+6.8
y-, 0.562°/frame	562	593	+5.5
y-, 1.124°/frame	1124	1215	+8.1
y-, 1.686°/frame	1686	1778	+5.5
z-, 0.281°/frame	281	295	+5.0
z-, 0.562°/frame	562	564	+0.3
z-, 1.124°/frame	1124	1092	−2.8
z-, 1.686°/frame	1686	1610	−4.5
6 mm polyurethane ball			
y-, 0.281°/frame	281	307	+9.3
y-, 0.562°/frame	562	612	+8.9
y-, 1.124°/frame	1124	1217	+8.3
y-, 1.686°/frame	1686	1633	−3.1
z-, 0.281°/frame	281	275	−2.1
z-, 0.562°/frame	562	540	−3.9
z-, 1.124°/frame	1124	1068	−5.0
z-, 1.686°/frame	1686	1585	−6.0
400 mm Styrofoam ball			
y-, 0.117°/frame	702	713	+1.6
y-, 0.234°/frame	1404	1577	+12.3
y-, 0.473°/frame	2838	3166	+11.6

center is exactly in the middle of the frame, $center_x = 112$ px, $center_y = 70$ px):

$$116\,\text{px} = f\frac{30\,\text{mm}}{1400\,\text{mm}}$$

$$f = 116\,\text{px}\frac{1400\,\text{mm}}{30\,\text{mm}} = 5413\,\text{px}$$

$$f_x = f_y = 5410\,\text{px}$$

$$\text{center}_x = 112\,\text{px}$$

$$\text{center}_y = 70\,\text{px}$$

The model with these parameters was used to estimate the optical flow induced by different rotations according to expressions ((6)–(8)). The calibration factors found with this method are close to ones obtained with ground truth rotations:

$$c_{\text{xy rad}} = 95.37\,\text{px/rad} = 1.66\text{px/°},$$

$$c_{\text{xy tan}} = 70.31\,\text{px/rad} = 1.21\text{px/°},$$

$$c_z = 22.28\,\text{px/rad} = 0.39\text{px/°}.$$

A.9.3 Two-camera calibration

According to the mathematical model described above the z-component of the angular velocity measured by the tracking camera is detected as the xy-plane component by a second camera filming the same rotation from an orthogonal direction (see Fig. 16).

As shown using the optical flow model (Fig. 2), a rotation of the ball around the camera's z-axis results in all of its points moving along circular trajectories in the focal plane of the camera. This means that the angular displacement of the ball equals the angular displacement of the points in the frame along their trajectories, and the tangential optical flow in the ROI induced by this motion is proportional to the radius of the ROI, meaning that c_z can be estimated with a single camera without measuring any additionally parameters.

Different from z-factors, $c_{xy\ rad}$ and $c_{xy\ tan}$ depend on the camera properties and the scene geometry. Therefore, a second camera is introduced, filming the ball from an orthogonal direction, as shown in Fig. (16). The z-component measured by camera 1 is then equal to the x-component measured by camera 2, and camera 1 can therefore be used to calibrate xy-tracking by camera 2:

$$\begin{cases} c_{\mathrm{xy\ rad}} = \dfrac{c_{z\ \mathrm{estimated}}\,\omega_{z1}}{\omega_{\mathrm{xy\ rad2}}\cos\phi_2} \\ c_{\mathrm{xy\ tan}} = \dfrac{c_{z\ \mathrm{estimated}}\,\omega_{z1}}{\omega_{\mathrm{xy\ tan2}}\cos\phi_2}, \end{cases} \tag{10}$$

where $c_{z\ estimated}$ is estimated as described, ω_{z1} is measured by camera 1, $\omega_{xy\ rad2}$, $\omega_{xy\ tan2}$ and ϕ_2 are found by fitting function (1) on radial and tangential optical flow in the ROI from camera 2, respectively, initially assuming $c_{xy\ rad} = c_{xy\ tan} = 1.0$. ϕ_2 is the orientation of the ball's axis of rotation in the xy-plane of camera 2, and the cosine yields the x-axis projection of that rotation, corresponding to measured z-rotation by camera 1.

To perform this calibration, both cameras need to be set up with identical objectives and aligned at the same distance from the tracked ball at right angles (see Fig. 16); the ball is allowed to spin freely (at a speed within the range where the tracking algorithm is valid) with a stream of air while the cameras capture frames synchronized by an external triggering signal.

This calibration procedure was automated in a calibration script included with the application. It requires synchronized footage recorded by two cameras with two views of the ball from orthogonal directions. Running the script with our tracking configuration resulted in calibration constants very close to the ones calculated using the scene model with the same resolution and objective focal length:

$$c_{\mathrm{xy\ rad}} = 94.61\,\mathrm{px/rad} = 1.65\mathrm{px}/^\circ,$$

$$c_{\mathrm{xy\ tan}} = 70.86\,\mathrm{px/rad} = 1.23\mathrm{px}/^\circ,$$

$$c_z = 22.28\,\mathrm{px/rad} = 0.39\mathrm{px}/^\circ.$$

This calibration script has also successfully been used for 40 cm ball and different camera objectives.

A.10 Accuracy evaluation

Using these datasets, tracking accuracy was evaluated with the calibrated tracking algorithm. Simulated data allowed to evaluate both rotation magnitude and axis orientation errors (Fig. 17), as well as their effects on reconstruction of the fictive trajectory (Figs. 4 and 5).

The stepper-motor actuated ball rotations were used to evaluate the rotation magnitude errors integrated over the whole rotation sequence, and the results are listed in Table 5. They demonstrate that the tracking system can be scaled up to be used on different size treadmill balls as well as the validity of the 2-camera calibration procedure.

References

Bayer, B.E., 1973. An optimum method for two-level rendition of continuous tone pictures. In: June. IEEE International Conference on Communications, vol. 26.

Bouguet, J.-Y., 2001. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. Intel Corp. 5, 4.

Bradski, G., 2000. The OpenCV Library. Dr. Dobb's J. Softw. Tools.

Bradski, G., 2016. Camera Calibration and 3d Reconstruction. From OpenCV Reference Manual. (visited on 15.03.19). https://docs.opencv.org/3.2.0/d9/d0c/group_calib3d.html.

Brox, T., Bruhn, A., Papenberg, N., Weickert, J., 2004. High accuracy optical flow estimation based on a theory for warping. European Conference on Computer Vision. Springer, pp. 25–36.

Dombeck, D.A., Reiser, M.B., 2012. Real neuroscience in virtual worlds. Curr. Opin. Neurobiol. 22, 3–10.

Dombeck, D.A., Khabbaz, A.N., Collman, F., Adelman, T.L., Tank, D.W., 2007. Imaging large-scale neural activity with cellular resolution in awake, mobile mice. Neuron 56, 43–57.

Farnebäck, G., 2003. Two-frame motion estimation based on polynomial expansion. Scandinavian Conference on Image Analysis. Springer, pp. 363–370.

Grabowska, M.J., Steeves, J., Alpay, J., Van De Poll, M., Ertekin, D., van Swinderen, B., 2018. Innate visual preferences and behavioral flexibility in drosophila. J. Exp. Biol. 221, jeb185918.

Haberkern, H., Basnak, M.A., Ahanonu, B., Schauder, D., Cohen, J.D., Bolstad, M., Bruns, C., Jayaraman, V., 2018. On the Adaptive Behavior of Head-fixed Flies Navigating in Two-Dimensional, Visual Virtual Reality.

Jönsson, A., Contributors, 2018. Angelscript. (visited on 14.03.19). https://www.angelcode.com/angelscript/.

Kroeger, T., Timofte, R., Dai, D., Van Gool, L., 2016. Fast optical flow using dense inverse search. European Conference on Computer Vision. Springer, pp. 471–488.

Lott, G.K., Rosen, M.J., Hoy, R.R., 2007. An inexpensive sub-millisecond system for walking measurements of small animals based on optical computer mouse technology. J. Neurosci. Methods 161, 55–61.

Mason, A.C., Oshinsky, M.L., Hoy, R.R., 2001. Hyperacute directional hearing in a microscale auditory system. Nature 410, 686.

Moore, R.J., Taylor, G.J., Paulk, A.C., Pearson, T., van Swinderen, B., Srinivasan, M.V., 2014. FicTrac: a visual method for tracking spherical motion and generating fictive animal paths. J. Neurosci. Methods 225, 106–119.

Nelder, J.A., Mead, R., 1965. A simplex method for function minimization. Comput. J. 7, 308–313.

Palais, B., Palais, R., 2007. Euler's fixed point theorem: the axis of a rotation. J. Fixed Point Theory Appl. 2, 215–220.

Seelig, J.D., Chiappe, M.E., Lott, G.K., Dutta, A., Osborne, J.E., Reiser, M.B., Jayaraman, V., 2010. Two-photon calcium imaging from head-fixed *Drosophila* during optomotor walking behavior. Nat. Methods 7, 535–540.

Simen Svale Skogsrud, S.K.J., 2009. Grbl. https://github.com/grbl/grbl.

Stowers, J.R., Hofbauer, M., Bastien, R., Griessner, J., Higgins, P., Farooqui, S., Fischer, R.M., Nowikovsky, K., Haubensak, W., Couzin, I.D., et al., 2017. Virtual reality for freely moving animals. Nat. Methods 14, 995.

Sturm, P., 2014. Pinhole Camera Model. Computer Vision. Springer, pp. 610–613.

Tao, M., Bai, J., Kohli, P., Paris, S., 2012. Simpleflow: a non-iterative, sublinear optical flow algorithm. Computer Graphics Forum, vol. 31. Wiley Online Library, pp. 345–353.

Turner-Evans, D., Wegener, S., Rouault, H., Franconville, R., Wolff, T., Seelig, J.D., Druckmann, S., Jayaraman, V., 2017. Angular velocity integration in a fly heading circuit. eLife 6, e23496.

Urho3D contributors, 2019Da. Urho3d: A Cross-Platform 2d and 3d Game Engine. https://urho3d.github.io/.

Urho3D contributors, 2019Db. Urho3d Scene Model. (visited on 14.03.2019). https://urho3d.github.io/documentation/1.7/_scene_model.html.

Weinzaepfel, P., Revaud, J., Harchaoui, Z., Schmid, C., 2013. Deepflow: large displacement optical flow with deep matching. 2013 IEEE International Conference on Computer Vision (ICCV). IEEE, pp. 1385–1392.

Weisstein, E.W., 2005. Spherical Coordinates. From MathWorld-A Wolfram Web Resource. (visited on 20.03.18). http://mathworld.wolfram.com/SphericalCoordinates.html.

Wulff, J., Black, M.J., 2015. Efficient sparse-to-dense optical flow estimation using a learned basis and layers. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 120–130.

Zach, C., Pock, T., Bischof, H., 2007. A duality based approach for realtime tv-l 1 optical flow. Joint Pattern Recognition Symposium. Springer, pp. 214–223.