Journal of Neuroscience Methods 318 (2019) 69–77

Contents lists available at ScienceDirect

Journal of Neuroscience Methods

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

A novel low-noise movement tracking system with real-time analog output for closed-loop experiments

Nora Gaspar[a], Ronny Eichler[b], Eran Stark[a,*]

[a] Sagol School of Neuroscience and Department of Physiology and Pharmacology, Faculty of Medicine, Tel Aviv University, Tel Aviv, Israel
[b] Donders Institute for Brain, Cognition and Behavior, Radboud University Nijmegen, Nijmegen, the Netherlands

ARTICLE INFO

Keywords:
Closed-loop
Design for experiments
Electrophysiology
Image processing
Kalman filter
Open-source

ABSTRACT

Background: Modern electrophysiological experiments are moving towards closing the loop, where the extrinsic (behavioral) and intrinsic (neuronal) variables automatically affect stimulation parameters. Rodent experiments targeting spatial behavior require animal 2D kinematics to be continuously monitored in a reliable and accurate manner. Cameras provide a robust, flexible, and simple way to track kinematics on the fly. Indeed, several available camera-based systems yield high spatiotemporal resolution. However, the acquired kinematic data cannot be accessed with sufficient temporal resolution for precise real-time feedback.

New method: Here, we describe a novel software and hardware system for movement tracking based on color-markers with real-time low-noise output that works in both light and dark conditions. The analog outputs precisely represent 2D movement features including position, orientation, and their temporal derivatives, velocity and angular velocity.

Results: Using adaptive windowing, contour extraction, and rigid-body Kalman filtering, a 640-by-360 pixel frame is processed in 28 ms with less than 4 ms jitter, for 100 frames per second. The system is robust to outliers, has low noise, and maintains a smooth, accurate output even when one or more markers are temporarily missing. Using freely-moving mice, we demonstrate novel applications such as replacing conventional sensors in a behavioral arena and inducing novel place fields via closed-loop optogenetic stimulation.

Comparison with existing method(s): To the best of our knowledge, this is the first tracking system that yields analog output in real-time.

Conclusions: This modular system for closed-loop experiment tracking can be implemented by downloading an open-source software and assembling low-cost hardware circuity.

1. Introduction

Compared to manual administration of experiments, automated experimental designs have obvious advantages in terms of throughput, accuracy, reproducibility, experimenter effort, and robustness to human error. Furthermore, automated systems enable closed-loop experiments such as manipulating neuronal activity according to animal behavior (Wiener et al., 1989). In behavioral experiments, especially those targeting spatial memory and navigation, the most important extrinsic variables are the head location and orientation of the animal (Grieves et al., 2016; Moser et al., 2015; O'Keefe and Dostrovsky, 1971). Multiple approaches have been used to obtain these movement features including a grid of motion sensors (Opto-Varimex system; Columbus Instruments, USA) and piezoelectric sensors on the floor (Flores et al., 2007). Yet arguably the cheapest, most robust, and most widely-used method is recording with a single overhead camera. This approach can be used with or without markers attached to the subject (Maghsoudi et al., 2017; Mathis et al., 2018); clearly, markers increase target salience at the cost of limiting the number of targets and reducing freedom of movement. The camera-based technique has been used for over thirty years (Skaggs et al., 1998; Wiener et al., 1989), and multiple commercial (e.g. ANY-maze; CinePlex, Plexon; EthoVision, Spink et al., 2001; OptiTrack, NaturalPoint Inc. U.S; NetCom API, NeuraLynx, U.S.) and open source (e.g. Bonsai, Buccino et al., 2018; Lopes et al., 2015; DeepLabCut, Mathis et al., 2018; MouseMove, Samson et al., 2015; Pyper) systems are available. These tools are useful for behavioral logging and offline analyses but typically do not support any online functionality. Those that do, either enable only limited real-time output (collision of subject position with a given region-of-interest) or are tailored to a specific platform. In sum, presently-available tracking tools

* Corresponding author.
E-mail address: eranstark@tauex.tau.ac.il (E. Stark).

https://doi.org/10.1016/j.jneumeth.2018.12.016
Received 17 November 2018; Received in revised form 16 December 2018; Accepted 20 December 2018
Available online 14 January 2019

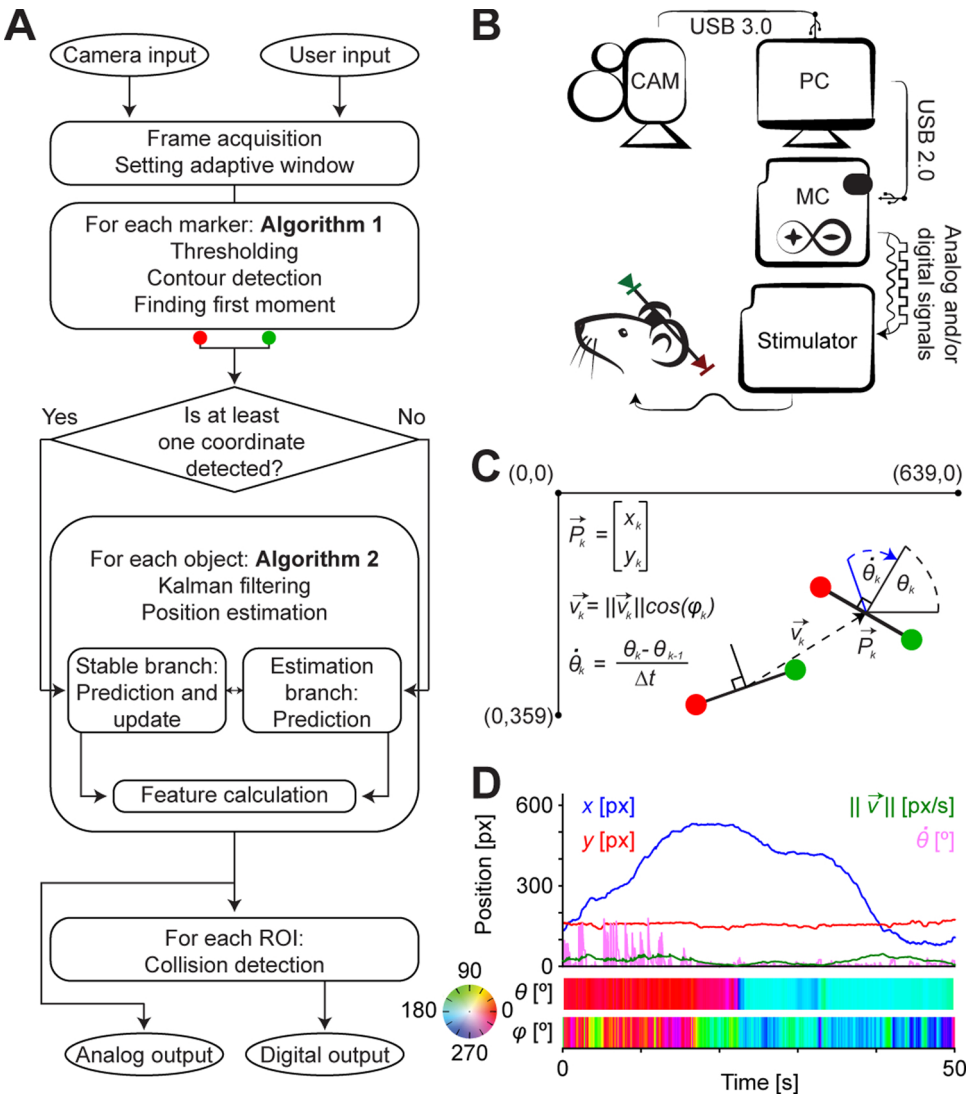

Fig. 1. System design. (A) Processing flowchart from input (camera) to output. The tracked subject is defined as an Object consisting of Marker/s. Object kinematic features are processed and forwarded to a microcontroller where they are converted to analog and digital signals. (**B**) Block diagram of the system. (**C**) Image coordinate frame and definitions of the six kinematic features. Origin is at the top left corner. $\vec{P}_k$ represents 2D object location. The angle (ϕ_k) of the velocity vector ($\vec{v}_k$) is movement direction, and $||\vec{v}_k||$ is speed. Orientation (θ_k) is the angle of the normal to the line connecting two markers, and angular velocity ($\dot{\theta}_k$) is its time derivative. (**D**) System output (six kinematic features) during a 50 s recording of a mouse running on a linear track.

are suboptimal for low-latency closed-loop experiments that rely on detailed kinematics such as orientation, velocity, or combinations thereof.

In the present work, we developed a marker-based system (Fig. 1A) that provides accurate high-resolution (100 samples/s, 4.5 mm/pixel) real-time (28 ± 3 ms delay) feedback of animal position, orientation, and their temporal derivatives (velocity and angular velocity). These features are conveyed as real-time analog (0–5 V with 12-bit resolution) signals that can be easily integrated with other variables such as neuronal recordings and be used for closed-loop manipulations such as electrical or optogenetic stimulation. Furthermore, the system outputs digital signals to indicate whether the animal is within user-defined regions of interest. Implementing the system is simple, and involves downloading an open-source software and assembling low-cost hardware circuity.

The system (Fig. 1B) for tracking a subject (e.g. an animal) consists of three blocks: (1) a camera; (2) a PC that runs the software ("Spotter") and controls the downstream hardware system; and (3) custom hardware ("Movement Controller", MC), outputting low-noise digital and analog signals. In the implementation described here, a Digital Signal Processor (DSP; RX8, Tucker-Davis Technologies) and a precision Current Source (CS; Stark et al., 2012) are used for closing the loop by applying intra-cortical illumination, resulting in neuronal activation of the tracked rodent. In this use case, the rodent has an implanted head-stage with two color-markers (brightly-painted blobs/LEDs). While the system works well in both light and dark conditions, each marker must have a color that is clearly distinct from the background and from other markers. The software is a Python-based application that uses a modular structure consisting of two main parts: a command line application, and an optional graphical user interface (GUI).

To provide modularity, three distinct levels of tracking are defined: Markers, Objects, and Regions of Interest (ROIs). *Markers* are the elementary tracking units, representing a color blob or an empty contour such as an LED. Markers are defined by a set of four parameters (hue, saturation, value, and size) that are used by the detection algorithm (Algorithm 1). An *Object* is composed of linked Markers and has up to

six features that can be routed to analog outputs: (1) x position; (2) y position; (3) orientation ; (4) speed $||\vec{v}||$; (5) movement direction ϕ; and (6) angular velocity $\dot{\theta}$. The first three features are first-order in the sense that they can be determined from a single frame, whereas the last three are second-order, based on the temporal derivative of multi-frame data (Fig. 1C). Note that θ and $\dot{\theta}$ are defined only for multi-marker objects.

Although the detection algorithm typically results in a highly accurate single-frame color-blob detection rate, continuity is not imposed, and thus multi-frame tracking is typically erratic. To account for these and other sources of noise, we designed an adaptive denoising and location estimation algorithm. The procedure (Algorithm 2) is based on the Kalman filter (Kalman, 1960), a real-time data fusion procedure that combines noisy measurements with estimates, resulting in smoother and more accurate output.

The last level of modularity is a *Region of Interest (ROI)*, which is a part of the image described by one or more geometric shapes (circles, lines, and/or squares). Linking an ROI to an Object will continuously check for collision between Object location and the ROI. The real-time state of this check can be emitted as a binary (digital) output.

2. Results

The system can track in real-time (28 ± 3 ms delay) the simultaneous movement of up to four Objects (e.g. behaving animals) and maintain up to sixteen ROIs (see Application I below). With a camera, the software can work as a stand-alone logger that may record (and later play back) 2D kinematics. Integration with the MC yields real-time analog and digital outputs available for data acquisition (DAQ) and/or processing on the fly. The generic MC design (Fig. 7) allows scaling the number of tracked features routed to analog outputs in real-time. All elements are open source, and available online (https://github.com/gasparnori/Spotter) under a *Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International* license.

2.1. Camera and computer

The system can work with any USB camera. Specifically, we tested three different cameras: (1) a machine vision camera (acA1300-200uc with C125-0618-5 M lens, Basler); (2) a high-resolution webcam (Q2F-00013 LifeCam, Microsoft); and (3) a VGA webcam (C170, Logitech). Unless specified otherwise, measurements were made by the Basler camera. This camera was chosen due to its high maximal temporal resolution (203 fps), fast communication protocol (USB 3.0), and the fact that it works well in low light conditions. While the default acquisition rate during standalone operation is 100 fps, instantaneous frame rate may vary due to OS control of CPU resources and RAM buffering.

Performance was assessed on several 64-bit machines: (1) 8-core (i7-5960 X 3.0 GHz) with 32GB RAM running Windows 7; (2) 4-core (i7-6700 K, 4 GHz) with 16 GB RAM running Ubuntu 14.04.3; (3) dual-core (i5-7260U, 2.2 GHz) with 4 GB RAM running Windows 7; and (4) dual-core (i5-7260U, 2.2 GHz) with 8 GB RAM running Windows 10. While the software functions well on all configurations, the most favorable hardware configuration is the Basler/i7 combination, whereas the least involved is Logitech/i5. Unless specified otherwise, results were obtained using the 8 core/32 GB machine running Windows 7 (Table 1).

Table 1
Temporal statistics (resolution and delay) for different computer/camera combinations. All measurements were done in fast mode.

		Basler acA1300	Logitech c170
8 cores, 32GB	delay [ms]	29 ± 16	60 ± 11
	rate [fps]	132 ± 67	30 ± 3
2 cores, 4GB	delay [ms]	43 ± 8	58 ± 11
	rate [fps]	66 ± 34	30 ± 3

Table 2
Temporal statistics (resolution and delay) for different speed modes in the fastest and slowest computer/camera combinations.

	8-core (i7), Basler camera		2-core (i5), Logitech camera	
	delay [ms]	fps	delay [ms]	fps
Normal mode	37 ± 5	60 ± 4	57 ± 11	30 ± 3
Fast mode	29 ± 16	132 ± 67	57.5 ± 11	30 ± 3
No GUI	27 ± 3	100 ± 8	58 ± 11	30 ± 3

2.2. Temporal properties: delay, jitter, and resolution

The system can work in three modes, distinguished mainly by processing time: (1) normal mode, with GUI; (2) fast mode, with GUI (the default mode); (3) a no-GUI fast mode. In the *normal mode*, each frame is used for both movement tracking and GUI update. In the *fast mode*, each frame is used for movement tracking and, independently and in parallel, the GUI is updated only once every 20 ms. This reduces the delay and increases the mean frame rate (Table 2) at the cost of increased frame-to-frame variability. In the *no-GUI mode*, the GUI is not updated at all, whereas movement tracking and analog and digital outputs are maintained. Different modes may suit distinct use cases.

To synchronize the tracking system with any other data recorded on the DAQ such as neuronal, optical, and behavioral signals, an external synchronization signal (1 Hz, 50% duty cycle square wave) generated by a custom 555 circuit was used. This sync signal was recorded on the DAQ as is, and also used to blink a blue LED within the camera field of view. The tracking system was configured to detect blue blinks as an Object within an ROI, and routed these detections to a digital output recorded by the DAQ. Since the operation is sequential and digital outputs are set last in each frame, the temporal difference between (1) the rising edge of the sync signal recorded directly by the DAQ, and (2) the rising edge of the digital signal representing the blue ROI provide an upper bound estimate of the delay of the entire system (Fig. 1B): camera, PC, microcontroller, hardware circuitry, and inter-block communication.

Temporal resolution was measured using a digital signal (generated by Spotter) indicating frame onset. Each temporal resolution and delay measurement was carried out over 8–10 min for every camera/computer combination and each speed mode. The results are summarized in **Tables 1** and **2,** and full distributions are shown in Fig. 2. In the most favorable scenario, temporal resolution was 100.5 ± 7.7 fps (mean ± SD), and the delay of the system was 27.2 ± 3.4 ms. Notably, even the least favorable configuration provided stable frame rate (30 ± 3 fps) and delay (58 ± 11 ms).

2.3. Spatial properties: scaling and resolution

All features that can be linked to analog outputs (position, orientation, speed, movement direction, and angular velocity) are measured on a linear scale from 0 to 639, where 639 is the maximal value that can be measured in the system (Fig. 1C). The system uses 12-bit DACs (digital to analog converters; MCP4921, Microchip Technology) with low linearity errors (voltage offset of −3.8 mV) and a 0 to 4.64 V output, representing 640 levels at a step resolution of 7.26 mV/pixel.

When imaging a plane from a single point, the image is bound to be distorted at the edges. The precise distortion depends on setup geometry, lens, and camera. Yet for a given workspace, distortion can be reduced by elevating the camera at the cost of spatial resolution. We

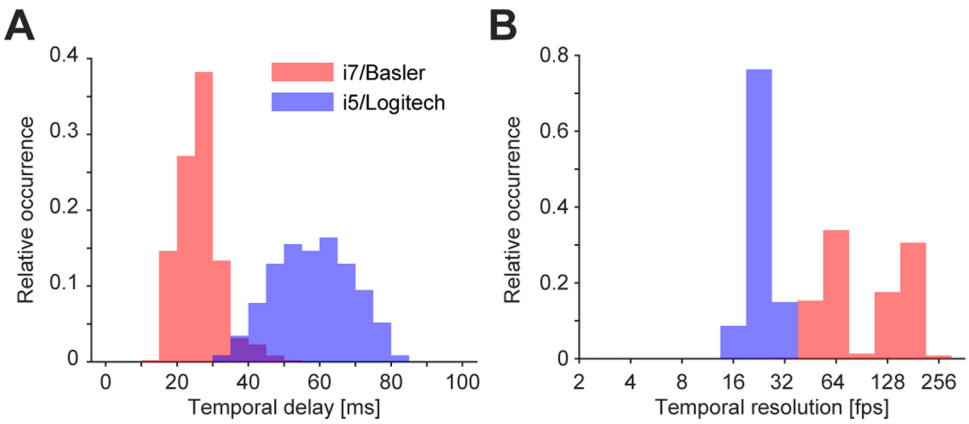

Fig. 2. Temporal properties. Distribution of temporal delay (A) and resolution (B) measurements for the fastest and the slowest combinations in the fast mode.

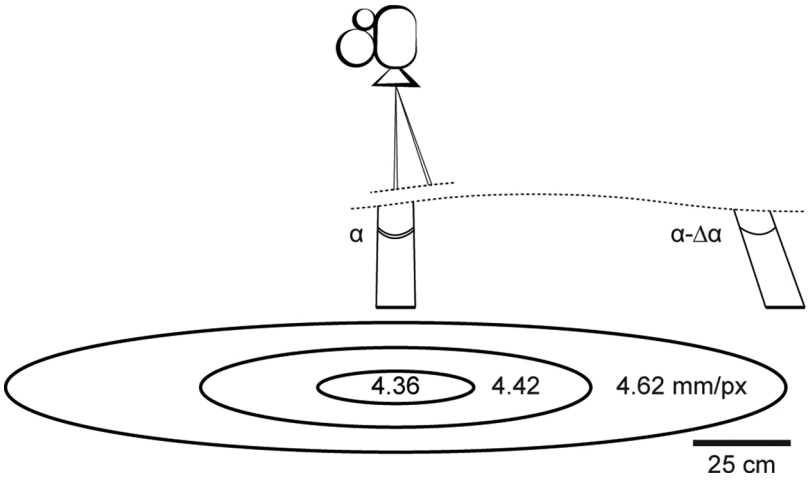

Fig. 3. Field of view. Image distortion for a camera 280 cm above arena. Spatial resolution is higher at the center of the workspace, just below the camera. The maximal distortion, at a 1 m radius, is less than 6%.

placed the camera 280 cm above the floor to record the entire 2 × 2 m arena, yielding a viewing half-angle of 19.6°. To empirically quantify spatial resolution and distortion, a two-Marker Object was connected to the edge of a rotatable 20 cm long rigid object. The object was positioned at various locations on the floor of the experimental room and rotated such that object length could be measured in pixels. Directly below the camera (i.e. at the center of the field of view), spatial resolution was 4.36 mm/pixel. One meter away from the center, spatial resolution was 4.62 mm/pixel (Fig. 3). This distortion (maximum: $(4.62 - 4.36)/4.36 = 6\%$) is an inherent caveat when imaging via a convex lens: given a particular camera and lens, distortion magnitude depends mainly on camera height. To obtain precise estimates of movement kinematics, the values calculated by the system should thus be multiplied by a position-dependent scaling factor. Such corrections were not implemented in the present system due to the small magnitude of the distortion.

2.4. Maintaining a continuous smooth output for a noisy input

For objects moving freely in a reflection- and obstacle-free workspace, the detection process (Algorithm 1) typically results in a high single-frame color-blob detection rate (> 99%). However, for a tethered mouse equipped with head-mounted LEDs running on a linear track, both LEDs are detected only in ˜75% of the frames. Furthermore, continuity is not imposed and thus features that require combining information from multiple frames (e.g. speed) may be erratic.

We identified three particular sources of inter-frame noise. First, an object may be temporarily obscured, for instance by a cable, causing loss of one or more markers. Second, changes in ambient illumination level or non-homogeneous arena topology can cause false detections (e.g. due to object reflections). Third, imperfections in the system (e.g. finite camera resolution) and user-defined parameters (e.g. poorly-defined threshold values) may cause spatial jitter such as perceived

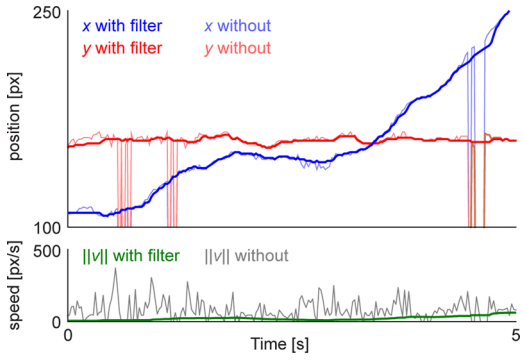

Fig. 4. Denoising effect. Position and speed with and without Algorithm 2.

Table 3
Quantification of denoising effect. The ratio between signal to noise ratio (SNR) measured before and after applying Algorithm 2. For a signal x, SNR: = LPF(x)/(x-LPF(x)), where LPF(x) is x passed through a 10 Hz lowpass filter (4-pole Butterworth).

	x	y	θ	$\|\|v\|\|$	ϕ	$\dot{\theta}$
$\frac{SNR_{before}}{SNR_{after}}$	1.67	1.79	26.5	58.7	5.73	59.1

movement of a perfectly stationary object. To account for these and other possible sources of spatial noise, we designed a denoising and location estimation procedure based on a Kalman filter (Algorithm 2), which can be enabled independently for each Object.

Without the filter, some frames are completely lost since both Markers (LEDs) are missed and no tracking is possible in 3 ± 1% of the samples (mean ± SD over 12 recordings). In 25 ± 3% other cases, only one LED is detected. In all of these cases, only position and speed may be tracked, whereas head orientation and angular velocity simply cannot be defined. Furthermore, small deviations in position are greatly amplified during temporal differentiation, inducing large deviations in speed and angular velocity.

As seen in Fig. 4, the denoising procedure (Algorithm 2) eliminated all errors caused by missing values and smoothed the deviations. Quantitatively, this resulted in an improved signal to noise ratio for all features, especially velocities (Table 3; $p < 0.001$, Mann-Whitney *U*-tests).

2.5. Application I: sensor free behavioral arena

One of the main advantages of real-time digital feedback is the possibility to create sensor-free behavioral arenas. The following two experiments demonstrate the simplicity of this concept, using digital output to replace apparatus sensors. In Application Ia, four line ROIs were defined in proximity to four photosensors positioned on a linear track (150 x 4.5 cm), and the correspondence between sensor and ROI collision events was measured (Fig. 5A).

Whenever a mouse passed a sensor, two digital pulses were generated: one by the sensor and one by the tracking system. The mouse ran across the track 98 times (left to right); during this time, sensor-ROI correspondence was 1:1 (392 sensor crossings and 392 ROI detections) with a 31.2 ± 21.8 ms delay. This delay is only 2.5 ms longer than the system delay (28.7 ± 15.6 ms; $p = 0.017$, Wilcoxon's signed rank test). The extra delay (and jitter) may be due to the difference in width between the sensor beam and a line ROI, variability in running speed, the distance between the detection triggers (tip of the nose vs. Marker on the head), and/or photosensor noise. For all practical purposes, the two types of signals are interchangeable, and both can be used to gate neural stimulation or control behavioral feedback (e.g. reward delivery).

Indeed, in Application Ib three non-consecutive, partially overlapping ROIs were used to govern reward delivery on the linear track, effectively replacing the four photosensors. The ROIs were placed on top of each other, and their activation status was interpreted as digital bits such that an object colliding with all three ROIs generates one "word" (111), while an object colliding only with the first two ROIs generates another word (110), and so on (Fig. 5Bi). These words were processed on a microcontroller, the output of which was used to open solenoids valves, dispensing liquid reward for the mouse. Behavior was similar during ROI- and photosensor-controlled runs (e.g. track traversal durations: 2.2 ± 0.499 vs. 2.25 ± 0.521 s, $p = 0.165$, Mann-Whitney *U*-test; Fig. 5Bii). Using $n = 4$ digital outputs as a binary code (0001–1111; with 0000 representing "no sensors crossed"), up to $2^n - 1 = 15$ sensors can be replaced. This approach is scalable: 8 digital outputs would enable mapping 256 non-overlapping regions in space, replacing 255 sensors.

2.6. Application II: optogenetic activation in a place-field model

Since the system can output analog signals in real-time, spatial spiking can be synthetized by selectively depolarizing neurons according to a place field model (Fig. 6A). While a freely-moving mouse traversed a linear track, 2D kinematics were monitored and the output

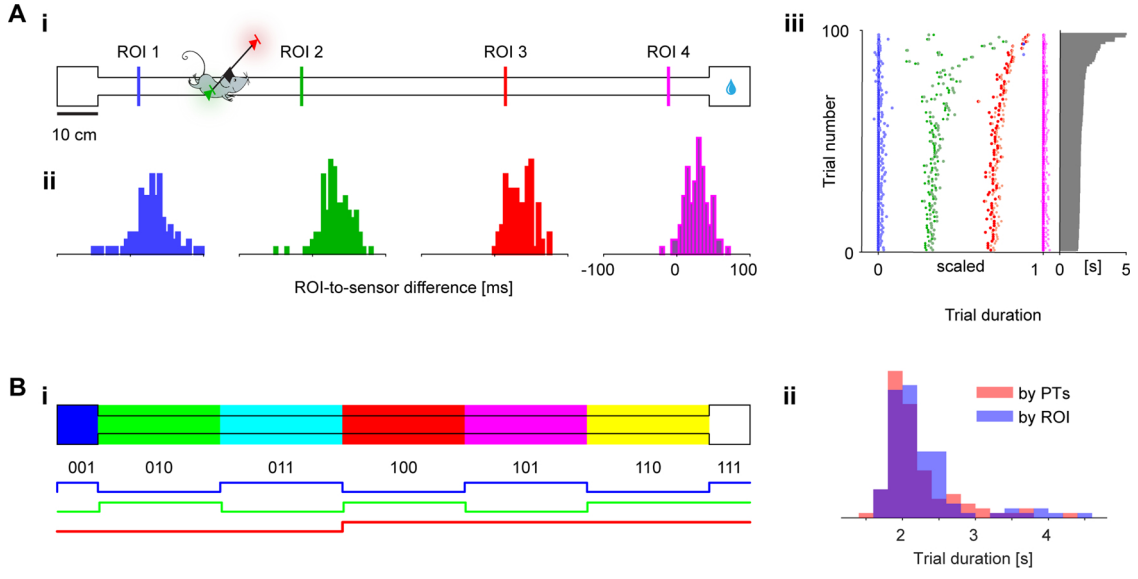

Fig. 5. Sensor-free behavioral arena. The apparatus is a linear track, consisting of a 150 x 4.5 cm runway between two square reward platforms, with four equally-spaced photosensors. (A) Replacing sensors with line ROIs. (i) Linear track with line ROIs placed close to the sensors. (ii) Histograms showing sensor-to-ROI time lag for each pair. (iii) ROI (bright colors) and sensor (dark colors) crossing times during each trial. Trials, defined as the time between the first and last sensor crossing, were scaled to the 0–1 range and sorted according to duration (horizontal bars at right). (B) (i) Partially-overlapping rectangular ROIs covering the linear track governing reward delivery, instead of sensors. (ii) Histograms show same-animal track traversal times during an ROI-controlled (blue) and a phototransistor-controlled (red) session.

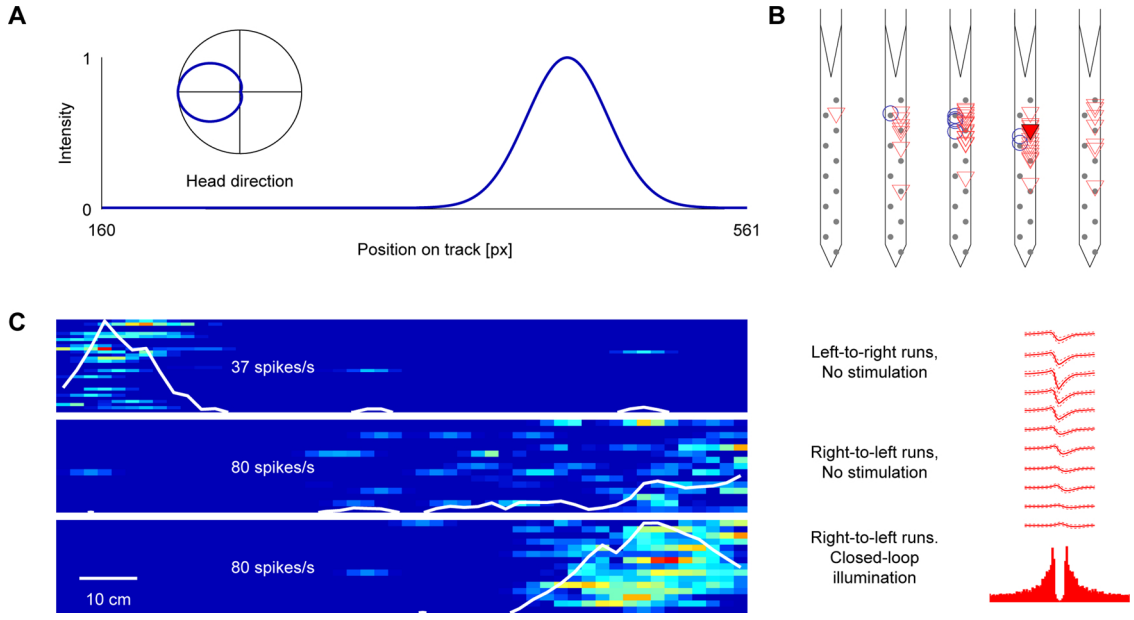

Fig. 6. Closed-loop stimulation of pyramidal cells as a function of first-order kinematics. (A) Place field model was based on animal position and head orientation. (B) Recordings were from CA1 of a freely-moving CAMKII::ChR2 mouse implanted with a 5-shank diode-probe (Stark et al., 2012). (C) The resulting spiking activity shows robust place-specific activity of a well-isolated pyramidal cell on the illuminated shank (panel B, fourth shank from left) specifically during right-to-left runs with closed-loop stimulation (30 trials per run direction).

was forwarded to a DSP. The DSP compared real-time position and orientation with a place field model (essentially, a stimulation intensity lookup table), and issued a voltage command to a current source driving a head-mounted LED coupled to an implanted optical fiber, located ~50 µm above the CA1 pyramidal cell layer (Fig. 6B). Thus, 1–80 pyramidal cells (Stark et al., 2014) were depolarized according to the real-time location and orientation of the animal, effectively transforming some into cells with spatially-selective firing (Fig. 6C).

3. Methods

3.1. The software: spotter

Spotter is a modular library written in Python 2.7. The software grabs a new frame from the camera utilizing OpenCV 2.4, an open-source machine vision library (Bradski, 2000). The frame then passes through a processing module; results are communicated via a serial interface to an external microcontroller. User input can be added through the command line or through the GUI, built with PyQt4 (Qt 4.8, The Qt Company).

3.2. Finding marker positions

Algorithm 1is based on color segmentation. It identifies user-defined Markers on a given frame using adaptive windowing. The procedure takes an RGB (red, green, blue) frame from the camera and converts it to HSV (hue, saturation, value) color space. This is particularly useful when working with bright objects such as LEDs which can saturate the RGB sensors. To minimize CPU time dedicated to detection, an adaptive window is used. For each tracked Marker, detection is attempted within a small square window (25 × 25 pixels) centered on the last detected location of the object. This window size was determined by two opposing constraints: it should be as small as possible to reduce computational load, and sufficiently large to account for the fastest changes in marker position. If the detection is unsuccessful, the window size increases for the next frame. Only expanding the window in the next frame prevents false detections and thus coising.

Algorithm 1: Detection of a Single Marker

Input: *User-specific detection parameters for Marker i:*
color HSV_i
area A_i
If exists: previous centroid C_i
previous window size

Steps:
1. **if** C_i *!= None* **then**
2. *Center w around* C_i
3. **else**
4. *Initialize detection window for whole frame*
5. **end if**
6. *Threshold pixels according to* HSV_i
7. *Ignore white pixels*
8. *Extract contours*
9. $C_i = 0$
10. **for** *each contour*
11. *Compute area* m_{00}
12. *Compute centroid* m_{xy}
13. **if** $m_{00} \geq A_i$ **then** // *detection*
14. **if** $m_{xy} \geq C_i$ **then** // *finding largest centroid*
15. $C_i = m_{xy}$
16. **end if**
17. **end if**
18. **if** $C_i > 0$ **then** // *contour detected*
19. $w = 25$
20. **return** C_i, w
21. **else** // *no detection in this frame*
22. $w = w + 25$ // *increase detection window*
23. **return** *None*, w
24. **end if**

Output: C_i *(x,y)*, w

Pixels are masked based on user-defined HSV thresholds. This is followed by contour extraction, finding continuous blobs of supra-threshold pixels. The first raw moments are then used to calculate the area (m_{00}) and centroid (m_{xy}) of each contour within the window. Marker coordinates are defined as the centroid of the contour with the largest area. White pixels, that may be caused by bright sources, are simply ignored. Thus, if light-emitting markers such as LEDs are used, their intensity must be high enough to be detected, but low enough to

prevent sensor saturation. The present experiments employed Osram PointLEDs LA P47 F (red, 617 nm) and LT P4SG (green, 528 nm) 1.9 mm diameter LEDs in a dark room. Each diode was driven by a 3 mA current, emitting 210 μW (617 nm) and 48 μW (528 nm) of light. These were assigned H value ranges of 150–10 (617 nm) and 30–80 (528 nm); the blue (470 nm) sync LED was assigned H value range 80–100. SV values were 60–255 for all wavelengths. By narrowing the H-band to 20 per wavelength, we have monitored up to 5 LEDs simultaneously (470, 528, 565, 590, and 633 nm), with room for additional markers in the violet, cyan, and deep-red bands.

3.3. Determining object features

Object position (x_{object}, y_{object}) is the arithmetic mean of the linked Markers (Fig. 1C). For a two-Marker Object, orientation θ is the angle of the normal vector between the markers on a 0 to 360° scale (Table 4). For two LEDs mounted on a mouse, inter-marker distance is limited by headstage size (e.g., 31 mm, or 7 pixels). Thus, only a finite set of orientation values can be measured. To improve orientation resolution, Markers may be placed further apart at the cost of increased headstage size. Second order features are calculated using the empirical inter-frame duration Δt. Speed $||\vec{v}||$ is the length of the vector pointing from the position of the Object in the previous frame to the current one, divided by Δt. Movement direction ϕ is the angle of this vector. Angular velocity $\dot{\theta}$ is the difference between orientation in two consecutive frames, divided by Δt.

3.4. Establishing frame to frame continuity

Once the coordinates of each Marker are found, they are fed into a denoising and position estimation algorithm (Algorithm 2) based on an adaptive Kalman filter. The algorithm combines sensor measurements from individual Markers into the state estimation of the headstage as a rigid-body with all six kinematic parameters (Lin et al., 2015; Moghari and Abolmaesumi, 2007). For a two-Marker Object, we define a frame-dependent kinematic state variable X_k (a vector of 14 elements) comprised of instantaneous Marker positions, Object position, orientation, and all velocities (v_x and v_y for each Marker and Object; angular velocity) in the k^{th} frame. The algorithm takes the best *a posteriori* estimate of kinematics in the previous frame X_{k-1} and applies a transition matrix F_k to generate an estimate $\hat{X}_k$ of the next set of measurements. F_k accounts for the underlying physics. The next measurement m is represented with a 14-element vector containing all positions and velocities. The two values (estimated, $\hat{X}_k$, and measured, m) are combined into an updated state variable X_k.

To minimize false trajectories due to missing measurements, the algorithm iterates on two branches. When at least one Marker is detected, the algorithm iterates on a “stable branch” and an “estimation branch” in parallel (see Algorithm 2). When both Markers are missing, the algorithm continues only on the estimation branch, while the stable branch remains frozen on the last stable state. Since the algorithm assumes linear changes in position, if the Markers remain occluded for several continuous frames, the estimation branch can deviate from accurate predictions. Thus once new measurements are available, the stable branch is revived from the last stable state, yielding an output which is a fusion of that state with the most recent measurements. In parallel, the estimation branch is updated to the new stable state.

Algorithm 2: Adaptive Filtering of a two-Marker Object

Initialization:
$H = I_{14}$
$R = I_{14} * 10$
$Q_0 = I_{14} * 0.01$
$X = (0 \ldots 0)^T$

while *frames are available*
Input: $M1: (x_{M1}, y_{M1})$
$M2: (x_{M2}, y_{M2})$
$\Delta t_k = t_k - t_{k-1}$
Steps:

1. $F_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

2. $X_{k-1} = \begin{pmatrix} {x_{M1}}^{k-1} \\ {y_{M1}}^{k-1} \\ {x_{M2}}^{k-1} \\ {y_{M2}}^{k-1} \\ {x_o}^{k-1} \\ {y_o}^{k-1} \\ \theta^{k-1} \\ {v_{Mx1}}^{k-1} \\ {v_{My1}}^{k-1} \\ {v_{Mx2}}^{k-1} \\ {v_{My2}}^{k-1} \\ {v_{ox}}^{k-1} \\ {v_{oy}}^{k-1} \\ \dot{\theta}^{k-1} \end{pmatrix}$

3. $m = \begin{pmatrix} x_{M1} \\ y_{M1} \\ x_{M2} \\ y_{M2} \\ \frac{x_{M1}+x_{M2}}{2} \\ \frac{y_{M1}+y_{M2}}{2} \\ mod\left(atg2\left((y_{M2}-y_{M1}),(x_{M2}-x_{M1})\right)+90,360\right) \\ \frac{x_{M1}-{x_{M1}}^{k-1}}{\Delta t} \\ \frac{y_{M1}-{y_{M1}}^{k-1}}{\Delta t} \\ \frac{x_{M2}-{x_{M2}}^{k-1}}{\Delta t} \\ \frac{y_{M2}-{y_{M2}}^{k-1}}{\Delta t} \\ \frac{x_{obj}-{x_{obj}}^{k-1}}{\Delta t} \\ \frac{y_{obj}-{y_{obj}}^{k-1}}{\Delta t} \\ \frac{\theta-\theta^{k-1}}{\Delta t} \end{pmatrix}$,

4. $L1 = (M1! = None),\ L2 = (M2! = None)\ \ MASK =$ $\begin{pmatrix} L1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & L1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & L2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & L2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & L1\,\&\,L2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & L1\,\&\,L2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & L1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & L1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L1\,\&\,L2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L1\,\&\,L2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L1\,\&\,L2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L1\,\&\,L2 \end{pmatrix}$

5. **if** *L1* or *L2*
6. $\quad stableMode = True$ // stable branch
7. $\quad \hat{X}_k = F_k * X_{k-1}$
8. $\quad \hat{P}_k = F_k * P_{k-1} * {F_k}^T + Q_k$
9. **else**
10. $\quad stableMode = False$ // estimation branch
11. $\quad \hat{X}_k = F_k * X_{ek-1}$
12. $\quad \hat{P}_k = F_k * P_{ek-1} * {F_k}^T + Q_k$
13. **end if**
14. $S_k = H * \hat{P}_k * H^T + R$
15. $K_k = \hat{P}_k * H^T * {S_k}^{-1}$
16. **if** *stableMode*
17. $\quad d_k = (m - H * \hat{X}_k)$ // stable branch
18. $\quad P_k = \hat{P}_k - (K_k * S_k * {K_k}^T)$
19. $\quad P_{ek} = P_k$
20. $\quad X_k = \hat{X}_k + K_k * MASK * d_k$
21. $\quad X_{ek} = X_k$
22. **else** // estimation branch
23. $\quad P_{ek} = \hat{P}_k - (K_k * S_k * {K_k}^T)$
24. $\quad X_{ek} = \hat{X}_k$
25. **end if**
26. **if** *L1 and L2*
27. $\quad Q_k = (1 - \alpha) * Q_{k-1} + \alpha * K_k * d_k * {d_k}^T * {K_k}^T$
28. **end if**
29. **return** X_k

Output: X_k

To handle cases in which only one Marker is recognized, we introduce a *MASK* matrix (line 4 in Algorithm 2). If both Markers are detected, the mask is simply the identity matrix. When a Marker is missing, all elements that depend on that Marker are set to 0. This way we use the information inherent to the two-marker rigid body to achieve partial fusion of the existing measurements and estimation of the missing ones.

The state-to-sensor conversion matrix H is a 14 × 14 identity matrix. Other elements denoted in Algorithm 2 including K (the Kalman gain) and P (the state covariance matrix) are initialized as identity matrices and updated in every frame. Since Δt varies between frames, the transition matrix F_k is updated every frame with the current Δt.

The observation (Q) and sensor error (R) covariance matrices are sensitive parameters in Kalman filtering. R is initialized with a diagonal matrix consisting of constants that were calculated from the covariance of 100 consecutive samples for 10 trials during stationary recordings. Q is initialized as a diagonal matrix, and is adaptively updated in every frame (Akhlaghi et al., 2017) with a forgetting factor $\alpha = 0.01$. Since Q_k is calculated from the difference between the measurement and the prediction, it may change in every step, inducing noise in the position estimation. Setting α to 1 is equivalent to the latter situation, whereas setting it to zero is akin to using a fixed value to Q, reducing the ability of the algorithm to adapt to fast changes in the covariance between the parameters.

The algorithm presented above handles a two-Marker Object. When an Object is defined by only one Marker, the coordinates of the single Marker are duplicated and used in subsequent steps as a regular two-Marker object. Orientation and angular velocity are undefined in such cases.

Table 4
Object feature calculations (n-Marker Object).

Feature	Equation	Condition
x-position	$x_{object} = \sum_{i=1}^{n} \frac{x_i}{n}$	
y-position	$y_{object} = \sum_{i=1}^{n} \frac{y_i}{n}$	
Orientation	$\theta = mod(atg2((y_2 - y_1), (x_2 - x_1)) + 90, 360)$	iff (n = = 2)
Speed	$\|\|\overrightarrow{v_k}\|\| = \frac{\sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2}}{\Delta t}$	
Direction	$\varphi_k = mod(atg2((y_k - y_{k-1}), (x_k - x_{k-1})), 360)$	
angular velocity	$\dot{\theta}_k = \frac{\theta_k - \theta_{k-1}}{\Delta t}$	iff (n = = 2)

3.5. Hardware circuitry

Analog outputs are generated by communicating data from the PC to a microcontroller (Arduino Mega) using an RS232 protocol (115,200 baud) via a USB2 port. The microcontroller in turn interfaces with the custom MC circuit using an SPI protocol. An Arduino was chosen for simplicity and the high number of GPIO pins, yet any microcontroller can be used. The MC houses four 12-bit DACs and four digital channels (Fig. 7). Each digital output is buffered through a XOR-gate and converted from 5 V to 3.3 V level to prevent reverse current, avoid circuit loading, and conform to modern 3.3 V logic.

4. Discussion

We described a modular and flexible open-source software and hardware system that enables real-time (28 ± 3 ms delay, 100 fps) low-noise tracking of the 2D kinematics of multiple objects. The system was applied to the task of tracking a freely-moving rodent equipped

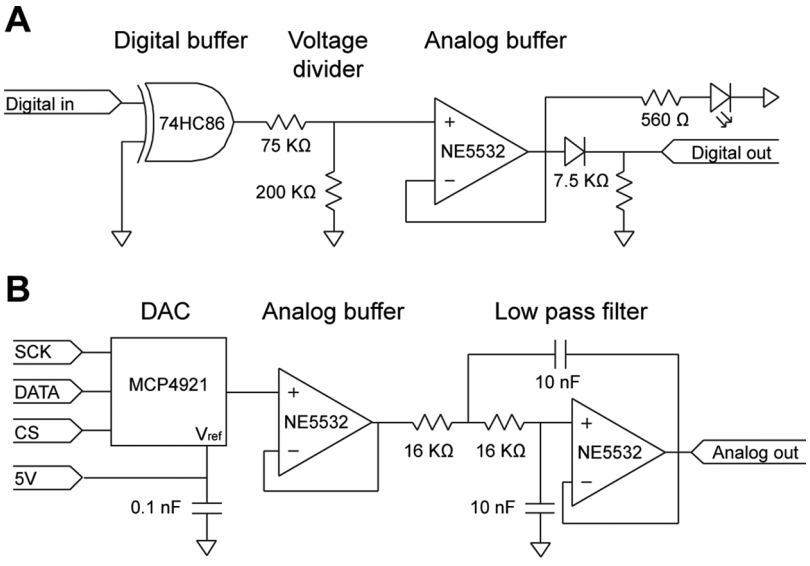

Fig. 7. Custom hardware. The Movement Controller consists of an Arduino and circuitry supporting four digital and four analog output channels. The design of each digital (A) and analog (B) channel is shown.

with on-head LEDs. Using two adaptive algorithms, the system maintained stable object tracking even when one or both LEDs were obscured, and when noise and reflections were present in the field of view. This approach eliminates the need for dedicated movement sensors in a behavioral arena and can identify up to 2^n distinct areas, n being the number of digital outputs. Combined with electrophysiological recordings, real-time calculations, and optogenetic manipulations, the system was used for closed-loop feedback and activating pyramidal cells according to animal location and head orientation, effectively transforming them into synthetic place cells.

Compared to any other open source project or commercially-available system, the system provides low-noise, real-time digital and analog outputs with high temporal resolution. A simple cross-platform GUI is provided, offering a flexible solution compatible with low-cost electronics such as an Arduino. The presented implementation included four digital and four analog outputs, yet the circuitry is scalable and the number of outputs can be increased to the number of GPIO pins on the microcontroller. By modifying the number of digital and analog outputs, the number of monitored ROIs and analog outputs can be increased without slowing the system down.

The software, Spotter, contains several additional features that can assist in reducing noise, improving the measurement, calibration and documentation of an experiment. These include blind spots (masked areas in the field of view that are therefore ignored by the algorithm), log files, video records, and Object-specific output plots (Fig. 1D was generated via Spotter). While the system performed best with a specialized machine vision camera, it yielded stable output with a two-frame delay on low-cost consumer-level devices.

Stemming from physical constraints and the nature of the targeted use-cases, several assumptions were made. The camera is assumed to be placed above an imaged field composed of uniformly colored, non-reflective material, which is completely open from above. The size, shape, and contour of the color markers, as well as inter-marker distance, limit spatial resolution. Markers must have different colors to be recognized as distinct. In small environments such as the 2 x 2 m arena used here, optical distortions are negligible (Fig. 3) and can be ignored. However, when tracking animals in large environments, distortions must be compensated for, and/or multiple cameras should be used (Saxena et al., 2018; Sourioux et al., 2018).

A natural next step is to turn the system into a standalone device, removing the need for a PC. This can be done using a USB 3.0 compatible microcontroller with strong processing capabilities. A PC may still be used for user interface. Another possible extension would be to replace visible with IR LEDs. This could utilize an OpenCV-compatible thermal camera and IR LEDs with distinct wavelengths. A third possible future direction is to allow defining custom functions on the kinematic variables before routing them to analog/digital output. For example, the distance of the mouse from a given location (Fig. 6) could be computed by the tracking system itself, or the system could output the uncertainty in the estimate of any variable (calculated from the state covariance matrix P in Algorithm 2). Finally, presently only planar kinematics (3 degrees of freedom: x, y, and yaw) are monitored. The system can be extended to track the full 3D orientation (roll, pitch, and yaw) of an object moving in 2D using a headstage with three markers arranged as a triangle perpendicular to the plane of movement.

Author contributions

N.G. wrote the software, developed and built the hardware, developed Algorithm II, collected and analyzed data, and wrote the manuscript. R.E. developed Algorithm I, wrote the software, and developed the hardware. E.S. conceived and supervised the project, developed Algorithm II, built optoelectronic devices and implanted animals, analyzed data, and wrote the manuscript.

Competing financial interests

The authors have no competing financial interests to disclose.

Acknowledgements

We thank Lisa Roux and Leore Heim for testing the system and for providing feedback on its limitations. This work was funded by a CRCNS grant (#2015577) from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and the United States National Science Foundation (NSF); and by the Israel Science Foundation (ISF; grant #638/16).

References

Akhlaghi, S., Zhou, N., Huang, Z., 2017. Adaptive Adjustment of Noise Covariance in Kalman Filter for Dynamic State Estimation. arXiv:1702.00884. .

Bradski, G., 2000. The OpenCV Library. Dr Dobbs J. Softw. Tools.

Buccino, A.P., Lepperød, M.E., Dragly, S.-A., Häfliger, P., Fyhn, M., Hafting, T., 2018. Open source modules for tracking animal behavior and closed-loop stimulation based on Open Ephys and Bonsai. J. Neural Eng. 15, 055002. https://doi.org/10.1088/1741-2552/aacf45.

Flores, A.E., Flores, J.E., Deshpande, H., Picazo, J.A., Xie, X., Franken, P., Heller, H.C., Grahn, D.A., O'Hara, B.F., 2007. Pattern recognition of sleep in rodents using piezoelectric signals generated by gross body movements. IEEE Trans. Biomed. Eng. 54, 225–233. https://doi.org/10.1109/TBME.2006.886938.

Grieves, R.M., Wood, E.R., Dudchenko, P.A., 2016. Place cells on a maze encode routes rather than destinations. eLife 5. https://doi.org/10.7554/eLife.15986.

Kalman, R.E., 1960. A new approach to linear filtering and prediction problems. J. Basic Eng. 82, 35–45. https://doi.org/10.1115/1.3662552.

Lin, Y., Chen, T., Xi, F., Fu, G., 2015. Relative pose estimation from points by Kalman filters. 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO). Presented at the 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO) 1495–1500. https://doi.org/10.1109/ROBIO.2015.7418982.

Lopes, G., Bonacchi, N., Frazão, J., Neto, J.P., Atallah, B.V., Soares, S., Moreira, L., Matias, S., Itskov, P.M., Correia, P.A., Medina, R.E., Calcaterra, L., Dreosti, E., Paton, J.J., Kampff, A.R., 2015. Bonsai: an event-based framework for processing and controlling data streams. Front. Neuroinformatics 9. https://doi.org/10.3389/fninf.2015.00007.

Maghsoudi, O.H., Tabrizi, A.V., Robertson, B., Spence, A., 2017. Superpixels Based Marker Tracking Vs. Hue Thresholding In Rodent Biomechanics Application. 2017 51st Asilomar Conf. Signals Syst. Comput. 209–213. https://doi.org/10.1109/ACSSC.2017.8335168.

Mathis, A., Mamidanna, P., Cury, K.M., Abe, T., Murthy, V.N., Mathis, M.W., Bethge, M., 2018. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. Nat. Neurosci. 21, 1281. https://doi.org/10.1038/s41593-018-0209-y.

Moghari, M.H., Abolmaesumi, P., 2007. Point-based rigid-body registration using an Unscented Kalman Filter. IEEE Trans. Med. Imaging 26, 1708–1728. https://doi.org/10.1109/TMI.2007.901984.

Moser, M.-B., Rowland, D.C., Moser, E.I., 2015. Place cells, grid cells, and memory. Cold Spring Harb. Perspect. Biol. 7. https://doi.org/10.1101/cshperspect.a021808.

O'Keefe, J., Dostrovsky, J., 1971. The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. Brain Res. 34, 171–175.

Samson, A.L., Ju, L., Kim, H.A., Zhang, S.R., Lee, J.A.A., Sturgeon, S.A., Sobey, C.G., Jackson, S.P., Schoenwaelder, S.M., 2015. Mouse Move: an open source program for semi-automated analysis of movement and cognitive testing in rodents. Sci. Rep. 5, 16171. https://doi.org/10.1038/srep16171.

Saxena, R., Barde, W., Deshmukh, S.S., 2018. Inexpensive, scalable camera system for tracking rats in large spaces. J. Neurophysiol. 120, 2383–2395. https://doi.org/10.1152/jn.00215.2018.

Skaggs, W.E., McNaughton, B.L., Wilson, M.A., Barnes, C.A., 1998. Theta phase precession in hippocampal neuronal populations and the compression of temporal sequences. Hippocampus 6, 149–172. https://doi.org/10.1002/(SICI)1098-1063(1996)6:2<149::AID-HIPO6>3.0.CO;2-K.

Sourioux, M., Bestaven, E., Guillaud, E., Bertrand, S., Cabanas, M., Milan, L., Mayo, W., Garret, M., Cazalets, J.-R., 2018. 3-D motion capture for long-term tracking of spontaneous locomotor behaviors and circadian sleep/wake rhythms in mouse. J. Neurosci. Methods 295, 51–57. https://doi.org/10.1016/j.jneumeth.2017.11.016.

Spink, A.J., Tegelenbosch, R.A.J., Buma, M.O.S., Noldus, L.P.J.J., 2001. The EthoVision video tracking system—a tool for behavioral phenotyping of transgenic mice. Physiol. Behav. Mol. Behav. Genetics Mouse 73, 731–744. https://doi.org/10.1016/S0031-9384(01)00530-3.

Stark, E., Koos, T., Buzsáki, G., 2012. Diode probes for spatiotemporal optical control of multiple neurons in freely moving animals. J. Neurophysiol. 108, 349–363. https://doi.org/10.1152/jn.00153.2012.

Stark, E., Roux, L., Eichler, R., Senzai, Y., Royer, S., Buzsáki, G., 2014. Pyramidal cell-interneuron interactions underlie hippocampal ripple oscillations. Neuron 83, 467–480. https://doi.org/10.1016/j.neuron.2014.06.023.

Wiener, S.I., Paul, C.A., Eichenbaum, H., 1989. Spatial and behavioral correlates of hippocampal neuronal activity. J. Neurosci. 9, 2737–2763. https://doi.org/10.1523/JNEUROSCI.09-08-02737.1989.