

# Physics-informed digital twin and onboard control of a brainbot for intelligent active matter

I. Mammadli <sup>1</sup>, P. Shrestha <sup>2</sup>, J. Pande <sup>3,\*</sup>, F. Novkoski <sup>1,†</sup>, S. Mohapatra <sup>1</sup>, M. Noirhomme,<sup>4</sup>  
A. Maier,<sup>2</sup> N. Vandewalle <sup>4</sup>, and A.-S. Smith <sup>1,5</sup>

<sup>1</sup>PULS, Institute for Theoretical Physics, FAU Erlangen-Nürnberg, 91058 Erlangen, Germany

<sup>2</sup>Pattern Recognition Lab, Department of Computer Science, FAU Erlangen-Nürnberg, 91058 Erlangen, Germany

<sup>3</sup>Department of Physical and Natural Sciences, FLAME University, Pune 412115, India

<sup>4</sup>GRASP, Institute of Physics B5a, University of Liège, B4000 Liège, Belgium

<sup>5</sup>Group for Computational Life Sciences, Division of Physical Chemistry, Ruđer Bošković Institute, 10000 Zagreb, Croatia



(Received 30 March 2026; accepted 16 June 2026; published 10 August 2026)

Establishing adaptive particles that sense their state, anticipate their evolution, and compute control inputs onboard has been a major challenge in nonequilibrium physics. We address this challenge by realizing an autonomous brainbot, building on a recently developed programmable bristlebot. First, we construct a physics-informed digital twin of the device, based on a kinematic model that reproduces measured trajectory statistics and generates long, statistically faithful synthetic trajectories. The kinematics forms the foundation for implementing onboard model predictive control, enabling autonomous trajectory tracking, demonstrated by accurate execution of a nontrivial target path. This provides a proof of principle for a brainbot that senses its state, predicts its evolution, and computes control inputs onboard, unlike conventional active particles with fixed motility, thereby transforming the brainbot into an agentic physical entity. By integrating physical modeling, data-driven parameter identification, and control into a unified framework, our approach provides a scalable platform for machine-learning-enabled multiagent studies and lays the groundwork for intelligent, adaptive active matter.

DOI: [10.1103/rd1t-h2v6](https://doi.org/10.1103/rd1t-h2v6)

## I. INTRODUCTION

Active matter, unlike classical passive systems, consists of energy-consuming units operating far from equilibrium, enabling dynamic behaviors such as self-organization, collective motion, and synchronization. Several paradigmatic artificial designs have been proposed over the last decade to facilitate reproducible experimental study. These include systems such as active colloids that are capable of self-propulsion through mechanisms such as chemical reactions, light stimuli, and magnetic fields [1], Quincke rollers that exhibit complex swirling motion [2,3], and vibrated granular rods [4]. Additionally, there are a variety of active asymmetric particles that move directionally due to imbalanced friction forces resulting from their shape or surface properties. One example of such systems is vibrobots, which are small particles placed on a vertically vibrating bed resulting in their horizontal motion [5,6]. While they function without internal processing, vibrobots can exhibit interesting self-organization and swarming behaviors [7,8]. Recently, chiral vibrobots have been used

to demonstrate the role of asymmetric interactions in active self-assembly [9].

Another commonly used design of active particles, relying on an internal source of energy for motion generation, is bristlebots, such as the commercially available Hexbugs™ [10–12]. Bristlebots are equipped with flexible legs and an internal vibration motor powered by a battery, transforming vertical oscillation of the body into horizontal motion [13]. This setup allows a considerably larger operational space and has been used to study active particles in harmonic potentials [10], to construct active solids [11] and elastoactive structures [14], and to mimic antlike pheromone trail behavior by way of Hexbugs™ moving through fields of passive particles [15].

While bristlebots have been widely used, their locomotion became programmable and controllable only recently through the integration of ARM® microcontrollers into custom-built platforms, giving rise to so-called brainbots. These versatile, centimeter-sized devices are equipped with light, color, and magnetic field sensors, as well as infrared emitter-receivers [16,17]. Powered by a lithium-ion battery providing 40–60 min of operation, they exhibit motion patterns that can be tuned from ballistic to diffusive dynamics. Their trajectories depend sensitively on leg inclination and motor power [16]. This behavior can be captured by a single parameter,  $\eta$ , which quantifies the similarity of a trajectory to pure spinning and varies systematically with driving strength and geometry, enabling continuous transitions between straight, spiral, and circular motion [16]. However, the limited battery power restricts the exploration of long-term dynamics under complex

\*Contact author: jayant.pande@flame.edu.in

†Contact author: filip.novkoski@fau.de

Published by the American Physical Society under the terms of the Creative Commons Attribution 4.0 International license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

driving protocols and the collective behavior of multiple units. This limitation could be overcome by developing a digital twin to systematically investigate such regimes computationally and experimentally validate only the most relevant scenarios. At present, such a digital twin is not available.

Understanding the motion of bristlebots and brainbots is necessary to improve the control of their trajectories and enhance their performance and applicability. Physical models of bristlebot locomotion, based on the flexibility of the legs, have been introduced in one dimension [13,18] and two dimensions [19]. However, these have not been successfully combined with experimental data, nor do they account for the diversity of trajectories that such robots can exhibit [16]. As such, a theoretical framework to capture the relevant kinematics has been missing. Due to their widespread use as active matter systems, endowing individual bristlebots with control capabilities would pave the way for them to become compact experimental platforms for intelligent active particles [20]. Recent advances in the control algorithms tailored to resource-constrained microprocessors of the kind used on these bots have aided the development of such systems, enabling advanced control in real time [21–23].

In this work, we introduce a physics-based model that describes the various trajectories executed by a brainbot by decoupling the experimentally observed translational, spinning, and orbital components of the motion. We also define an optimization procedure that fits the model to the experiments to generate relevant parameter sets that can describe the complex experimental trajectories with high accuracy, effectively building a physics-informed digital twin. We use our digital brainbot to generate artificial trajectories on timescales far beyond experimentally accessible ones. Notably, the distributions of  $\eta$  and of the linear and angular velocities that emerge from training on experimental data are preserved in this extrapolation. Moreover, ensembles of independently generated long artificial trajectories maintain the same characteristic  $\eta$  and velocity distributions, demonstrating the robustness and predictive consistency of our digital representation.

Because the digital brainbot reproduces behavior accurately across the full range of  $\eta$ , spanning diverse driving intensities and geometries, it provides a powerful tool for optimizing task-specific driving protocols. More broadly, it enables systematic exploration of complex physical scenarios through a synergistic interplay between experiment and *in silico* realizations, made possible by the stability and fidelity of the digital twin established here.

Finally, our model of the brainbot, which relies on kinematics and does not require a complete resolution of the physics of the friction-based bristle-induced motion, is then employed to implement onboard predictive control of a single brainbot following a predefined trajectory. By relying on a lightweight solver designed for small-scale embedded systems, we enable a single brainbot to autonomously compute its control inputs to track a prescribed reference trajectory and make real-time adjustments in case of deviations. To the best of our knowledge, this is the first demonstration of onboard control for a bristlebot, underscoring the novelty of the approach and its relevance to the experimental realization of intelligent active matter.

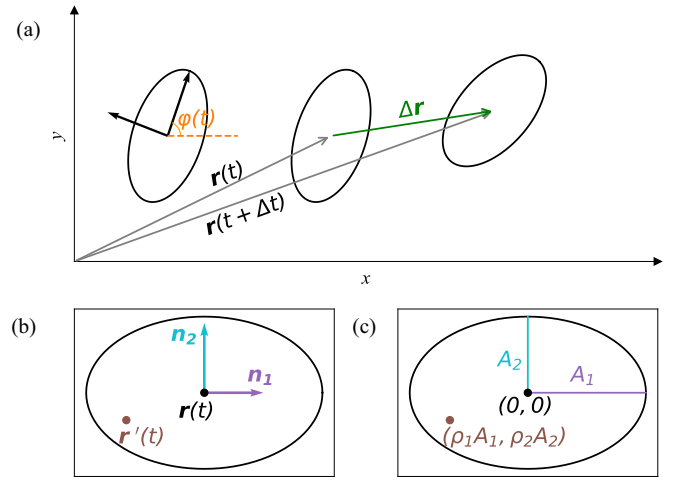


FIG. 1. (a) Position vector  $\mathbf{r}(t)$  and orientation angle  $\varphi(t)$  of a brainbot at time  $t$ , in the global reference frame. (b) Position vectors of the geometric center  $\mathbf{r}$  of the brainbot and an arbitrary point  $\mathbf{r}'$  in the laboratory frame, and the instantaneous unit vectors  $\hat{\mathbf{n}}_1$  and  $\hat{\mathbf{n}}_2$  along the major and minor axes, respectively. (c) Semimajor and semiminor axis lengths  $A_1$  and  $A_2$ , and the coordinates of the geometric center  $(0, 0)$  and an arbitrary point  $(\rho_1 A_1, \rho_2 A_2)$  in the body-fixed reference frame.

## II. KINEMATICS OF BRAINBOT MOTION

### A. Analytical description of trajectories

Inspired by its experimental counterpart, our digital brainbot is assumed to move predominantly in two dimensions, here taken to be the  $x$ - $y$  plane. The brainbot is represented by an elliptical shape, whose geometric center is described by the two-dimensional position vector  $\mathbf{r}(t)$  in the laboratory frame at time  $t$ . The orientation of the brainbot is described by the angle  $\varphi(t)$  that the major axis of the brainbot makes with the horizontal  $x$  axis at time  $t$  [Fig. 1(a)].

Let  $A_1$  and  $A_2$  be the semimajor and semiminor axis lengths of the brainbot ellipse, and let  $\hat{\mathbf{n}}_1$  and  $\hat{\mathbf{n}}_2$  be the unit vectors, in the laboratory frame, along the major and minor axes of the ellipse, given by

$$\hat{\mathbf{n}}_1(\varphi) = (\cos \varphi, \sin \varphi), \quad (1)$$

$$\hat{\mathbf{n}}_2(\varphi) = (-\sin \varphi, \cos \varphi). \quad (2)$$

Let  $\mathbf{r}'$  denote an arbitrary point in the plane (inside or outside the ellipse) in the laboratory frame, and let its coordinates be  $\rho_1 A_1$  and  $\rho_2 A_2$  in the reference frame of the brainbot, where  $\rho_1$  and  $\rho_2$  are scale factors. If  $\mathbf{r}'$  is on or inside the ellipse, then the  $\rho$ 's must lie between  $-1$  and  $1$ , and must in fact satisfy the condition

$$\rho_1^2 + \rho_2^2 \leq 1. \quad (3)$$

The coordinates of the geometric center  $\mathbf{r}$  and the arbitrary point  $\mathbf{r}'$  are related by

$$\mathbf{r}' = \mathbf{r} + \rho_1 A_1 \hat{\mathbf{n}}_1 + \rho_2 A_2 \hat{\mathbf{n}}_2, \quad (4)$$

while their velocities ( $\mathbf{v}'$  at  $\mathbf{r}'$  and  $\mathbf{v}$  at  $\mathbf{r}$ ) are related by

$$\mathbf{v}' = \mathbf{v} + \boldsymbol{\omega} \times (\mathbf{r}' - \mathbf{r}), \quad (5)$$

where  $\boldsymbol{\omega} \equiv (0, 0, \omega)$  is the angular velocity vector.

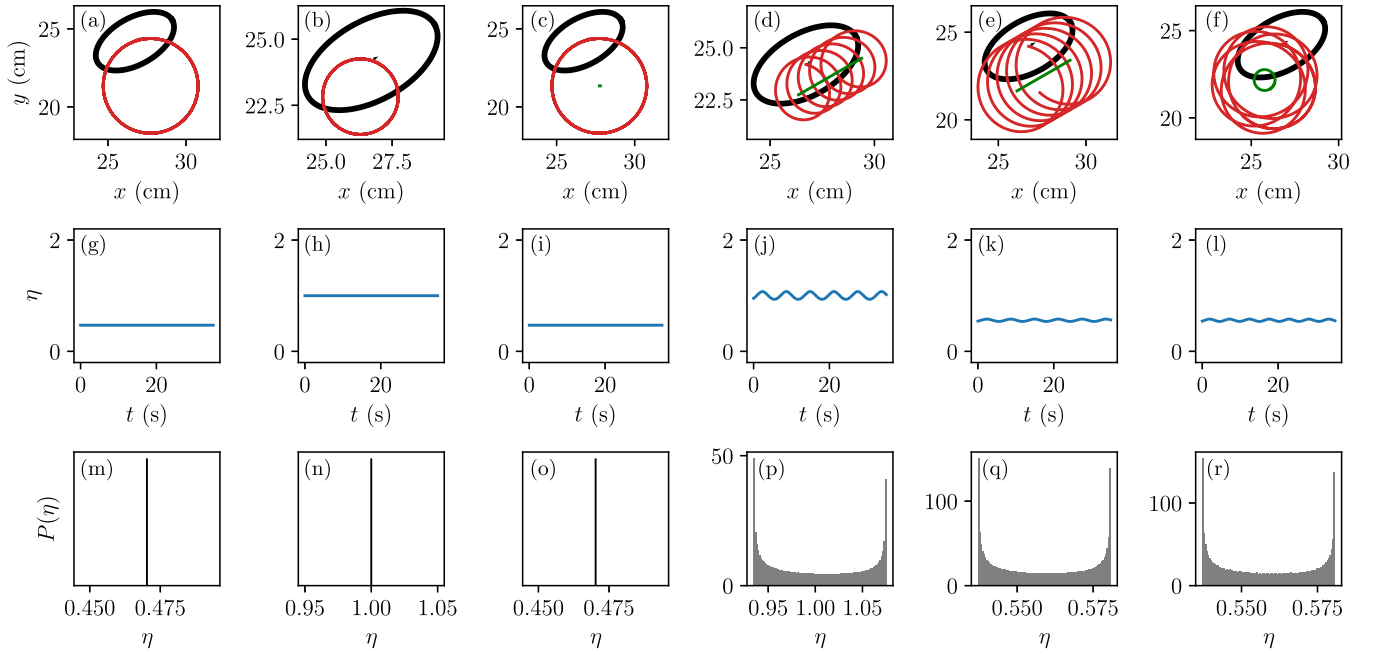


FIG. 2. Top row (a)–(f) Deterministic trajectories of the geometric center (shown in red) and the instantaneous center of rotation  $\mathbf{r}_c$  (shown in green) of a brainbot, found using the kinematic model. The black ellipse in each panel indicates the starting position of the brainbot. Middle row (g)–(l) The values of the  $\eta$  parameter corresponding to the different trajectories shown in the top row. For purely linear, spinning, and orbital trajectories, the  $\eta$  values are constant, while for other trajectories,  $\eta$  is sinusoidal. Bottom row (m)–(r) The probability distributions of the different  $\eta$  values attained in the corresponding plots in the middle row. To get each distribution, the trajectory was calculated for 200 s. For visualization purposes, the plotted trajectories cover only the first 30 s.

Let  $\mathbf{r}_0(t)$  denote the instantaneous center of rotation of the brainbot, i.e., the point at a given instant at which the velocity  $\mathbf{v}$  is zero. Then,

$$\mathbf{r}_0(t) \equiv \left( x - \frac{v_y}{\omega}, y + \frac{v_x}{\omega} \right), \quad (6)$$

where  $(x, y)$  and  $(v_x, v_y)$  are the components of  $\mathbf{r}$  and  $\mathbf{v}$ , respectively. At a given moment, the center of rotation of the brainbot may lie inside or outside the brainbot. Let  $\mathbf{r}_c(t)$  denote the position of either of the rearmost legs of the brainbot, such that  $|\mathbf{r} - \mathbf{r}_c|$  denotes the distance between the geometric center and the positions of the rearmost legs. Regardless of which rear leg is considered, the value of  $|\mathbf{r} - \mathbf{r}_c|$  is fixed due to the symmetry along the major axis of the brainbots. With these definitions, we introduce a curvature parameter  $\eta$  to quantify the similarity of an observed trajectory to spinning motion,

$$\eta = \frac{|\omega| |\mathbf{r} - \mathbf{r}_c|}{|\mathbf{v}|}. \quad (7)$$

As discussed in our previous work [16], this parameter is useful to characterize the different trajectories, with values of  $\eta$  close to 0 indicating linear motion and values close to 1 indicating spinning motion. In particular, spinning occurs when the brainbot rotates around one of its rear legs, in which case we let  $\mathbf{r}_c$  be the position of that rear leg, that is,  $\mathbf{r}_c = \mathbf{r}_0$ . From Eqs. (6) and (7), this implies  $\eta = 1$ .

To understand the relationship between the distribution of  $\eta$  along trajectories and the overall properties of brainbot motion, we study some basic driving protocols. Specifically, we present a kinematic model that assigns to the brainbot

a translational velocity  $\mathbf{v}_c$  that is held constant either in the body-fixed frame or in the laboratory frame.

### 1. $\mathbf{v}_c$ constant in the body-fixed frame

We first explore the case of constant  $\mathbf{v}_c$  always applied at the point  $(\rho_1 A_1, \rho_2 A_2)$  in the body-fixed frame, where  $\rho_1$  and  $\rho_2$  are constants. This may result, for example, from internal driving acting along the long axis of the ellipse. In the absence of an internal rotation, this trivially results in motion along a straight line [Fig. 2(a)]. Introducing a constant  $\omega$ , which seems to be intrinsic to the experimental design, gives  $\mathbf{v}_c$  of the form

$$\mathbf{v}_c = u_1 \hat{\mathbf{n}}_1 + u_2 \hat{\mathbf{n}}_2, \quad (8)$$

where  $u_1$  and  $u_2$  are the constant velocities in the  $\hat{\mathbf{n}}_1$  and  $\hat{\mathbf{n}}_2$  directions, respectively. Using Eq. (5), we obtain

$$\begin{aligned} \frac{d}{dt} \mathbf{r} &= u_1 \hat{\mathbf{n}}_1 + u_2 \hat{\mathbf{n}}_2 - \omega \times (\rho_1 A_1 \hat{\mathbf{n}}_1 + \rho_2 A_2 \hat{\mathbf{n}}_2) \\ &= [\sin \varphi (\omega \rho_1 A_1 - u_2) + \cos \varphi (\omega \rho_2 A_2 + u_1)] \hat{\mathbf{i}} \\ &\quad + [\sin \varphi (\omega \rho_2 A_2 + u_1) + \cos \varphi (-\omega \rho_1 A_1 + u_2)] \hat{\mathbf{j}}, \end{aligned} \quad (9)$$

where  $\hat{\mathbf{i}}$  and  $\hat{\mathbf{j}}$  denote the unit vectors along the  $x$  and  $y$  directions, respectively, in the laboratory frame. Integrating with respect to  $t$  (with  $dt = d\varphi/\omega$ ), we get

$$\begin{aligned} \mathbf{r}(t) &= \mathbf{r}(0) + \left\{ \left( \rho_2 + \frac{u_1}{A_2 \omega} \right) A_2 [\sin \varphi(t) - \sin \varphi(0)] \right. \\ &\quad \left. + \left( -\rho_1 + \frac{u_2}{A_1 \omega} \right) A_1 [\cos \varphi(t) - \cos \varphi(0)] \right\} \hat{\mathbf{i}} \end{aligned}$$

$$+ \left\{ \left( -\rho_1 + \frac{u_2}{A_1\omega} \right) A_1 [\sin \varphi(t) - \sin \varphi(0)] - \left( \rho_2 + \frac{u_1}{A_2\omega} \right) A_2 [\cos \varphi(t) - \cos \varphi(0)] \right\} \hat{\mathbf{j}}. \quad (10)$$

Defining the modified scale factors  $\rho'_1 \equiv \rho_1 - u_1/(A_2\omega)$  and  $\rho'_2 \equiv \rho_2 + u_1/(A_2\omega)$ , we obtain

$$\mathbf{r}(t) = \mathbf{r}_{\text{const}} - R \{ \cos[\varphi(t) + \alpha] \hat{\mathbf{i}} + \sin[\varphi(t) + \alpha] \hat{\mathbf{j}} \}, \quad (11)$$

where the constants  $\mathbf{r}_{\text{const}}$ ,  $R$ , and  $\alpha$  are defined by

$$\mathbf{r}_{\text{const}} = \mathbf{r}(0) - [\rho'_2 A_2 \sin \varphi(0) - \rho'_1 A_1 \cos \varphi(0)] \hat{\mathbf{i}} + [\rho'_1 A_1 \sin \varphi(0) + \rho'_2 A_2 \cos \varphi(0)] \hat{\mathbf{j}}, \quad (12)$$

$$R = \sqrt{\rho_1'^2 A_1^2 + \rho_2'^2 A_2^2}, \quad (13)$$

and

$$\alpha = \cos^{-1} \left( \frac{\rho'_1 A_1}{\sqrt{\rho_1'^2 A_1^2 + \rho_2'^2 A_2^2}} \right) \quad (14)$$

$$= \sin^{-1} \left( \frac{\rho'_2 A_2}{\sqrt{\rho_1'^2 A_1^2 + \rho_2'^2 A_2^2}} \right). \quad (15)$$

Eq. (11) shows that for a fixed  $\omega$  and  $\mathbf{v}_c$  constant in the body-fixed frame, the center of the brainbot executes a circular trajectory in the laboratory frame [Fig. 2(b)], with its center at  $\mathbf{r}_{\text{const}}$  and its radius given by  $R$ .

If  $\mathbf{v}_c = \mathbf{0}$ , then the brainbot undergoes purely spinning motion [Fig. 2(b)], centered at the point  $(\rho_1 A_1, \rho_2 A_2)$  in the body-fixed frame [which in this case is also fixed in the laboratory frame, at  $\mathbf{r}_{\text{const}}$  given by Eq. (12)]. In this case, the radius of the spinning trajectory is  $\sqrt{\rho_1^2 A_1^2 + \rho_2^2 A_2^2}$ .

For a nonzero  $\mathbf{v}_c$ , the radius of the circular trajectory can be larger or smaller than the spinning radius [Fig. 2(c)]. We call this orbital motion. In this case, depending on the signs of  $u_1$  and  $u_2$ , the center of rotation may move inward (toward the geometric center of the brainbot) or outward, and can lie outside the brainbot.

## 2. $\mathbf{v}_c$ constant in the laboratory frame

We now consider the case when  $\mathbf{v}_c$  is a constant vector in the laboratory frame and is again applied at the fixed point in the brainbot defined by  $(\rho_1 A_1, \rho_2 A_2)$  in the body-fixed frame, where  $\rho_1$  and  $\rho_2$  are constants. This corresponds to subjecting the device to constant external driving. Under such circumstances, and with constant  $\omega$ ,  $\mathbf{v}_c$  is sinusoidal in the body-fixed frame (Appendix A).

The analog of Eq. (11) in this case is

$$\mathbf{r}(t) = \mathbf{r}_{\text{const}} + \mathbf{v}_c t - R \{ \cos[\varphi(t) + \alpha] \hat{\mathbf{i}} + \sin[\varphi(t) + \alpha] \hat{\mathbf{j}} \}, \quad (16)$$

where  $\mathbf{r}_{\text{const}}$ ,  $R$ , and  $\alpha$  are as defined in Eqs. (12)–(14), except with  $\rho'_1$  and  $\rho'_2$  replaced by  $\rho_1$  and  $\rho_2$ , respectively.

Eq. (16) shows that when  $\mathbf{v}_c$  is kept fixed in the laboratory frame, the geometrical center of the brainbot undergoes a combination of translational motion (at constant velocity

TABLE I. Input velocities for the simulation trajectories shown in Fig. 2. The cases  $\omega = 0$  and  $\mathbf{v}_c = \mathbf{0}$  result in purely linear and purely spinning motion, respectively. In cases (d) and (e), the velocity  $\mathbf{v}_c$  is sinusoidal in the body-fixed frame (and constant in the laboratory frame; see Appendix A for details) and is defined using the oscillation frequency  $\omega$ . In case (f), a different frequency  $\nu = -1.2 \text{ rad/s} \neq \omega$  is used to define  $\mathbf{v}_c$ .

| Panel | Trajectory                    | $\omega$ (rad/s) | $\mathbf{v}_c$ (body-fixed frame, cm/s)             |
|-------|-------------------------------|------------------|-----------------------------------------------------|
| (a)   | Linear                        | 0                | 0.4                                                 |
| (b)   | Spinning                      | -1.0             | 0                                                   |
| (c)   | Orbital                       | -1.0             | (2, 0.5)                                            |
| (d)   | Helical                       | -1.0             | $0.1(\cos \omega t, -\sin \omega t)$                |
| (e)   | Orbital + Helical             | -1.0             | $(0.08, 0.08) + 0.1(\cos \omega t, -\sin \omega t)$ |
| (f)   | Complex ( $\nu \neq \omega$ ) | -1.0             | $(0.08, 0.08) + 0.1(\cos \nu t, -\sin \nu t)$       |

$\mathbf{v}_c$ ) and circular motion (which is a combination of spin and orbital motion, as in Sec. II A 1). This results in helical trajectories [Figs. 2(d) and 2(e)].

More complex trajectories, such as in Fig. 2(f), may be obtained by combining the two cases of  $\mathbf{v}_c$  constant and sinusoidal in the body-fixed frame.

## B. Classification of trajectories

The different forms of  $\mathbf{v}_c$  discussed in Secs. II A 1 and II A 2 allow us to reproduce the various autonomous trajectories executed by a brainbot. The different fundamental modes of motion are discussed in Table I, with the corresponding simulated trajectories shown in Fig. 2 (top row). To understand how the above-discussed modes of motion (top row in Fig. 2) correspond to the classification parameter  $\eta$ , we explore the time evolution of  $\eta$  (second row in Fig. 2) and its associated distribution (third row in Fig. 2). Linear and circular trajectories (the latter including both spinning and orbital trajectories) are characterized by a single value of  $\eta$ , resulting in a delta-function distribution  $P(\eta)$ . For linear and spinning motion,  $\eta$  equals 0 and 1, respectively, while for orbital motion  $0 < \eta < 1$ .

For the helical [Figs. 2(d) and 2(e)] and more complex trajectories [Fig. 2(f)],  $\eta$  varies with time following one sinusoidal mode. Consistent with the probability distribution of any sinusoidal function with binning, in these cases  $\eta$  has a probability distribution that is peaked at the extreme values of  $\eta$  and is at its minimum at the mean value of  $\eta$  [Figs. 2(d)–2(f), third row].

The experimental trajectories are inherently more complex. For example, a Fourier mode decomposition of  $\omega(t)$  from the experimental trajectory (Appendix C) shows that at least 10 Fourier modes are needed to get a reasonable distribution of  $\eta$ , while about 70 Fourier modes are required for an accurate representation of the measured trajectory. The complexity of the experimental trajectories suggests that for a digital representation of the brainbot to be faithful to the experiments, we must make it learn its parameters directly from the experiments. We must also choose an efficient approach for removing high-frequency noise in the data that provides an optimal trade-off between the descriptive accuracy achieved and the computational overhead required.

### III. DIGITAL BRAINBOT

#### A. Forward kinematic framework

The establishment of a digital twin of the brainbot requires the development of a numerical framework that guarantees adherence of the bot's motion to the underlying physical constraints at every time step. Consequently, the principles of the motion formalized above are cast into a forward kinematic model. This requires numerically integrating  $d\mathbf{r}/dt$ , like in Eq. (9). All the data and code used are made available [24,25].

A pseudocode for the digital brainbot is given in Algorithm 1. In this code,  $\rho_1$  and  $\rho_2$  are set to the values  $-0.374$  and  $0.661 \text{ sign}(\bar{\omega})$ , respectively, as these values correspond to the locations of the back legs, relative to the geometric center of the bot. The sign of  $\rho_2$ , which decides whether the left or the right rearmost leg is being considered, is adjusted based on the rotation direction  $\bar{\omega}$ . Subsequently,  $\mathbf{r}_c$  is recalculated (lines 8 and 9 in Algorithm 1), corresponding to the correct rearmost leg. The values as well as the adjustment of the rotation center ( $\rho_2$ ) are based on the inversion of the experimental trajectories, as detailed in Ref. [16]. Details of the numerical integration scheme used, including the rotation matrix  $\mathbf{M}(\bar{\omega}\Delta t)$  and the mean velocity  $\bar{\mathbf{v}}_c$ , are provided in Appendix B.

#### B. Offline parametric identification model

To accurately represent the brainbot in the digital form, the forward kinematic framework must be complemented with time-dependent  $\mathbf{v}_c$  and  $\omega$ . These parameters must be inferred from the experimental data. To this end, we rely on a gradient descent method to minimize the overall least-squares error between the experiments and the kinematic framework across the trajectories, for given geometric and driving conditions. We consider the process complete when the variance in the parameters stabilizes, which indicates that the digital twin effectively reflects and generalizes the brainbot's actual physical behavior.

For this purpose, we use a trajectory dataset that comprises the position  $(x, y)$  and the orientation  $\varphi$  of the brainbot's geometric center at each time step. Each trajectory lasts approximately 20 s and contains about 800 data points. To suppress high-frequency noise, we apply a Savitzky-Golay filter to the trajectories [26], following the procedure described in Ref. [16].

From these processed trajectories,  $\omega(t)$  can be obtained by numerically differentiating  $\varphi(t)$  with respect to time. This sequence is used to generate a histogram of  $\omega$  values for each trajectory across the entire run of the experiment and then fit the resulting distribution of  $\omega$  to a skew-normal distribution. These distributions are stored for each driving and geometry condition of the brainbot.

To extract  $\mathbf{v}_c$ , we assume it to be a sum of the velocity profiles in Secs. II A 1 and II A 2. That is, in the laboratory frame, we assume  $\mathbf{v}_c$  to be a sum of constant and sinusoidal terms, namely,

$$\mathbf{v}_c(t) = \{u_{11} + u_{12} \cos[\alpha_1 + \varphi(t) + \beta t]\}\hat{\mathbf{n}}_1 + \{u_{21} + u_{22} \sin[\alpha_2 + \varphi(t) + \beta t]\}\hat{\mathbf{n}}_2. \quad (17)$$

#### ALGORITHM 1. Digital brainbot kinematic framework.

---

```

1: Initialize Brainbot ( $x^{(0)}, y^{(0)}, \varphi^{(0)}, \rho_1 \leftarrow -0.374$ 
2: Prescribe  $\mathbf{v}_c(t)$  and  $\omega(t)$ 
3:  $\hat{\mathbf{n}}_1^{(0)} \leftarrow \hat{\mathbf{n}}_1(\varphi^{(0)})$  and  $\hat{\mathbf{n}}_2^{(0)} \leftarrow \hat{\mathbf{n}}_2(\varphi^{(0)})$ 
4:  $\omega^{(0)} \leftarrow \omega(0), \mathbf{v}_c^{(0)} \leftarrow \mathbf{v}_c(0)$ 
5: for  $k = 0$  to  $N_{\text{steps}} - 1$  do
6:    $\omega^{(k+1)} \leftarrow \omega((k+1)\Delta t), \mathbf{v}_c^{(k+1)} \leftarrow \mathbf{v}_c((k+1)\Delta t)$ 
7:    $\bar{\omega} \leftarrow 0.5(\omega^{(k)} + \omega^{(k+1)})$ 
8:    $\rho_2^{(k)} \leftarrow 0.661 \text{ sign}(\bar{\omega})$ 
9:    $\mathbf{r}_c^{(k)} \leftarrow \mathbf{r}^{(k)} + \rho_1 A_1 \hat{\mathbf{n}}_1^{(k)} + \rho_2^{(k)} A_2 \hat{\mathbf{n}}_2^{(k)}$ 
10:   $\varphi^{(k)} \leftarrow \varphi^{(k)} + \bar{\omega} \Delta t$ 
11:   $\hat{\mathbf{n}}_{1,2}^{(k+1)} \leftarrow \hat{\mathbf{n}}_{1,2}(\varphi^{(k+1)})$ 
12:   $\bar{\mathbf{v}}_c \leftarrow 0.5(\mathbf{v}_c^{(k)} + \mathbf{v}_c^{(k+1)})$ 
13:   $\mathbf{r}_c^{(k+1)} \leftarrow \mathbf{r}_c^{(k)} + \bar{\mathbf{v}}_c \Delta t$ 
14:   $\mathbf{r}^{(k+1)} \leftarrow \mathbf{M}(\bar{\omega} \Delta t)(\mathbf{r}^{(k)} - \mathbf{r}_c^{(k)}) + \mathbf{r}_c^{(k+1)}$ 
15: end for

```

---

The seven parameters  $\mathbf{u} \equiv (u_{11}, u_{12}, u_{21}, u_{22}, \alpha_1, \alpha_2, \beta)$  are found by minimizing  $|\mathbf{r}(t) - \mathbf{f}_s(t; \mathbf{u})|^2$  over  $\mathbf{u}$  (least squares), across all the 800 data points in each trajectory, using the explicit  $\omega(t)$  sequence. Here,  $\mathbf{r}(t) \equiv (x(t), y(t))$  denotes the trajectory points and  $\mathbf{f}_s(t; \mathbf{u})$  is the simulator that integrates the  $\mathbf{u}$  parameters. The term  $\beta t$  in the argument of the sinusoidal functions enables the long-term trajectory to exhibit a chiral nature, with a nonzero value of  $\beta$  corresponding to case (f) in Table I.

Providing the seven parameters to the kinematic framework completes the digital representation of the brainbot. We compare four experimental and synthetic trajectories [Figs. 3(a)–3(d)] with examples progressively going from circular to linear to cover a broad range of possible driving protocols and bot geometries. This is reflected in the  $\eta$ -parameter values corresponding to these trajectories, shown in panels (e)–(h) in Fig. 3, ranging from around 1 to 0 [16]. The very good agreement between the experimental and the simulated curves indicates that the form of  $\mathbf{v}_c$  in Eq. (17) is justified and that the parametric identification model is robust.

#### C. Applying the brainbot digital twin to generate long synthetic trajectories

In the previous section, synthetic trajectories arose from the parametric optimization procedure used to construct the digital twin. Here, we employ the fully trained digital twin to generate ensembles of synthetic trajectories, using the training data shown in Fig. 3 as reference. For each parameter set, ten independent trajectories were generated. The complete ensemble is shown in Fig. 4, where the top row displays trajectories of the same duration as the experimental template, while the second row presents trajectories that are ten times longer.

For each parameter set and for each trajectory, we use the values of  $\mathbf{u}$  as found in Sec. III B, while the values of  $\omega$  are drawn from the skew-normal distribution (dashed red line in Fig. 4, fourth row), yielding sequences for  $\omega$  shown in the third row of Fig. 4, with the dark curves showing the training set and the red curves the generated set. The main panels in this row are for long times (ten times as long as the

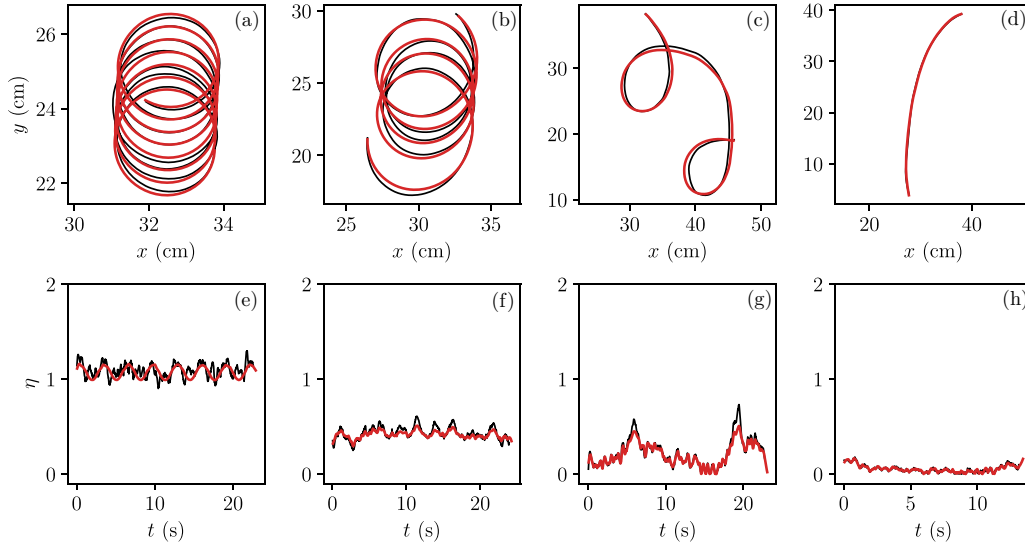


FIG. 3. Results of simulation (in red) compared to experimental data (in black). Given that the overall, cycle-averaged displacement of the brainbot in these experiments is linear,  $\beta$  in Eq. (17) is set to zero to reduce the number of fitted parameters. The upper row (a)–(d) shows the trajectories, while the lower row (e)–(h) shows the corresponding values of the  $\eta$  parameter for each trajectory as a function of time. Values of  $\eta$  close to 1 indicate circular trajectories, while values close to 0 indicate linear trajectories [16]. The experimental parameters (leg angle  $\alpha_{\text{leg}}$  and effective motor voltage  $V_E$ ) in panels (a)–(d) are (a)  $\alpha_{\text{leg}} = 15^\circ$ ,  $V_E = 2.7$  V, (b)  $\alpha_{\text{leg}} = 15^\circ$ ,  $V_E = 3.0$  V, (c)  $\alpha_{\text{leg}} = 25^\circ$ ,  $V_E = 2.1$  V, and (d)  $\alpha_{\text{leg}} = 5^\circ$ ,  $V_E = 2.1$  V.

experiments), while the insets show the  $\omega$  plots for short times (as long as the experiments).

Visual inspection of the synthetic trajectories demonstrates the robustness and accuracy of the brainbot’s digital twin. Each reinitialization produces a new trajectory that is consistent with the distributions of the linear velocity and  $\eta$  across the phase diagram (fifth and last rows in Fig. 4, respectively), while the distribution of  $\omega$  is maintained by construction. Even a significant extrapolation in time keeps the statistical properties of the trajectories intact (as evidenced by the distributions of  $v$  and  $\eta$ ). At the same time, it expands the observed morphology of the trajectories: for instance, the apparently linear trajectory in Fig. 4(d) becomes helical when extended for a long time. This shows the digital twin to be a faithful and informative extension of the experimental brainbot, while being computationally efficient.

#### D. Validation of the digital twin

To demonstrate that the long-term dynamics exhibited by the digital twin reproduce those of the physical brainbot, we examine two representative modes of motion: a linear helix [Figs. 5(a)–5(f)] and a circular helix [Figs. 5(g)–5(l)]. These cases combine translational and rotational dynamics while remaining sufficiently confined to be explored within our experimental setup.

The experimental trajectories (in black in Fig. 5, first row) last 65 s each. In each case, 10% of the experimental trajectory (outlined in green in Fig. 5, first row) is used to train the digital twin. Using the training data, a skew-normal distribution for  $\omega$  is found (Fig. 5, second row), from which  $\omega$  is picked randomly for our simulation runs to capture the effects of noise. For the linear helical trajectory, the fitting in Eq. (17) is done without the parameter  $\beta$ , whereas  $\beta$  is retained in the

case of the circular helical trajectory due to the chirality of this trajectory. The third row in Fig. 5 shows for each case one example trajectory obtained from the simulations, and the last three rows in Fig. 5 display a comparison between the distributions of  $\omega$ ,  $v$ , and  $\eta$  as obtained from the experiments (in black) and in the long-time simulations of the digital twin of the brainbot (in red). These results provide clear evidence that the short segments of experimental trajectories used to train the simulations are sufficient to reproduce both the qualitative behavior of the brainbot across the different trajectory classes tested and the long-term statistical properties of its modes of motion. This indicates that the kinematic framework provides a sound foundation for the digital twin and that the trained model generalizes well from short experimental observations to long-term behavior. Consequently, the digital twin fulfills its intended role as a statistical surrogate of the brainbot: rather than predicting the exact future trajectory of a particular experiment beyond its runtime or extrapolating beyond the physical information contained in the training data, it is designed to generate ensembles of trajectories that preserve the experimentally determined statistical distributions of  $\omega$ ,  $v$ , and  $\eta$  over timescales inaccessible to direct experiment. The ability to reproduce these statistical properties is a necessary prerequisite both for using the digital twin as a statistical surrogate and for employing it as a development platform for an onboard controller introduced in the following section.

#### IV. IMPLEMENTATION OF THE KINEMATIC FRAMEWORK ON A PHYSICAL BOT

We now exploit the kinematic model to control a physical brainbot [17]. To this end, we employ model predictive control [27], a control framework that leverages an

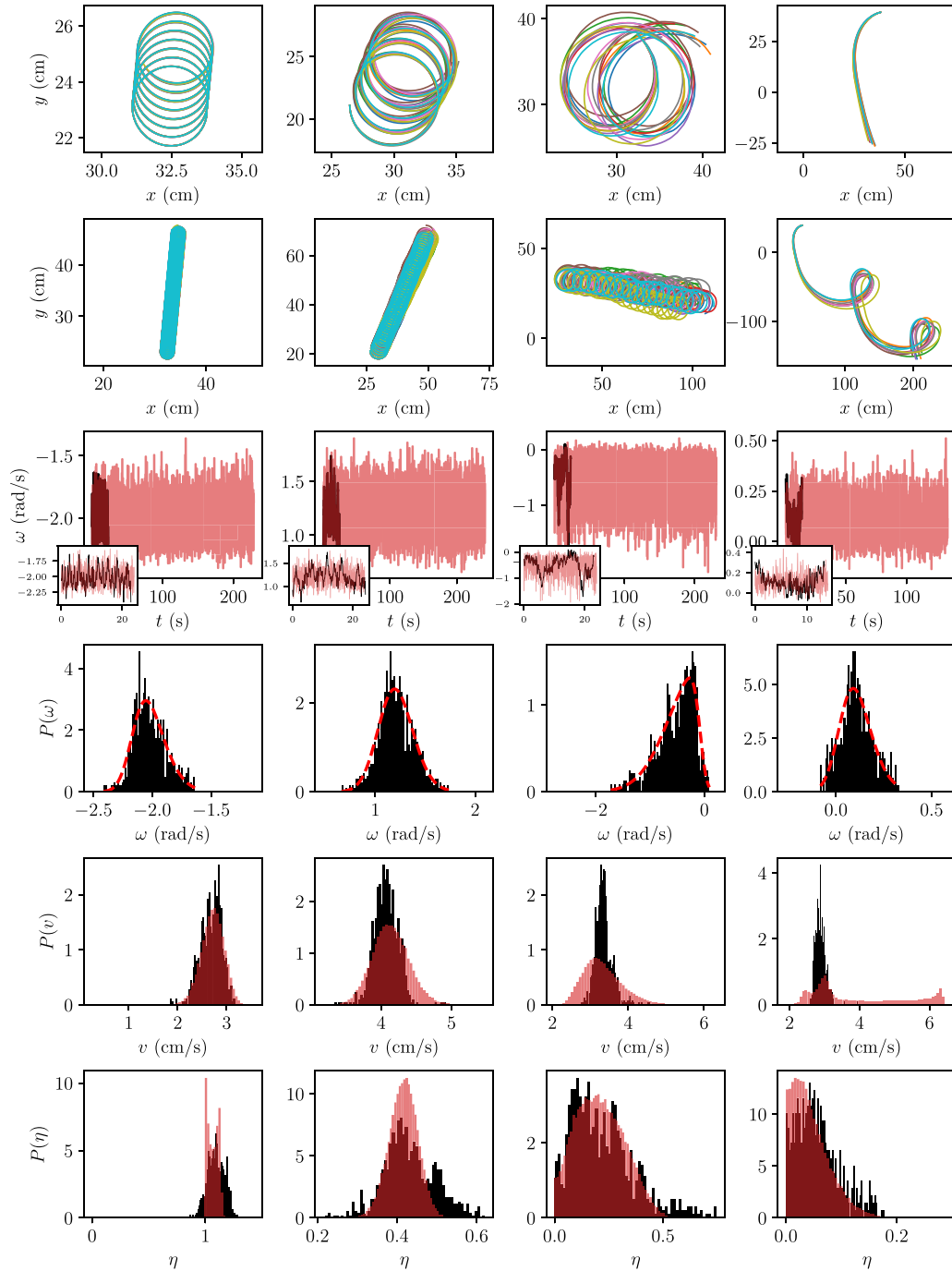


FIG. 4. Using a digital twin to generate long straight trajectories. Top row: Synthetic trajectories similar to the four experimental trajectories shown in Fig. 3, and run for the same time duration. There are ten different trajectories shown, each marked in a different color for clarity. Because the cycle-averaged trajectories are intrinsically straight, the six-parameter model was used, with  $\beta = 0$  in Eq. (17). Second row: The same respective trajectories, but now for simulation time steps that are ten times longer than the experimental duration. Third row: Variation in  $\omega$  with time, from the experiments (black curves) and the long simulations (red curves), with the insets showing the comparison when the time durations are kept the same for the simulations and the experiments. Fourth row: Histograms (in gray) of  $\omega$  values obtained from the simulations, and the resulting skew-normal distribution function (shown by the dashed red line), from which  $\omega$  values are picked to obtain the trajectories in the top row. Fifth row: Histograms of  $v$  from the experiments (in gray) and the simulations (in red). Bottom row: Histograms of  $\eta$  values from the experiments (in black) and from the simulations (in red).

empirical/analytical system model to predict the system's behavior over a short time horizon and provide optimal control inputs accordingly. The MPC solver has been implemented on board, enabling the brainbot to take autonomous actions based

on its current state. The digital twin developed in the previous section is central to this implementation, serving both as a test bed for prototyping the controller *in silico* and as the kinematic model that informs the onboard MPC.

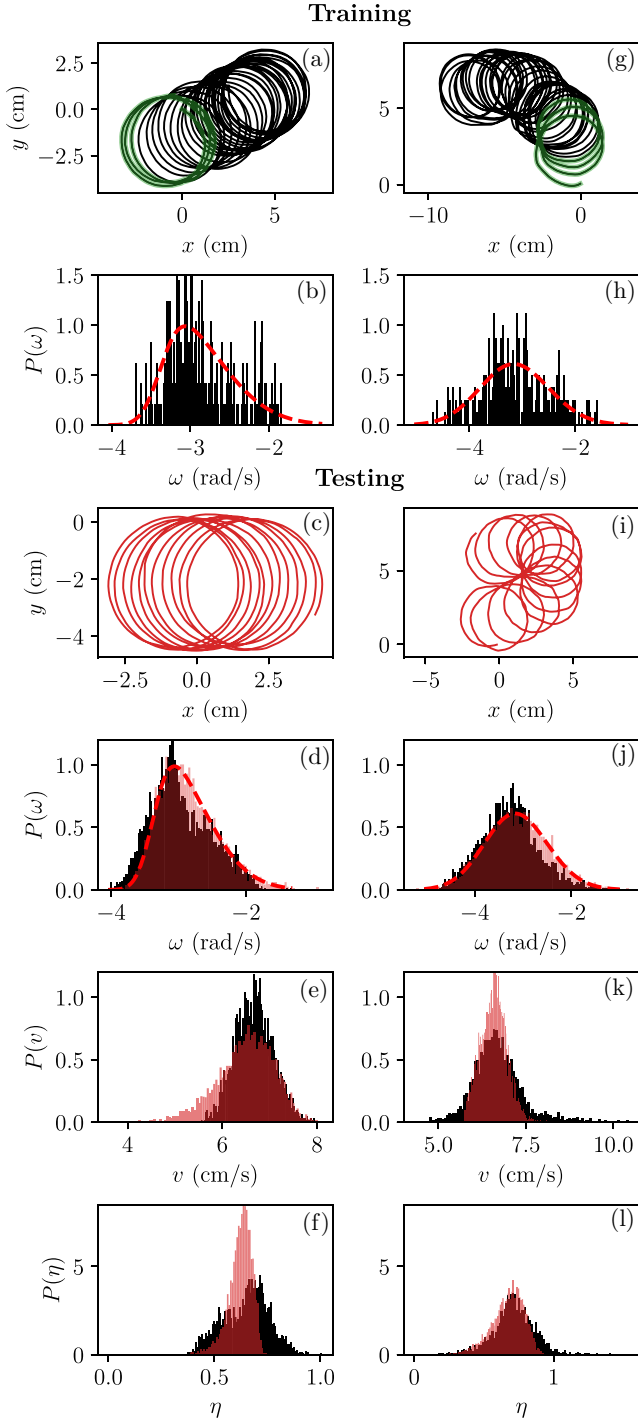


FIG. 5. Validation of the digital twin. Panels (a)–(f) pertain to a linear helical trajectory, while panels (g)–(l) pertain to a circular helical trajectory. The first row shows the experimental trajectories (in black) with 10% of the data (marked in green) used to train the digital twin [using Eq. (17)] with six and seven parameters, respectively, for the two trajectories. The second row shows the  $\omega$  histograms obtained from the training data and skew-normal distributions (dashed red line) fitted to these histograms. The third row shows an example of the resulting simulated trajectories (tested over the remaining duration, i.e., 90% of the total experimental runtime). The fourth, fifth, and sixth rows show, for  $\omega$ ,  $v$ , and  $\eta$ , respectively, the comparison between the distributions obtained from the experiments (in black) and from long simulations (in red).

Considering the computational constraints of the brainbot, TinyMPC [22] has been chosen for solving the control problem due to its well-balanced computational speed and memory footprint, and adaptability to small embedded systems. Moreover, its operation on the same class of microprocessors (ARM Cortex series) as used in the brainbot has been demonstrated *a priori* [22,28]. Specifically, the bot is equipped with a front Bluetooth module, similar to the one described in Ref. [17], which contains a Cortex-M4 (STM32WB55) for which we compile TinyMPC through the standard Arduino IDE and which is used for all computation needed for the MPC.

To be able to solve the control problem, TinyMPC requires the state of the brainbot  $s = [x, y, v_x, v_y]^\top$ , corresponding to its current position  $(x, y)$  and velocities of its geometric center in the laboratory frame  $(v_x, v_y)$ , respectively. The orientation of the bot  $\varphi$ , although not a part of the state variable, is also required for a subsequent step of the control workflow. Experimentally, the state is obtained through a positioning system using an optical camera and an Aruco marker attached to the brainbot, yielding real-time tracking of the brainbot's position, orientation, and velocity. These measurements are then transmitted to the brainbot at a frequency of 30 Hz through the Bluetooth protocol, initializing the finite horizon quadratic problem (QP) that is solved by TinyMPC. The full pipeline of the control process is shown in Fig. 6(a).

The solver predicts the evolution of the state of the bot over a horizon of predefined length  $N$  via the linear system model explained later, searching for the sequence of control inputs  $\{u^{(j)} \mid j = k, \dots, k + N - 1\}$  ( $k$  denoting the current time step), that minimizes the cost function, while constrained to the imposed state and control input bounds. Only the first control input of the sequence,  $u^{(k)}$ , is used for the state update, and the remaining sequence is discarded as the uncertainty associated with the prediction accumulates along the horizon. Equation (18) delineates the quadratic problem involving the cost function inspired by [22]

$$\begin{aligned}
 \min_{\{u^{(j)}\}_{j=k}^{k+N-1}} \quad & \sum_{j=k}^{k+N-1} (\Delta s^{(j)\top} Q \Delta s^{(j)} + u^{(j)\top} R u^{(j)}) \\
 & + \Delta s^{(k+N)\top} P \Delta s^{(k+N)}, \\
 \text{subject to} \quad & s^{(j+1)} = A_d s^{(j)} + B_d u^{(j)}, \\
 & u_{\min} \leq u^{(j)} \leq u_{\max}, \\
 & s_{\min} \leq s^{(j)} \leq s_{\max},
 \end{aligned} \tag{18}$$

where  $s^{(k)} = [x^{(k)}, y^{(k)}, v_x^{(k)}, v_y^{(k)}]^\top$  and  $u^{(k)} = [a_x^{(k)}, a_y^{(k)}]^\top$  are the state and control inputs at the  $k$ th time step,  $N$  is the prediction horizon, and  $\Delta s^{(k+j)} = s^{(k+j)} - s_{\text{ref}}^{(k+j)}$  is the deviation of the current/predicted state from that of the reference state.

The cost function consists of three terms, from left to right: the tracking cost, the control effort, and the terminal cost, with  $Q$ ,  $P$ , and  $R$  being positive semidefinite cost matrices. The matrices  $Q$  and  $R$  are diagonal matrices with the values representing the weights of the respective costs, while the matrix  $P$  is a symmetric matrix generated by TinyMPC as a solution to the Riccati equation [29]. The weights for the

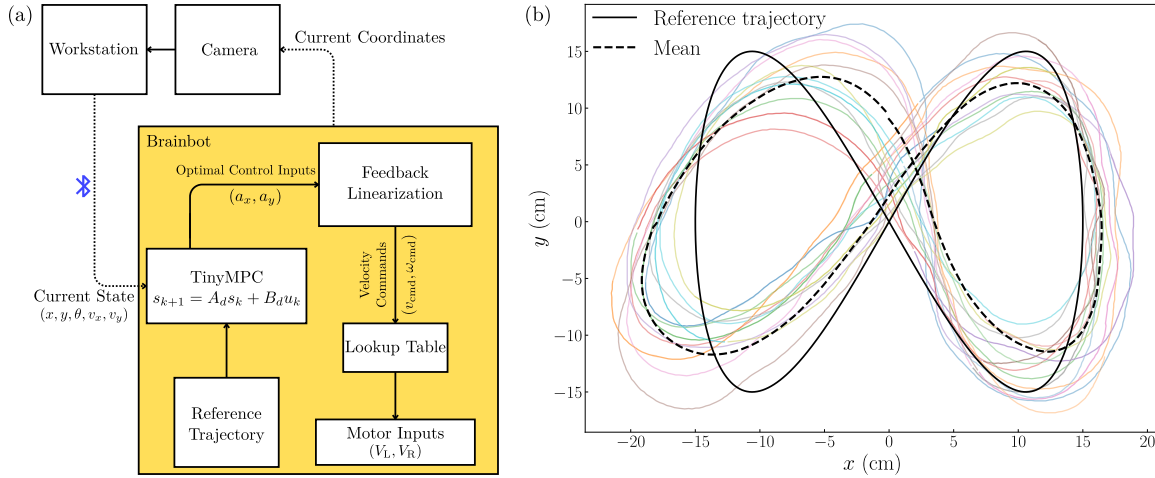


FIG. 6. (a) Schematic diagram of the model predictive control implementation for a brainbot, consisting of the onboard control based around TinyMPC (yellow) and the external positioning system, relying on a camera and tracking software, with the two communicating via Bluetooth. (b) Trajectory of a brainbot controlled through TinyMPC as it executes a lemniscate (figure eight), using a reference trajectory (solid black). Each of the differently colored lines represents individual cycles of the lemniscate, with the mean trajectory represented by the dashed black line.

costs are chosen heuristically from a preliminary *in silico* implementation of the TinyMPC pipeline, replacing the external positioning setup with a simulation-based state measurement of the virtual bot. These weights are then applied to the experimental implementation (see Appendix D for the initialization of the cost matrices). The tracking cost penalizes the deviation of the predicted state from the reference state, while the control effort penalizes the large control inputs for the state update. Together, these costs account for the stage cost and are accumulated over the prediction horizon  $\{k, \dots, k + N - 1\}$ , where  $k$  is the current time step. Terminal cost, on the other hand, represents the state deviation at the terminal step  $k + N$  and is essential to ensure numerical stability of the optimization scheme and to prevent undesirable state update beyond the prediction horizon.  $u_{\min}$ ,  $u_{\max}$ ,  $s_{\min}$ , and  $s_{\max}$  are the input and state bounds. These bounds are chosen based on the experimentally observed performance limits of the bot. Among the state variables, the positions  $x$  and  $y$  are left unconstrained, whereas the velocity components  $v_x$  and  $v_y$  are bounded by the maximum observed velocity of the bot, i.e., 15 cm/s. The control inputs  $a_x$  and  $a_y$ , on the other hand, are bounded by the maximum acceleration inferred from the bot reaching its top speed from rest in 0.066 s.

TinyMPC owes its efficiency in part to the use of linearized models, namely, through the linear double integrator, with a fixed prediction time step  $\Delta t$ ,

$$s^{(k+1)} = A_d s^{(k)} + B_d u^{(k)}, \quad (19)$$

where the discretized system matrices with prediction time step  $\Delta t$  are

$$A_d = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B_d = \begin{bmatrix} \frac{1}{2} \Delta t^2 & 0 \\ 0 & \frac{1}{2} \Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}.$$

The system model assumes the bot to be a point mass located at its geometric center. To ensure consistency with the physical brainbot, the kinematics laid out in Sec. II are reintroduced into the control pipeline in the feedback linearization layer. The virtual control input  $[a_x^{(k)}, a_y^{(k)}]^\top$  provided by TinyMPC after solving the minimization problem represents the desired accelerations of the geometric center in the laboratory frame and is used to obtain updated virtual velocities  $v_x$  and  $v_y$ . These virtual velocities, along with the orientation  $\varphi$  transmitted by the positioning setup, are then mapped to a desired absolute value of the velocity  $v_{\text{command}}$  of the geometric center of the bot and a desired angular velocity  $\omega_{\text{command}}$  about the center of rotation of the bot, through Eqs. (5) and (9). These values serve as a command to the bot and are the velocity values it is supposed to execute.

The subsequent stage in the control pipeline involves the conversion of these virtual desired values into motion of the physical brainbot by modulating the power inputs to the left and right motors ( $V_L$ ,  $V_R$ ). This conversion is carried out using precomputed lookup tables that define the motor commands as functions of  $v_{\text{command}}$  and  $\omega_{\text{command}}$ . These tables are generated offline and stored in the onboard memory of the bot for real-time use.

To construct the lookup tables, the translational and angular velocities of the bot are measured for a range of left and right motor-input combinations, each in the range of [60, 120]. The resulting calibration data are then used to construct forward models by bivariate polynomial regression, considering a third-order polynomial and a second-order polynomial for the translational and angular velocities, respectively (Fig. 8). These forward models are subsequently inverted using an iterative nonlinear least-squares procedure, yielding the motor inputs required to realize prescribed values of  $v_{\text{command}}$  and  $\omega_{\text{command}}$  (see Appendix E for details).

The performance of the embedded model predictive control on the brainbot is demonstrated through a path-following exercise, using a lemniscate as the reference trajectory for

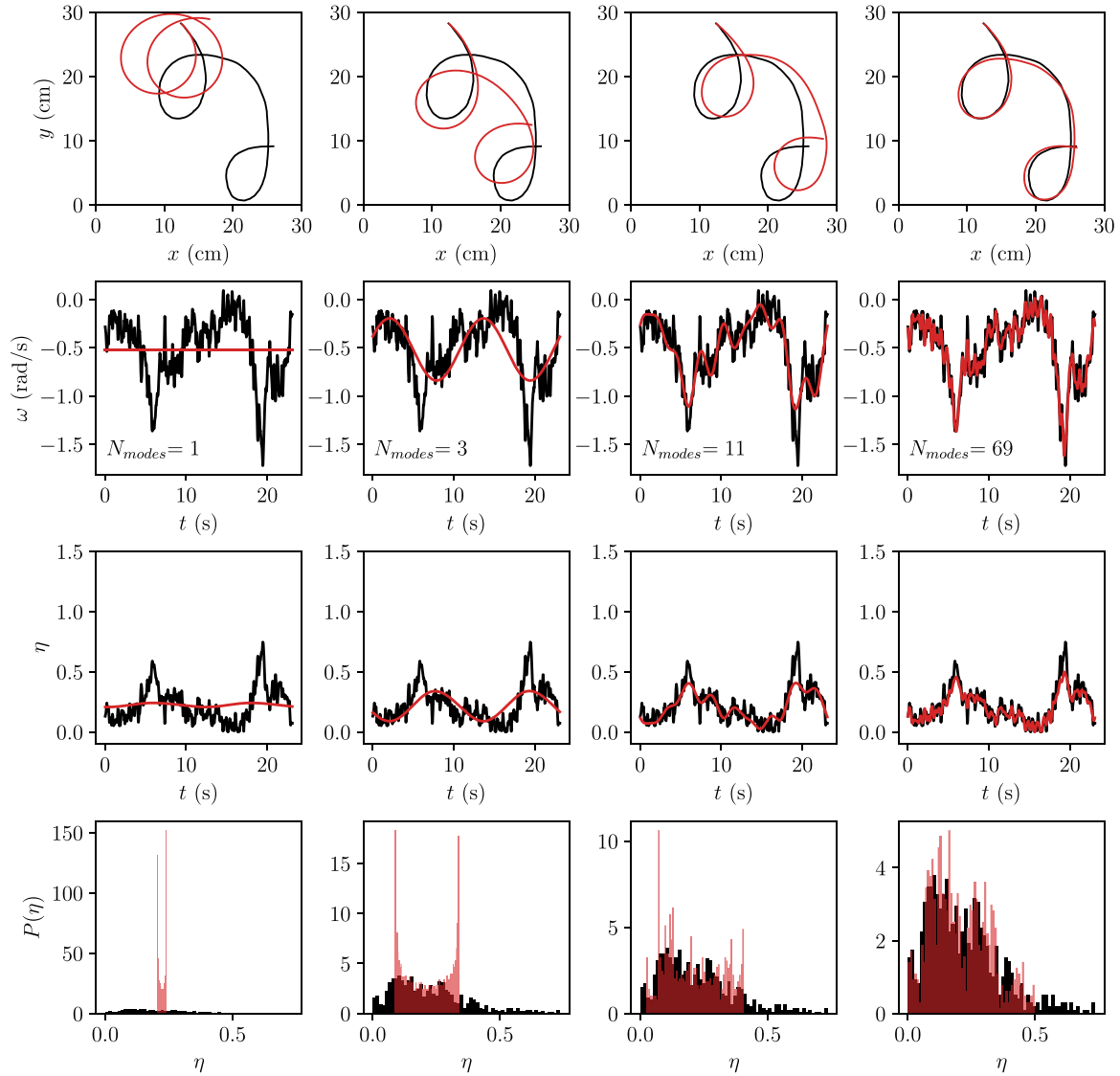


FIG. 7. The top row shows synthetic trajectories corresponding to the experimental trajectory in the third column of Fig. 3, for different values of  $\omega$  picked in the simulations. These  $\omega$  values are picked by first plotting  $\omega$  vs time (black curves in the second row), and then taking Fourier modes to different orders (red curves in the second row). The number of Fourier modes used is, respectively, 1, 3, 11, and 69 from left to right in the second row. The third row shows the resulting variation of  $\eta$  with time, as measured in the experiments (in black) and in the simulations (in red). The bottom row shows the histograms for  $\eta$  values, as found for the experiments (in black) and from the simulations (in red).

the bot. We place the bot on a  $1 \times 1 \text{ m}^2$  plexiglass plate, allowing ample space for trajectory execution, with the bot never reaching the boundaries. The lemniscate is parametrized as  $x = A \sin(2\pi t/T)$ ,  $y = B \sin(2\pi t/T) \cos(2\pi t/T)$ , where  $T$  allows control over the period of completion of one lemniscate cycle. The code used here has been made available [24]. The geometric parameters of the lemniscate are set to  $A = 15 \text{ cm}$  and  $B = 30 \text{ cm}$  based on the field of view of the camera, while the time period is set to  $T = 28 \text{ s}$  to avoid very low translational and angular velocities. Using the positioning system, the position  $(x(t), y(t))$  of the bot is recorded, allowing us to trace the path executed by the physical bot as shown in Fig. 6(b), over approximately 16 consecutive instances of the reference trajectory. In dashed lines, we show the mean trajectory of the bot, which is determined by taking

advantage of the periodicity of  $x(t)$  and  $y(t)$ , and averaging their periods.

The results show that the bot is able to systematically reproduce a lemniscate trajectory with a small error, consistent between different realizations of the trajectory and in agreement with previous implementations of TinyMPC [22]. Besides this generally good performance, an overall asymmetry of the trajectory is also observed, likely due to the inherent mechanical imperfections of the bot in turning left or right, as well as to the approximation introduced by the linearization of control. Despite the slight deviations, this experiment serves as a clear validation of the kinematic framework and the stability of the system. Furthermore, it demonstrates the ability of the brainbot to solve a control problem onboard, utilizing a kinematics-informed MPC approach, while also providing

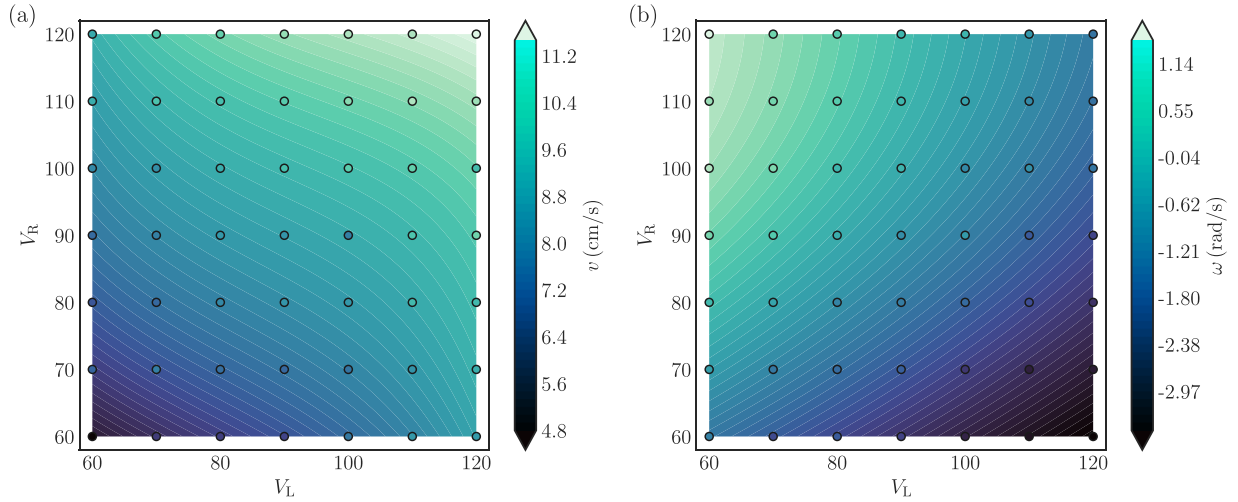


FIG. 8. The calibration maps for (a) velocity  $v$  and (b) angular velocity  $\omega$  against inputs to the left  $V_L$  and right  $V_R$  motors. The circles represent the recorded data from the calibration experiments.

a reliable experimental platform for a small-scale intelligent active particle capable of autonomous decision-making based on predictive control. To the best of our knowledge, this is the first realization of a bristlebot using onboard model predictive control for navigation of its environment.

## V. DISCUSSION AND CONCLUSION

We have presented a kinematic model that clearly decomposes the complex trajectories observed in experimental realizations of brainbot motion into basic modes of motion. Thereby, we are able to decouple the linear, spinning, and orbital components. The model is agnostic of the dependence of the linear and angular velocities of the brainbot on the driving parameters, which is necessary because the motion of the brainbot exhibits a complex dependence on a variety of factors such as the leg shape and angle, the motor voltage, and surface irregularities. Nevertheless, once the velocities are prescribed, the forward kinematic framework reproduces all the experimentally observed trajectories.

We have integrated this physical understanding into a forward kinematic framework, which was trained on experimental data using an offline parametric identification approach to create a digital representation of a brainbot. Apart from reproducing experimental trajectories, we have created synthetic trajectories, which differ from the experiments in controlled ways yet mimic their desired properties. In particular, we have demonstrated how we can pick random values of the angular velocity  $\omega$  from its experimentally obtained distribution, to generate simulation trajectories that can run for arbitrarily long times. In these simulations, the distributions of the curvature parameter  $\eta$  as well as of the linear and angular velocities  $v$  and  $\omega$  are similar to those found from the experiments, as confirmed by training the system on 10% of an experimental run and testing it against the rest of the experiment. This process effectively yields a functional digital replica of the brainbot.

These digital twins are important for providing data both on an infinite domain and for long time periods, overcoming experimental limitations associated with finite energy

resources and limited size of the observation platform. In previous work, we have shown that brainbots can be programmed to execute diffusive motion on long timescales, while on short timescales they exhibit ballistic motion [16,17]. The framework constructed here can be used to find more efficient algorithms of motion that lead to diffusion on a given timescale, without the need for a large number of experiments. It also allows an extension of the bot behavior to more complicated desired regimes, such as superdiffusive and subdiffusive motion. Additionally, access to a large ensemble of numerical trajectories can prove useful in training machine learning models. Such approaches can learn from the generally stochastic statistics of brainbot trajectories to train neural networks to achieve complex desired trajectories, while incorporating specific interactions between multiple bots, which may possibly reveal new dynamical regimes. We plan to implement such machine learning methods in future work.

Relying on a kinematic framework has enabled us to demonstrate the first-ever on-board predictive control on a bristlebot to efficiently track a reference trajectory. Although attempts have been made in the past to introduce basic control onto bristlebots [30], these rely on complex physical modeling of the bristles. Under our framework, a model predictive control approach that runs directly on the brainbot [22, 31] has been leveraged against the kinematic model to execute the tracking of a desired trajectory. The framework is computationally light, making it suitable for onboard applications on microprocessors with low compute capability. While certain trade-offs exist, most notably the lack of precision due to the linearization of the underlying kinematic problem and the asymmetry of the bot motion, these could be addressed by further improvements in processing power and the use of more powerful solvers such as operator splitting quadratic program [32] or more efficient optimal control toolchains such as Acados [33], which relies on fast QP solvers.

Extending this framework to incorporate dynamic obstacles and inter-robot collision avoidance remains a key direction. Since staying outside of obstacle boundaries defines a nonconvex safe set, safety-critical optimization is highly challenging and computationally intensive. However, recent

work on nanoquadrotors has shown that lightweight solvers like TinyMPC [22] can successfully execute real-time on-board collision avoidance by approximating these nonconvex constraints. Adapting similar lightweight optimization algorithms or utilizing a real-time nonlinear optimizer [33] on the brainbots would enable safe navigation. Nevertheless, the demonstrated proof of concept showcases onboard control as a promising route toward the use of brainbots as experimental realizations of decentralized intelligent active particles.

To conclude, the brainbot is a convenient experimental system that is easy to fabricate and program, and versatile in the range of motion it can undergo, either autonomously or through hard-coding. Here, we have demonstrated a robust theoretical and experimental control of its motion. The combination of a physics-informed digital twin with onboard MPC closes the design-implementation loop for active particles of this class for the first time. Behaviors can be designed *in silico* against a faithful kinematic surrogate and deployed directly on the physical bot, opening a route to prescribed anomalous transport, targeted navigation, and designed collective dynamics in multibot ensembles.

#### ACKNOWLEDGMENTS

N.V. thanks the Fondation Francqui for support. F.N. thanks the Alexander von Humboldt Foundation for a postdoctoral fellowship. M.N. thanks the Belgian Federal Science Policy Office (BELSPO) for financial support within the PRODEX Programme of the European Space Agency (ESA), Contract No. 4000103267. We furthermore acknowledge the support of the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) through Project No. 416229255—SFB 1411 Design of Particulate Products (subproject D01), and the Croatian Science Foundation with the Grant No. IP-2025-02-3976.

#### DATA AVAILABILITY

The code and data that support the findings of this article are openly available on GitHub [24] and Zenodo [25].

#### APPENDIX A: EQUIVALENCE OF CONSTANT VELOCITY IN THE LABORATORY FRAME AND SINUSOIDAL VELOCITY IN THE BODY-FIXED FRAME

Suppose  $\mathbf{v}_c$  is a constant vector in the laboratory frame, given by

$$\mathbf{v}_c = v_{c_1}\hat{\mathbf{i}} + v_{c_2}\hat{\mathbf{j}}, \quad (\text{A1})$$

where  $v_{c_1}$  and  $v_{c_2}$  are constants. Eq. (A1) can be written as

$$\mathbf{v}_c = v_0(\cos\delta\hat{\mathbf{i}} + \sin\delta\hat{\mathbf{j}}), \quad (\text{A2})$$

where

$$v_0 = \sqrt{v_{c_1}^2 + v_{c_2}^2} \quad \text{and} \quad \delta = \cos^{-1}\left(\frac{v_{c_1}}{v_0}\right) = \sin^{-1}\left(\frac{v_{c_2}}{v_0}\right) \quad (\text{A3})$$

are constants. Therefore, we can write

$$\mathbf{v}_c = v_0\{\cos[\varphi(0) + \alpha]\hat{\mathbf{i}} + \sin[\varphi(0) + \alpha]\hat{\mathbf{j}}\}, \quad (\text{A4})$$

where  $\varphi(0)$  and  $\alpha \equiv \delta - \varphi(0)$  are constants.

Since  $\varphi(t) = \varphi(0) + \omega t$ , we can write Eq. (A4) as

$$\begin{aligned} \mathbf{v}_c &= v_0[\cos(\varphi(t) - \omega t + \alpha)\hat{\mathbf{i}} + \sin(\varphi(t) - \omega t + \alpha)\hat{\mathbf{j}}] \\ &= v_0[\cos(\alpha - \omega t)\cos\varphi(t)\hat{\mathbf{i}} - \sin(\alpha - \omega t)\sin\varphi(t)\hat{\mathbf{i}} \\ &\quad + \sin(\alpha - \omega t)\cos\varphi(t)\hat{\mathbf{j}} + \cos(\alpha - \omega t)\sin\varphi(t)\hat{\mathbf{j}}] \\ &= v_0[\cos(\alpha - \omega t)(\cos\varphi(t)\hat{\mathbf{i}} + \sin\varphi(t)\hat{\mathbf{j}}) \\ &\quad + \sin(\alpha - \omega t)(-\sin\varphi(t)\hat{\mathbf{i}} + \cos\varphi(t)\hat{\mathbf{j}})] \\ &= v_0[\cos(\alpha - \omega t)\hat{\mathbf{n}}_1 + \sin(\alpha - \omega t)\hat{\mathbf{n}}_2], \end{aligned} \quad (\text{A5})$$

thus showing that in the body-fixed frame,  $\mathbf{v}_c$  is a sinusoidal vector.

#### APPENDIX B: NUMERICAL INTEGRATION SCHEME USED IN THE SIMULATIONS

A general time-dependent function  $\mathbf{v}_c$  can be integrated numerically using the Crank-Nicolson scheme, accurate to the second order [33],

$$\int_t^{t+\Delta t} \mathbf{v}_c dt \approx \bar{\mathbf{v}}_c(t)\Delta t, \quad (\text{B1})$$

where  $\bar{\mathbf{v}}_c(t)$  is the mean velocity in the interval  $t$  to  $t + \Delta t$ , defined as

$$\bar{\mathbf{v}}_c(t) \equiv \left( \frac{\mathbf{v}_c(t) + \mathbf{v}_c(t + \Delta t)}{2} \right). \quad (\text{B2})$$

Similarly, we define the mean value of the angular velocity in the interval  $t$  to  $t + \Delta t$  as

$$\bar{\omega}(t) \equiv \left( \frac{\omega(t) + \omega(t + \Delta t)}{2} \right). \quad (\text{B3})$$

We reformulate Eq. (10) in terms of the unit vectors  $\hat{\mathbf{n}}_i$  while keeping  $\mathbf{v}_c$  general, thereby obtaining

$$\begin{aligned} \mathbf{r}(t + \Delta t) &= \mathbf{r}(t) + \bar{\mathbf{v}}_c\Delta t - \rho_1 A_1 [\hat{\mathbf{n}}_1(t + \Delta t) - \hat{\mathbf{n}}_1(t)] \\ &\quad - \rho_2 A_2 [\hat{\mathbf{n}}_2(t + \Delta t) - \hat{\mathbf{n}}_2(t)]. \end{aligned} \quad (\text{B4})$$

Next, we write the unit vectors in terms of a rotation matrix,

$$\hat{\mathbf{n}}(\varphi(t + \Delta t)) = \mathbf{M}(\bar{\omega}\Delta t)\hat{\mathbf{n}}(\varphi(t)) \quad (\text{B5})$$

where

$$\mathbf{M}(\bar{\omega}\Delta t) = \begin{bmatrix} \cos(\bar{\omega}\Delta t) & -\sin(\bar{\omega}\Delta t) \\ \sin(\bar{\omega}\Delta t) & \cos(\bar{\omega}\Delta t) \end{bmatrix}. \quad (\text{B6})$$

Inserting Eqs. (B5) and (B6) in Eq. (B4), we get

$$\begin{aligned} \mathbf{r}(t + \Delta t) &= \mathbf{r}(t) + \bar{\mathbf{v}}_c\Delta t + [\rho_1 A_1 \hat{\mathbf{n}}_1(t) + \rho_2 A_2 \hat{\mathbf{n}}_2(t)] \\ &\quad - \mathbf{M}(\bar{\omega}\Delta t)[\rho_1 A_1 \hat{\mathbf{n}}_1(t) + \rho_2 A_2 \hat{\mathbf{n}}_2(t)]. \end{aligned} \quad (\text{B7})$$

Replacing the terms in the square brackets with  $\mathbf{r}_c(t) - \mathbf{r}(t)$  from Eq. (4) and simplifying, we obtain

$$\mathbf{r}(t + \Delta t) = \mathbf{M}(\bar{\omega}\Delta t)[\mathbf{r}(t) - \mathbf{r}_c(t)] + \mathbf{r}_c(t) + \bar{\mathbf{v}}_c\Delta t. \quad (\text{B8})$$

Writing this equation out for two consecutive time steps  $t = k\Delta t$  and  $t + \Delta t = (k + 1)\Delta t$  yields

$$\mathbf{r}^{(k+1)} = \mathbf{M}(\bar{\omega}^{(k)}\Delta t)(\mathbf{r}^{(k)} - \mathbf{r}_c^{(k)}) + \mathbf{r}_c^{(k)} + \bar{\mathbf{v}}_c^{(k)}\Delta t. \quad (\text{B9})$$

We allow  $\rho_2$  to change from one time step to another. This requires  $\mathbf{r}_c$  to be recalculated, as detailed in Algorithm 1.

### APPENDIX C: COMPARISON OF EXPERIMENTAL AND SIMULATED TRAJECTORIES FOR DIFFERENT NUMBERS OF FOURIER MODES INCLUDED

Figure 7 provides a comparison between an experimental trajectory executed by a brainbot and several synthetic trajectories created in simulations, in which the angular velocity  $\omega$  of the brainbot is taken to be a sum of sinusoidal functions. These functions are picked from a Fourier decomposition of the experimentally obtained curve for  $\omega$  as a function of time. In Fig. 7, the number of these modes increases progressively from 1 in the leftmost column to 69 in the rightmost. As seen in the third and fourth rows of Fig. 7, the time dependence and the distribution of the  $\eta$  parameter approach the experimentally obtained values as the number of Fourier modes included increases.

### APPENDIX D: COST MATRICES FOR TINYMPC

Before implementing the TinyMPC pipeline on a physical brainbot, simulations were carried out on a virtual bot to validate the control framework. These simulations focused on solving a similar lemniscate tracking problem, replacing the external positioning setup in Fig. 6 with a simulation-based state measurement. One of the objectives of this exercise was to fine-tune the weight matrices  $Q$  and  $R$ , associated with the tracking cost and the control effect, respectively. The weights contained in these matrices are mentioned in Eq. (D1) and have been implemented in the experimental implementation of the control pipeline as

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}. \quad (\text{D1})$$

The costs for the deviation of position and velocity are penalized by weights equal to 10 and 1, respectively, while the weights associated with the acceleration (control input) are 0.1.

The matrix  $P$ , concerning the weights associated with the terminal cost, is computed internally by TinyMPC using the Riccati recursion method [28]. The weights contained in this matrix are

$$P = \begin{bmatrix} 131.3201 & 0 & 50.2887 & 0 \\ 0 & 131.3201 & 0 & 50.2887 \\ 50.2887 & 0 & 54.0187 & 0 \\ 0 & 50.2887 & 0 & 54.0187 \end{bmatrix}. \quad (\text{D2})$$

### APPENDIX E: GENERATING THE LOOKUP TABLES

The lookup tables are used to map the desired translational and angular velocities generated by the MPC to the motor commands of the bot. The bot is actuated by two motors, each of which accepts an input in the range [0, 127]. The motor

voltage scales approximately linearly with this input, with the maximum input value of 127 corresponding to around 3 V. In practice, however, inputs below 40 are insufficient to overcome the static friction and, hence, do not produce motion. Moreover, inputs in the range [40, 60] lead to highly unreliable bot behavior due to the dominant influence of static friction and stick-slip effects. For these reasons, the calibration is restricted to motor inputs in the range [60, 120], sampled in increments of 5. The preliminary experiments for calibration involve measuring the translational and angular velocities of the bot by image-based motion analysis, for all possible combinations of the motor inputs in the aforementioned range.

The resulting calibration data are then used to construct forward models relating the motor inputs to the bot's velocity and angular velocity. Bivariate polynomial regression is employed to construct these models, with a third-degree polynomial for translational velocity  $v$  and a second-degree polynomial for angular velocity  $\omega$ . The fitted models are given by

$$v = -26.5286 + 10^{-1}(3.641V_L + 6.406V_R - 0.0229V_L^2 - 0.0507V_R^2 - 0.0251V_LV_R) + 10^{-5}(V_L^3 + 1.3V_R^3 + 1.7V_LV_R^2 - 0.5V_L^2V_R), \quad (\text{E1})$$

$$\omega = -5.088834 - 0.034432V_L + 0.133761V_R - 0.000070V_L^2 - 0.000573V_R^2 + 0.000101V_LV_R, \quad (\text{E2})$$

where  $V_L$  and  $V_R$  are the left and right motor inputs. The goodness-of-fit values for  $v$  and  $\omega$  are  $R^2 = 0.92$  and  $R^2 = 0.97$ , respectively. The resulting mappings from the motor inputs to  $v$  and  $\omega$  are shown in Fig. 8.

After obtaining this forward calibration model, the inverse mapping is computed using a nonlinear least-squares procedure. Here, we consider desired translational velocities in the range  $v_{\text{command}} \in [4.8, 11.5]$  cm/s with a resolution of 0.1 cm/s and desired angular velocities in the range  $\omega_{\text{command}} \in [-3.46, 1.43]$  rad/s with a resolution of 0.01 rad/s. These limits directly reflect those of the calibration data. For each pair  $(v_{\text{command}}, \omega_{\text{command}})$ , the inverse problem is solved iteratively to determine the corresponding motor inputs  $(V_L, V_R)$ . This process yields lookup tables that map desired motion commands to motor inputs.

The resulting lookup tables are stored in the onboard memory of the bot and accessed during the control execution. At run-time, the control routine rounds the desired translational and angular velocities to the nearest discretized values in the tables (i.e., 0.1 cm/s for  $v_{\text{command}}$  and 0.05 rad/s for  $\omega_{\text{command}}$ ) and retrieves the corresponding motor commands. A clamping routine further enforces the admissible motor-command range by saturating the left and right inputs to lie between [60, 120].

[1] C. Bechinger, R. Di Leonardo, H. Löwen, C. Reichardt, G. Volpe, and G. Volpe, Active particles in complex and crowded environments, *Rev. Mod. Phys.* **88**, 045006 (2016).

[2] Z. T. Liu, Y. Shi, Y. Zhao, H. Chaté, X. qing Shi, and T. H. Zhang, Activity waves and freestanding vortices in populations of subcritical Quincke rollers, *Proc. Natl. Acad. Sci. USA* **118**, e2104724118 (2021).

- [3] R. R. Garza, N. Kyriakopoulos, Z. M. Cenev, C. Rigoni, and J. V. I. Timonen, Magnetic Quincke rollers with tunable single-particle dynamics and collective states, *Sci. Adv.* **9**, eadh2522 (2023).
- [4] V. Narayan, S. Ramaswamy, and N. Menon, Long-lived giant number fluctuations in a swarming granular nematic, *Science* **317**, 105 (2007).
- [5] C. Scholz, M. Engel, and T. Pöschel, Rotating robots move collectively and self-organize, *Nat. Commun.* **9**, 931 (2018).
- [6] C. Scholz, S. Jahanshahi, A. Ldov, and H. Löwen, Inertial delay of self-propelled particles, *Nat. Commun.* **9**, 5156 (2018).
- [7] L. Caprini, D. Breoni, A. Ldov, C. Scholz, and H. Löwen, Dynamical clustering and wetting phenomena in inertial active matter, *Commun. Phys.* **7**, 343 (2024).
- [8] Y. Chen and J. Zhang, Anomalous flocking in nonpolar granular Brownian vibrators, *Nat. Commun.* **15**, 6032 (2024).
- [9] L. Caprini, B. Liebchen, and H. Löwen, Self-reverting vortices in chiral active matter, *Commun. Phys.* **7**, 153 (2024).
- [10] O. Dauchot and V. Démery, Dynamics of a self-propelled particle in a harmonic trap, *Phys. Rev. Lett.* **122**, 068002 (2019).
- [11] P. Baconnier, D. Shohat, C. H. López, C. Coulais, V. Démery, G. Düring, and O. Dauchot, Selective and collective actuation in active solids, *Nat. Phys.* **18**, 1234 (2022).
- [12] O. Chor, A. Sohachi, R. Goerlich, E. Rosen, S. Rahav, and Y. Roichman, Many-body Szilárd engine with giant number fluctuations, *Phys. Rev. Res.* **5**, 043193 (2023).
- [13] L. Giomi, N. Hawley-Weld, and L. Mahadevan, Swarming, swirling and stasis in sequestered bristle-bots, *Proc. R. Soc. A* **469**, 20120637 (2013).
- [14] E. Zheng, M. Brandenbourger, L. Robinet, P. Schall, E. Lerner, and C. Coulais, Self-oscillation and synchronization transitions in elastoactive structures, *Phys. Rev. Lett.* **130**, 178202 (2023).
- [15] A. Altshuler, O. L. Bonomo, N. Gorohovsky, S. Marchini, E. Rosen, O. Tal-Friedman, S. Reuveni, and Y. Roichman, Environmental memory facilitates search with home returns, *Phys. Rev. Res.* **6**, 023255 (2024).
- [16] M. Noirhomme, I. Mammadli, N. Vanesse, J. Pande, A.-S. Smith, and N. Vandewalle, Brainbots as smart autonomous active particles with programmable motion, *Phys. Rev. Appl.* **23**, 064008 (2025).
- [17] F. Novkoski, M. Mélard, M. Delens, F. Wéry, M. Noirhomme, J. Pande, A. Maier, A.-S. Smith, and N. Vandewalle, GRASP-Pion: An open-source, programmable brainbot for active matter research, *Rev. Sci. Instrum.* **97**, 014704 (2026).
- [18] F. Becker, S. Boerner, V. Lysenko, I. Zeidis, and K. Zimmermann, On the mechanics of Bristle-Bots – modeling, simulation and experiments, in *ISR/Robotik 2014; 41st International Symposium on Robotics* (IEEE, Munich, Germany, 2014), pp. 1–6.
- [19] T. Majewski, D. Szwedowicz, and M. Majewski, Locomotion of a mini bristle robot with inertial excitation, *J. Mech. Robot* **9**, 061008 (2017).
- [20] H. Löwen and B. Liebchen, Towards intelligent active particles, in *Artificial Intelligence and Intelligent Matter: Nanoscience, Soft Matter, Philosophy*, edited by M. te Vrugt (Springer, Cham, 2026), pp. 257–271.
- [21] I. Mahajan and B. Plancher, Robust and efficient embedded convex optimization through first-order adaptive caching, in *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, Hangzhou, China, 2025), pp. 2984–2990.
- [22] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester, Tinympc model-predictive control on resource-constrained microcontrollers, in *2024 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, Yokohama, Japan, 2024), pp. 1–7.
- [23] J. Wikner, D. Arnström, and D. Axehill, On real-time feasibility of high-rate MPC using an active-set method on nano-quadcopters, [arXiv:2603.09342](https://arxiv.org/abs/2603.09342).
- [24] I. Mammadli, P. Shrestha, J. Pande, F. Novkoski, S. Mohapatra, M. Noirhomme, A. Maier, N. Vandewalle, and A.-S. Smith, Brainbot digital twin code GitHub, <https://github.com/GRASP-LAB/brainbot-digital-twin>.
- [25] I. Mammadli, P. Shrestha, J. Pande, F. Novkoski, S. Mohapatra, M. Noirhomme, A. Maier, N. Vandewalle, and A.-S. Smith, Physics-informed digital twin and onboard control of a brainbot for intelligent active matter [Data set], Zenodo, 2026, <https://doi.org/10.5281/zenodo.19329859>.
- [26] A. Savitzky and M. J. E. Golay, Smoothing and differentiation of data by simplified least squares procedures, *Anal. Chem.* **36**, 1627 (1964).
- [27] B. Kouvaritakis and M. Cannon, *Model Predictive Control: Classical, Robust and Stochastic* (Springer International Publishing, Switzerland, 2016).
- [28] E. Çintaş and B. Özyer, A robust fault-tolerant control algorithm for GPS-denied mini quadrotors using PID-TinyMPC and visual-inertial odometry, *Control Eng. Pract.* **169**, 106779 (2026).
- [29] R. Tedrake, Underactuated robotics (2023), <https://underactuated.csail.mit.edu/misc.html>.
- [30] S. M. Mousavi and V. Fakhari, Design and development of closed-loop controllers for trajectory tracking of a planar vibration-driven robot, *J. Vib. Control* **31**, 3691 (2025).
- [31] I. Mahajan, K. Nguyen, S. Schoedel, E. Nedumaran, M. Mata, B. Plancher, and Z. Manchester, Code generation and conic constraints for model-predictive control on microcontrollers with conic-tinyMPC, [arXiv:2403.18149](https://arxiv.org/abs/2403.18149).
- [32] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, OSQP: An operator splitting solver for quadratic programs, *Math. Program. Comput.* **12**, 637 (2020).
- [33] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, Acados—A modular open-source framework for fast embedded optimal control, *Math. Program. Comput.* **14**, 147 (2022).