# Learning a Tracking Controller for Rolling $\mu$bots

Logan E. Beaver, *Member, IEEE*, Max Sokolich, Suhail Alsalehi, Ron Weiss, Sambeeta Das, and Calin Belta, *Fellow, IEEE*

***Abstract*—Micron-scale robots ($\mu$bots) have recently shown great promise for emerging medical applications. Accurate control of $\mu$bots, while critical to their successful deployment, is challenging. In this work, we consider the problem of tracking a reference trajectory using a $\mu$bot in the presence of disturbances and uncertainty. The disturbances primarily come from Brownian motion and other environmental phenomena, while the uncertainty originates from errors in the model parameters. We model the $\mu$bot as an uncertain unicycle that is controlled by a global magnetic field. To compensate for disturbances and uncertainties, we develop a nonlinear mismatch controller. We define the *model mismatch error* as the difference between our model's predicted velocity and the actual velocity of the $\mu$bot. We employ a Gaussian Process to learn the model mismatch error as a function of the applied control input. Then we use a least-squares minimization to select a control action that minimizes the difference between the actual velocity of the $\mu$bot and a reference velocity. We demonstrate the online performance of our joint learning and control algorithm in simulation, where our approach accurately learns the model mismatch and improves tracking performance. We also validate our approach in an experiment and show that certain error metrics are reduced by up to 40%.**

***Index Terms*—Micro/nano robots, machine learning for robot control, optimization and optimal control.**

## I. Introduction

INTEREST in micron-scale robots ($\mu$bots) has grown exponentially in recent decades [1]. Medical applications have been of particular interest, including drug delivery [2], [3], biopsy [4], microsurgery [5], and cellular manipulation [6], [7], [8], [9]. Despite these advances, there are numerous challenges associated with the control of $\mu$bots. The extremely small scale of $\mu$bots incentivizes novel actuation techniques, such as electrophoretic [10], optical [11], magnetic [12], thermal [13], or by attachment to swimming microorganisms [14].

Our $\mu$bot is controlled by a rotating 3D magnetic field. The field induces a rotating moment on the $\mu$bot, which causes it to roll along the substrate surface during experiments. As a consequence, this method uses significantly less energy than other actuation methods, e.g., translating particles using strong magnetic gradients. A similar control technique has been previously used to control the micron scale "rod-bot" [15] for micron-scale manipulation. The $\mu$bot we control is spherical, non-toxic to living cells, and can be embedded within cells without damaging them [16]. This makes it an ideal candidate for emerging medical applications involving cellular manipulation. However, the small size of the $\mu$bot also implies that Brownian motion plays a significant role in its dynamics (see [17], [18]), and modeling error makes the $\mu$bot difficult to control accurately. The readers are referred to [19] for further details on different actuation techniques and motion control strategies for robots at the micron scale.

In this article, we develop a joint learning and control approach to improve the tracking capabilities of rolling $\mu$bots. A related vision-based control system to manipulate rolling $\mu$bots was presented in [20], where the authors used closed-loop visual feedback to navigate through an environment with impurities and obstacles. Similarly, [20] used a combination of PID control with a Kalman filter to guide rolling magnetic microrobots. It is important to note that there are few microrobotic control algorithms that use learning techniques, especially at scales relevant to this paper. [21] used deep reinforcement learning to enable a model microswimmer to self-learn effective locomotory gaits for translation, rotation and combined motions. [22] used analytical and reinforcement learning control strategies for path planning to a target by multiple swimmers using a uniform magnetic field. Also, [23] combined real-world artificial active particles with machine learning algorithms to explore their adaptive behavior in a noisy environment with reinforcement learning. In contrast, we propose an open-loop strategy that takes the desired velocity as an input and yields a corrected control signal that minimizes the difference between the desired and actual velocities of the $\mu$bot. As a consequence, our approach is straightforward to include as an intermediate step for receding horizon control and other closed-loop feedback strategies. Compared to related learning-based approaches, e.g., controlling swimming micron-scale robots [24], our approach is model-based and explicitly embeds the learning within the controller.

Inspired by [25], in this work we derive a controller to minimize the *nonlinear mismatch error*, that is, the error between our nonlinear $\mu$bot model and the actual dynamics. We achieve this in three steps. First, we invert an empirically derived $\mu$bot model to convert a desired velocity into a desired control action. Then, we use the learned nonlinear mismatch error and

Manuscript received 13 August 2023; accepted 13 December 2023. Date of publication 8 January 2024; date of current version 16 January 2024. This letter was recommended for publication by Associate Editor H. Liu and Editor X. Liu upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Grant GCR 2219101 and in part by the National Institute of Health under Grant R35GM147451. *(Corresponding author: Logan E. Beaver.)*

Logan E. Beaver, Suhail Alsalehi, and Calin Belta are with the Division of Systems Engineering, Boston University, Boston, MA 02215 USA (e-mail: lbeaver@odu.edu; alsalehi@bu.edu; cbelta@bu.edu).

Max Sokolich and Sambeeta Das are with the Department of Mechanical Engineering, University of Delaware, Newark, DE 29716 USA (e-mail: sokolich@udel.org; samdas@udel.org).

Ron Weiss is with the Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, MA 02142 USA (e-mail: rweiss@mit.edu).

Digital Object Identifier 10.1109/LRA.2024.3350968

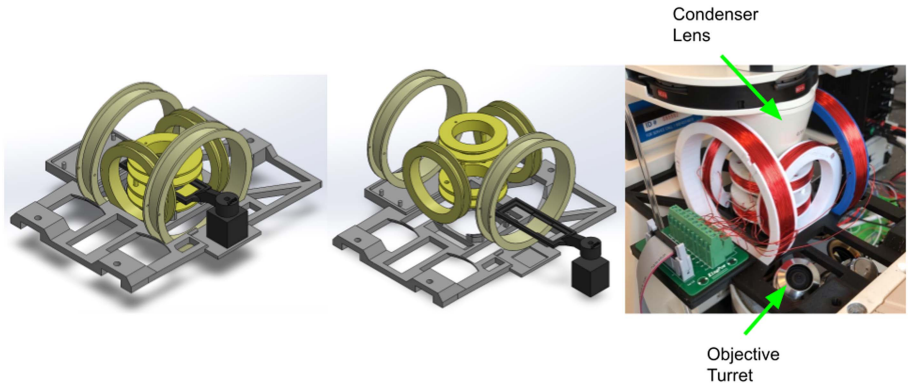


Fig. 1. Design schematics and image of the 3D Helmholtz-based System.

least-squares optimization to generate a corrected control signal. The corrected control signal exploits the learned error to minimize the difference between the desired and actual velocity of the $\mu$bot. In comparison, [25] updates the desired velocity of the system before inverting the dynamics.

The contributions of this article are as follows:

- we extend the *inverse nonlinear mismatch* approach of [25] from a 1D regression problem with linear dynamics to a 2D trajectory tracking problem;
- we derive an explicit functional form of the $\mu$bot's input-output velocity error to demonstrate that model parameter fitting is insufficient for accurate control—this also motivates the development of nonlinear control techniques;
- we derive an novel control strategy to correct the nonlinear model mismatch error by explicitly embedding a Gaussian Process regression model within a least-squares optimization problem; and
- we demonstrate improvement in the $\mu$bot's tracking capability in simulation and experiment, and we show that our online learning approach is real-time implementable.

The remainder of this article is organized as follows: We present our experimental setup in Section II and formulate the tracking problem in Section III. We present our learning approach in Section IV. Simulation and experimental results are included in Section V, and we draw conclusions and discuss future work in Section VI.

## II. Experimental Setup

In order to generate the magnetic fields necessary to actuate the rolling $\mu$bots, 6 Helmholtz coils are designed and arranged in parallel pairs as shown in Fig. 1. The coils are mounted on a Zeiss Axiovert 100 inverted microscope. To power the coils, we use an Arduino Mega micro-controller connected to a Jetson Xavier NX single board computer similar to a Raspberry Pi. The Jetson Xavier NX is capable of running a full Linux distribution with the help of a keyboard, mouse and monitor. A custom tracking and control program is written in python to read incoming images from a FLIR BFS-U3-28S5M-C USB 3.1 Blackfly S Monochrome Camera. The continuous stream of images is analyzed in Python's OpenCV library, which is used to extract position and velocity data for detected microrobots.

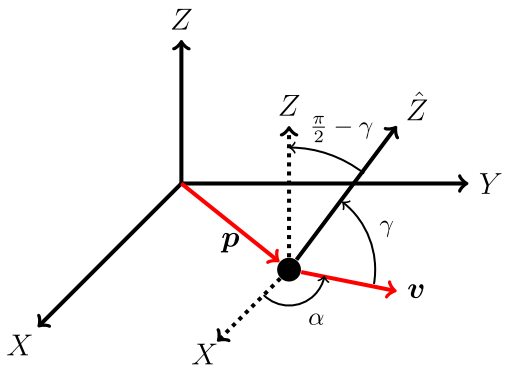


Fig. 2. Schematic illustrating the notation. The solid axes $X, Y, Z$ define the Cartesian coordinate system. The motion of the $\mu$bot is the in the $(X, Y)$-plane. The fixed attitude angle $\gamma$ is out of the plane, while the heading angle $\alpha$ is in plane. The frequency $f$ determines the $\mu$bot's forward speed along $\boldsymbol{v}$, which is in the $(X, Y)$-plane.

Action commands in the form of a heading angle $\alpha$ to steer the $\mu$bot, a constant attitude angle $\gamma$, and a frequency $f$ to set the speed at which the magnetic field rotates are sent to the Arduino over a serial communication protocol. These action/input commands are converted to a 3D rotating magnetic field. Because current is proportional to the magnetic field generated from electromagnets, the magnetic field $\boldsymbol{B} = [B_x, B_y, B_z]^T$ is mapped to the heading, attitude, and frequency signals via

$$\boldsymbol{B} = \begin{bmatrix} \cos(\gamma)\cos(\alpha)\cos(2\pi ft) + \sin(\alpha)\sin(2\pi ft) \\ -\cos(\gamma)\sin(\alpha)\cos(2\pi ft) + \cos(\alpha)\sin(2\pi ft) \\ \sin(\gamma)\cos(2\pi ft) \end{bmatrix}, \tag{1}$$

where $\gamma \in [-\pi, \pi]$ is a fixed attitude angle, $\alpha \in [-\pi, \pi]$ is the heading angle, and $f \in \mathbb{R}$ is the rolling frequency. The angles and coordinate system are depicted in Fig. 2. Note that since $\boldsymbol{B}$ is 3 Dimensional, half of the required current is sent to each pair of parallel electromagnets to achieve the desired magnetic field strength.

The $\mu$bots are constructed by plasma cleaning a plain glass slide on high for 5 minutes, wherein 24 µm paramagnetic, fluorescent microspheres (Spherotech FCM-10052-2) mixed with ethanol are drop casted and left to dry. The microspheres are coated with a 100 nm thick layer of Nickel in a dual electron

beam deposition chamber, which increases the $\mu$bot's magnetic moment. Due to the inherent surface properties of the $\mu$bot and the substrate surface, there are often very large attractive forces that result in the $\mu$bot sticking to the surface, hindering its motion. This is highly unpredictable and quite common despite adequate cleaning of the microscope slide surface. As a result, two additions were made to the experimental procedure to help reduce the likelihood of sticking. Firstly, the plasma cleaned glass slide was additionally incubated in a PFOTS (1H,1H,2H,2H-perfluorooctyltrichlorosilane) vapor at 85 °C for 30 minutes. This results in a hydrophobic surface that allows the $\mu$bot to more easily roll across the surface. Secondly, instead of suspending the $\mu$bot's in DI water, they are suspended in a $0.1\%$ solution of Sodium Dodecyl Sulfate, which is a surfactant. Although this reduces the rolling speed of the microrobot due to the increased viscosity, it significantly reduces the chances of the microrobot sticking.

## III. Problem Formulation and Approach

The $\mu$bot is a roughly spherical magnetic particle that we control using a 3D magnetic field. We control the $\mu$bot by continuously rotating the magnetic field using (1), which induces a rotational moment in the $\mu$bot and causes it to roll along the substrate surface. Varying the frequency $f$ affects the forward speed of the $\mu$bot, while varying the heading angle $\alpha$ affects its heading direction; these are depicted in Fig. 2. Based on the rolling motion of the $\mu$bot, we model it as a unicycle subject to a generalized disturbance term [26]:

$$\dot{\boldsymbol{p}} = a_0 f \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix} + \boldsymbol{D}, \tag{2}$$

where $\boldsymbol{p} \in \mathbb{R}^2$ is the position in a given reference frame, $a_0 \in \mathbb{R}_{>0}$ is an empirically determined effective radius of the $\mu$bot, and $f, \alpha$ are the $\mu$bot's rotation frequency and heading angle, respectively. Finally, $\boldsymbol{D} \in \mathbb{R}^2$ is a disturbance term that captures Brownian motion and other micron-scale disturbances. Note that we subsequently justify that 1) the control actions are identical to the heading angle and rolling frequency of (1), and 2) the frequency $f$ can be fixed in practice, and thus we consider only a single control input $\alpha$.

Our objective is to follow a reference trajectory. Let $\boldsymbol{v} = \dot{\boldsymbol{p}}$ denote the velocity of the $\mu$bot (see Fig. 2). Given a desired velocity signal $\boldsymbol{v}^d(t)$, we seek to find the optimal control input $\alpha$ such that the difference between $\dot{\boldsymbol{p}}(\alpha(t))$ and $\boldsymbol{v}^d(t)$ is minimized. This control achieves our objective given that the $\mu$bot starts on the reference trajectory. This approach is useful for high-level planners, e.g., those using RRT* and MPC, as they can generate trajectories using only the kinematic model $\dot{\boldsymbol{p}} \approx \boldsymbol{v}^d$. This decouples trajectory tracking from high-level motion planning, which is an area of ongoing research. To achieve this, we present our working assumptions for our tracking controller next.

*Assumption 1:* The environmental disturbance $\boldsymbol{D}$ and any error in our model of the true dynamics (2) are isotropic, i.e., they do not depend on the $\mu$bot's position $\boldsymbol{p}$.

*Assumption 2:* There error in aligning the $\mu$bot with the magnetic field is negligible, i.e., $\alpha$ and $f$ in (1) and (2) are identical.

Assumptions 1 and 2 simplify the learning process and are reasonable for the laboratory environment. The disturbance affecting the $\mu$bot is primarily Brownian motion, which acts uniformly at random to disturb the velocity. Assumption 1 could be relaxed by having the $\mu$bots infer hydrodynamic disturbances caused by heat, density, and chemical concentration differences, e.g., using an approach similar to [27]. In previous work, we have also found that the alignment of $\mu$bots to the global magnetic field is nearly instantaneous (see [28]), which justifies Assumption 2.

*Assumption 3:* The $\mu$bot is controlled to roll at a fixed rate, i.e., $f(t)$ is a known constant selected a priori.

Assumption 3 does not affect the derivation of our controller, as we only employ it when training our machine learning algorithm and solving the least-squares optimization. Relaxing this assumption increases the amount of training data and learning time, but not prohibitively so. Furthermore, operating $\mu$bots with a constant rolling frequency is common practice [16].

Our technical approach is as follows: First, we parameterize the $\mu$bot's dynamics with an approximate model, and we derive the functional form of the model error. We show that the domain of the model error function is a subset of the state space, which we use as the features (inputs) for a machine learning algorithm. After learning the model error, we employ least-squares optimization to minimize the difference between the predicted and actual velocity of the $\mu$bot. Note that we do not minimize the predicted velocity error directly. Instead, we adjust the control signal sent to the $\mu$bot to compensate for the model error indirectly.

## IV. Nonlinear Mismatch Controller

The unicycle model satisfies the property of *differential flatness* with the output variable $\boldsymbol{p}$, that is, we can change the coordinates of our unicycle dynamics (2) to only consider the variable $\boldsymbol{p}$ and its derivative $\dot{\boldsymbol{p}} := \boldsymbol{v}$ (see [29]). Furthermore, while the frequency $f$ is fixed under Assumption 3, it is straightforward to relax this assumption for our analysis. In fact, allowing a variable frequency $f$ only introduces computational complexity while training the machine learning model and solving the least-squares optimization problem. The mapping for our rolling dynamics is

$$\alpha = \arctan\left(\frac{v_y - D_y}{v_x - D_x}\right),$$
$$f = \frac{||\boldsymbol{v} - \boldsymbol{D}||}{a_0}, \tag{3}$$

where $\boldsymbol{D} = [D_x \; D_y]$ and $\boldsymbol{v} = [v_x \; v_y]$. In reality we do not know the actual value of $a_0$, nor do we know the stochastic disturbance $\boldsymbol{D}$. Thus, we denote our approximate model parameters using $\hat{\cdot}$. In particular, $\hat{a}_0$ is a constant scalar that estimates $a_0$ and $\hat{\boldsymbol{D}}$ is a constant vector that estimates the disturbance $\boldsymbol{D}$. With these estimates, it is possible to convert a desired velocity into a control signal using (3), i.e.,

$$\alpha^d = \arctan\left(\frac{v_y^d - \hat{D}_y}{v_x^d - \hat{D}_x}\right),$$

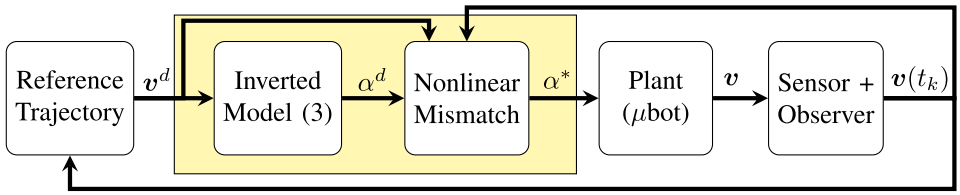


Fig. 3. Control block diagram showing how our proposed system (yellow box) transforms the desired velocity signal ($\boldsymbol{v}^d$) into a heading angle ($\alpha^*$) such that the difference between $\boldsymbol{v}^d$ and $\boldsymbol{v}$ is minimized.

$$f^d = \frac{||\boldsymbol{v}^d - \hat{\boldsymbol{D}}||}{\hat{a}_0}. \tag{4}$$

This enables us to convert the desired velocity $\boldsymbol{v}^d$, which is defined on a Cartesian basis, into control actions for the $\mu$bot. However, due to the inherent inaccuracies of our model, naïvely applying the control input $\alpha^d$ leads to tracking error. Substituting the control signals generated by the approximate model (4) into the dynamics (2) yields the closed-loop velocity of the $\mu$bot,

$$\boldsymbol{v} = \frac{a_0}{\hat{a}_0}\boldsymbol{v}^d + \boldsymbol{D} - \frac{a_0}{\hat{a}_0}\hat{\boldsymbol{D}}. \tag{5}$$

Equivalently, (5) shows the final $\mu$bot velocity after feedback linearizing with an approximate model. Note that this contains the nonlinear product of $\frac{a_0}{\hat{a}_0}$ and $\hat{\boldsymbol{D}}$. This explains why a model parameter estimation alone is insufficient to achieve a desired trajectory, as the error in our model parameters is amplified by this nonlinearity.

To enhance our ability to track the desired trajectory beyond parameter estimation, we follow the approach outlined by [25] and explicitly define a velocity error $\boldsymbol{v}^e$,

$$\boldsymbol{v}^e := \boldsymbol{v} - \boldsymbol{v}^d, \tag{6}$$

which, using (5), we can write in closed form as

$$\boldsymbol{v}^e = \boldsymbol{v}^d\left(\frac{a_0}{\hat{a}_0} - 1\right) + \boldsymbol{D} - \frac{a_o}{\hat{a}_o}\hat{\boldsymbol{D}}. \tag{7}$$

Note that (4) enables us to freely convert between $\boldsymbol{v}^d$ and $\alpha^d, f^d$. Thus, the right hand side of (7) is a function of $\alpha^d$, $f^d$, the domain of $\boldsymbol{D}$, and the domain of $\hat{\boldsymbol{D}}$. This implies that, under Assumptions 1–2, we can completely capture the behavior of the velocity error as some function $\boldsymbol{v}^e$ using machine learning with $\alpha$ and $f$ as the only features.

Note that under Assumption 3, we can also neglect the dependence of $\boldsymbol{v}_e$ on $f$; this is not a technical limitation, and can easily be relaxed in applications where $f$ is not constant. Next, we approximate $\boldsymbol{v}^e$ using Gaussian Process (GP) regression. We train the GP online using experimental data, where the Cartesian axes of $\boldsymbol{v}^e$ are each captured by a GP. In particular, we use (6) to generate training data by applying a known sequence of control inputs; we discuss this process further in Section IV-B.

A GP is completely defined by its mean $\mu(\alpha)$ and kernel (or covariance) $K(\alpha, \alpha')$ functions for two features $\alpha, \alpha'$. The prior of the mean is generally zero, while the kernel describes a statistical distribution over a function space. For accurate regression, the kernel should be a basis for the underlying function $\boldsymbol{v}^e(\alpha)$. After learning the velocity error, the GP takes the heading angle $\alpha$ as an input and produces a Gaussian distribution over each component of $\boldsymbol{v}^e$. The mean of this distribution predicts the expected value of $\boldsymbol{v}^e$ and the standard deviation predicts the uncertainty in the velocity. Compensating for the nonlinearity of $\boldsymbol{v}^e$ is the basis for our *nonlinear mismatch controller*, which we present next. A control block diagram of our control approach is shown in Fig. 3.

### A. Optimization-Based Controller

To construct our optimization-based controller, we start by replacing the desired velocity $\boldsymbol{v}^d$ in our error dynamics (6) with our model. This yields the actual velocity of the $\mu$bot for any given heading angle $\alpha$ in the form:

$$\hat{a}_0\, f \begin{bmatrix}\cos(\alpha)\\ \sin(\alpha)\end{bmatrix} + \hat{\boldsymbol{D}} + \boldsymbol{v}^e(\alpha) = \boldsymbol{v}. \tag{8}$$

Next, we replace the unknown velocity error $\boldsymbol{v}^e$ with the mean prediction from the GP. This assumes that the GP has sufficiently learned the velocity error function $\boldsymbol{v}_e$, which yields

$$\hat{a}_0\, f \begin{bmatrix}\cos(\alpha)\\ \sin(\alpha)\end{bmatrix} + \hat{\boldsymbol{D}} + \boldsymbol{\mu}(\alpha) = \boldsymbol{v}(\alpha), \tag{9}$$

where $\boldsymbol{\mu}$ is the GP's estimate for each component of the velocity error. To minimize the velocity error of our $\mu$bot, we perform a least squares minimization of (9) from the desired velocity, i.e.,

$$\min_{\alpha} ||\boldsymbol{v}(\alpha) - \boldsymbol{v}^d||^2. \tag{10}$$

Applying Assumptions 1–3 and expanding (10) yields a one-dimensional least-squares cost function

$$\begin{aligned} J(\alpha) = &\left(\hat{a}_0 f\cos(\alpha) + \mu_x(\alpha) + \hat{D}_x - v_x^d\right)^2 \\ &+ \left(\hat{a}_0 f\sin(\alpha) + \mu_y(\alpha) + \hat{D}_y - v_y^d\right)^2. \end{aligned} \tag{11}$$

Expanding the cost function and applying the Pythagorean identity yields the following least-squares optimization problem

$$\begin{aligned} \min_{\alpha\in[-\pi,\pi]} &(\hat{a}_0 f)^2 + ||\boldsymbol{\mu}(\alpha) + \hat{\boldsymbol{D}} - \boldsymbol{v}^d||^2 \\ &+ 2\hat{a}_0 f\left(\boldsymbol{\mu}(\alpha) + \hat{\boldsymbol{D}} - \boldsymbol{v}^d\right)\cdot\begin{bmatrix}\cos(\alpha)\\ \sin(\alpha)\end{bmatrix}, \end{aligned} \tag{12}$$

which is a differentiable scalar optimization problem over a compact set.

### B. Online Learning

We train the GP during an initial *learning phase*, where the $\mu$bot is given a sequence of control inputs, either from a human operator or open-loop control sequence. The learning phase occurs in the same environment and with the same robots as the experiment; thus the training and testing environments are consistent. We collect position and control action data for the $\mu$bot at discrete time steps $t_k$; we denote the position data by $\mathcal{P} = \{\boldsymbol{p}(t_k)\}$ and action data as $\mathcal{X} = \{\alpha(t_k)\}$.

We use these data to numerically derive the inputs and outputs of the Gaussian Process model of (7), i.e., the control action and the velocity error, respectively. We calculate the actual velocity $\boldsymbol{v}(t_k)$ by taking a numerical derivative of $\mathcal{P}$ and passing the result through a low-pass filter; this yields the actual velocity of the $\mu$bot at each step, which we store in the set $\mathcal{V} = \{\boldsymbol{v}(t_k)\}$.

Once the data is collected, we estimate the model parameters and desired velocity as follows. First, we estimate $\boldsymbol{D}$ by applying a control input of $f(t) = \alpha(t) = 0$, which yields

$$\dot{\boldsymbol{p}} = \boldsymbol{D}. \tag{13}$$

Taking the expectation of both sides yields the mean disturbance

$$\frac{1}{|\mathcal{V}|} \sum_{\boldsymbol{v}(t_k)\in\mathcal{V}} ||\boldsymbol{v}(t_k)|| = \mathbb{E}\left[\boldsymbol{D}\right] = \hat{\boldsymbol{D}}, \tag{14}$$

where $|\cdot|$ is set cardinality. Note that our estimate $\hat{\boldsymbol{D}}$ is a constant parameter that comes from a linear regression; the stochasticity of $\boldsymbol{D}$ is captured by the Gaussian process that approximates (7).

Next, we determine $\hat{a}_0$ using data from an open loop control sequence; taking the expectation of (2) and squaring both sides yields

$$\mathbb{E}\left[||\boldsymbol{v} - \hat{\boldsymbol{D}}||\right]^2 = a_0^2 f^2. \tag{15}$$

Substituting the expectation of $\boldsymbol{v}$ with the experimental data, re-arranging, and taking the square root of both sides yields the best statistical estimate for $\hat{a}_0$:

$$\hat{a}_0 = \frac{1}{|\mathcal{V}|} \sum_{\boldsymbol{v}(t_k)\in\mathcal{V}} \frac{||\boldsymbol{v}(t_k) - \hat{\boldsymbol{D}}||}{f}. \tag{16}$$

Finally, the resulting set of velocity errors is

$$\mathcal{Y} = \left\{\boldsymbol{v}^e(t_k) \,:\, \boldsymbol{v}^e = \boldsymbol{v}(t_k) - \boldsymbol{v}^d(t_k)\right\}, \tag{17}$$

where $\boldsymbol{v}^d(t_k)$ is the desired velocity of the $\mu$bot at each time $t_k$. We use (17) in conjunction with our empirical model (3) to generate the velocity error and control action at each time step. With this data, we compute a posterior distribution on the mean and standard deviation of the GP to determine the expected velocity error and its uncertainty for each control input.

## V. Experimental Results

We validated our learning approach *in silico* and *in situ*[1]; we present our simulation results in Subsection V-A and experimental findings in Subsection V-B. In both cases, we first perform an online learning step, where we apply a pre-computed control input to generate training data. Then, to validate our learning approach, we apply a pre-defined sequence of control actions in open-loop with and without the Nonlinear Mismatch module (Fig. 3).

Note that we do training and testing in the same environment, thus we do not consider issues that may arise from policy transfer or environmental inconsistency. This procedure yields the *corrected* and *baseline* cases, respectively, which we use to explicitly quantify the impact of our approach.

We implemented our GP approach using the Scikit-Learn toolbox (see [30]) for Python3, which provides an API to easily select a large number of kernels and train the GP. Scikit-learn also automatically optimizes the kernel hyperparameters during training, which provided insights for kernel selection. In particular, some hyperparameters for the rational quadratic, Matern, and periodic kernels grew arbitrarily small during training, which implies that these kernels include extraneous dynamics that do not describe the true behavior of the rolling $\mu$bot's velocity error. We found that a linear combination of a radial basis function and white noise in the form

$$K(\alpha, \alpha') = \exp \frac{||\alpha - \alpha'||^2}{2\sigma} + \eta \tag{18}$$

yielded a kernel that adequately captured the velocity error of the rolling $\mu$bot. In the equation above, $\sigma$ is a length hyperparameter and $\eta$ is drawn from a normal distribution where the mean is zero and the variance is another hyperparameter.

### A. In Silico Experiment

We developed a $\mu$bot simulator as an *OpenAI Gym*[2] environment in Python3. We implemented two simulation modes using a '*model-mismatch*' flag, which disturbs the model parameters and adds stochastic zero-mean noise to mimic a physical experiment. Omitting this flag uses the exact model parameters with no noise to generate the desired system trajectory. We implemented the learning approach of Section IV as follows. First, we applied zero input over 100 time steps (3 seconds) with the 'model-mismatch' flag to estimate the mean disturbance using (14). Next, we generated a sequence of control inputs that swept the entire control domain $\alpha \in [-\pi, \pi]$ three times over 1800 time steps (60 seconds) with the 'model-mismatch' flag, which produced our training data. In training, we estimated $\hat{a}_0$ and updated the GPs using (16) and (17), respectively. Finally, to validate our approach, we performed three experiments in silico; 1) we generated the desired trajectory without the 'model-mismatch' flag, 2) we generated the *baseline* trajectory by repeating the experiment with the 'model-mismatch' flag enabled, and 3) we updated the reference control inputs using (12) to generate the *corrected* trajectory with the 'model mismatch' flag.

Fig. 4 shows the resulting desired, baseline, and corrected trajectories overlaid for 100 trials with the same initial state. While the learning component significantly improves the velocity tracking, it is unable to completely compensate for the

[1]Videos of the experiments and supplemental material are available online: https://sites.google.com/udel.edu/l4ub

[2]For more information on the Gym environment see: https://github.com/openai/gym

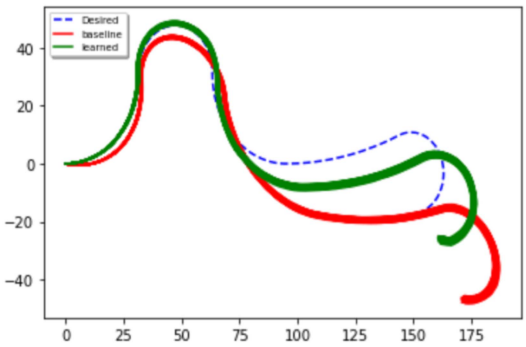


Fig. 4. Desired, (blue, dashed), corrected (green), and baseline (orange) trajectories from 100 different trials of the *in silico* experiment.

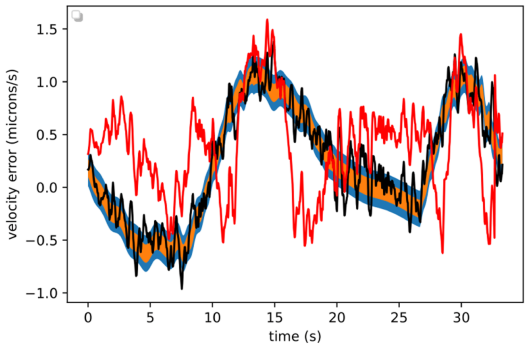


Fig. 5. GP's prediction of the $x$-axis velocity errors at each time step; the orange band corresponds to one standard deviation (65%), and the blue band corresponds to two standard deviations (95%) of uncertainty. The black line is the actual velocity error, and the red line is the neural network approximation.

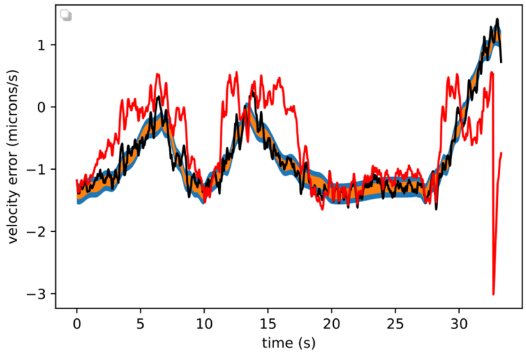


Fig. 6. GP's prediction of the $y$-axis velocity errors at each time step; orange band corresponds to one standard deviation (65%), and the blue band corresponds to two standard deviations (95%) of uncertainty. The black line is the actual velocity error, and the red line is the neural network approximation.

model mismatch–even in an environment with no noise. The velocity error estimate for one trial is presented in Figs. 5 and 6, which demonstrates that the GP has captured a reasonably good estimate of the velocity error at each time step. This implies that the nonlinear mismatch approach is unable to achieve perfect tracking for the system, which is likely related to the reachability of the system's dynamics (9). This stems from correcting the $x$ and $y$ components of the velocity error while only controlling $\alpha$.

As a comparison with existing methods, we also trained an ensemble of neural networks (NNs) to learn the $\mu$bot model.

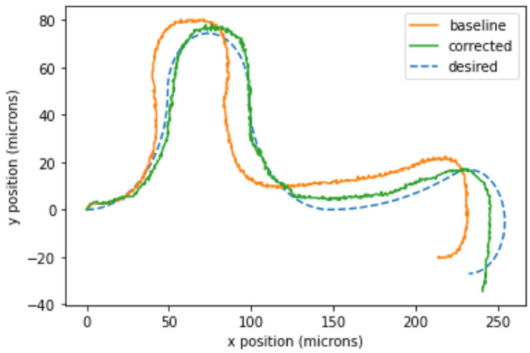


Fig. 7. Comparison of the baseline (orange), corrected (green), and desired (blue, dashed) position trajectories from the *in situ* experiment.

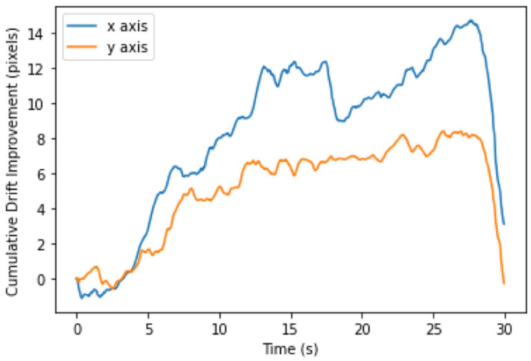


Fig. 8. Cumulative drift of the $\mu$bot in the $x$ and $y$ directions for the *in situ* experiment. The baseline has a value of 0, and positive numbers are an improvement in performance.

Using the same data as the GP training, we trained 5 NNs that had one hidden layer with either $10, 20, 50, 75$ or 100 nodes. The NNs took the control actions as an input and predicted the velocity of the $\mu$bot; each of the networks was given randomly initialized weights with ReLU activation functions. We found the network with 75 nodes in the hidden layer had the best testing performance, and the error in model prediction is shown in Figs. 5 and 6. While the NN is able to capture the trends in the velocity error, the model prediction is significantly worse than the Gaussian process–and the NN does not produce confidence intervals to capture the stochasticity of the system. This result is intuitive, as the GP learns a single model mismatch term, whereas the neural network models the full $\mu$bot dynamics. While it may be possible to increase NN performance with additional data and a larger network architecture, such an analysis is beyond the scope of this article.

### B. *In Situ Experiment*

We repeated the same procedure from Section V-A using 24um $\mu$bots over 300 time steps (10 seconds) at the experimental facility at the University of Delaware as described in Section II. The resulting desired, baseline, and corrected trajectories are presented in Fig. 7, and the drift error for the baseline and corrected cases is shown in Fig. 8. Photos of an experiment with the trajectories overlaid are presented in Fig. 9.

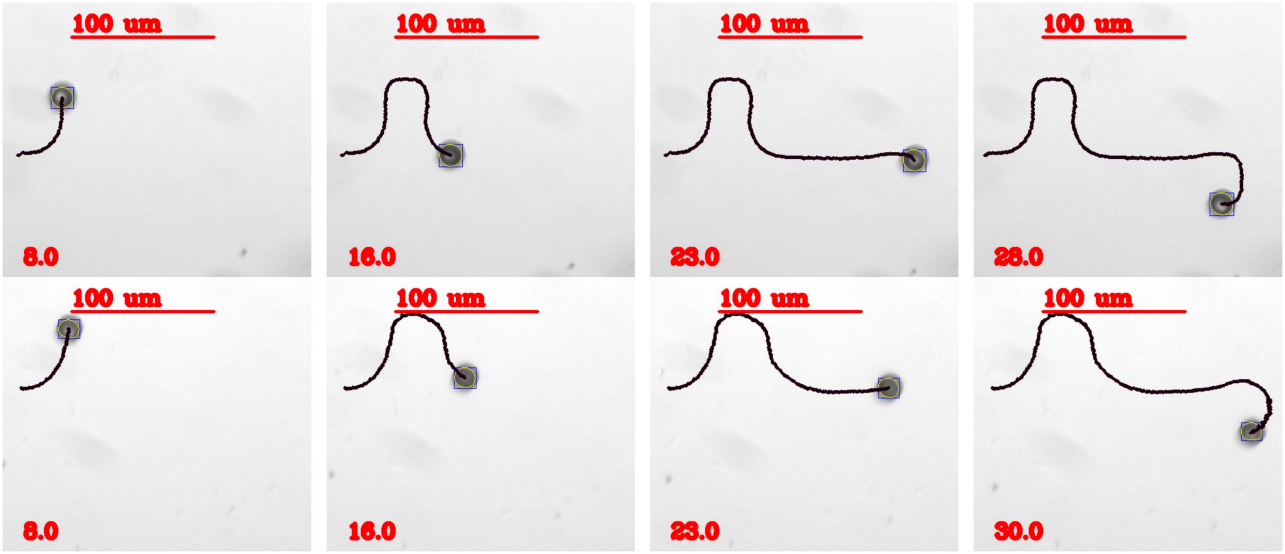


Fig. 9. Photographs of a separate set of *baseline* (top) and *corrected* (bottom) experiments, taken approximately 8 seconds apart. The $\mu$bot history is overlaid.

TABLE I
ERRORS FOR THE *IN SITU* $\mu$BOT EXPERIMENT; THE MEDIAN USES THE ABSOLUTE VALUE OF THE ERROR

| | Baseline | Corrected | Improvement |
|---|---|---|---|
| Final Position Error | 12.56 microns | 7.14 microns | 43 % |
| Median $v_x$ Error | 1.19 microns/s | 0.72 microns/s | 40 % |
| Median $v_y$ Error | 0.80 microns/s | 0.61 microns/s | 23 % |

Fig. 7 shows significant improvement in the $\mu$bot's ability to track the open-loop trajectory. We translated the normalized and corrected position data to the origin to compare it with the desired trajectory. This shows a significant improvement in the $\mu$bot's position trajectory tracking, which comes from a combination of our improved tracking controller and random disturbances. In other words, integrating the $\mu$bot's velocity after applying our corrected control signal yields less error than the uncorrected case. This is shown explicitly in Fig. 8, which depicts the $\mu$bot's drift throughout the experiment.

To calculate the $\mu$bot's drift, we subtracted the desired and actual velocity along each axis to calculate the velocity error. Next, we performed a cumulative trapezoidal integration on the norm of the velocity error. This yields the worst-case scenario for how far the $\mu$bot could drift from the reference trajectory. As a result, drift was reduced by at least 6 pixels (3.7 microns) along each axis for the majority of the experiment.

The median velocity error along each axis is presented in Table I, along with the error in the $\mu$bot's final position for each case. These results show that despite the poor tracking in the final 3 seconds, our learning controller significantly reduces the drift of the $\mu$bot by matching the desired open-loop control policy and brings the $\mu$bot closer to the desired final position. Due to the nature of the experimental environment, it is not uncommon for unexpected disturbances, such as stiction, debris, and nearby magnetic particles, to disturb the $\mu$bot's trajectory in a way that our tracking controller cannot compensate for. These exogenous factors are the source of error in the last 3 seconds of the *corrected* experiment.

## VI. CONCLUSION

We developed a nonlinear mismatch controller to improve the performance of a tracking controller in $2D$. We motivated the use of nonlinear mismatch over a parameter estimation scheme, and we proposed a least-squares based optimization problem to minimize the tracking error. Finally, we demonstrated in simulation and experiments that our approach significantly improves the tracking performance of rolling $\mu$bots.

Future work includes relaxing Assumption 3 and including $f$ as parameter in the model mismatch. Deriving guarantees on the resulting velocity error using fixed-point analysis is another interesting research direction; employing the GP's uncertainty estimate as a measure of robustness in a high-level planner may also yield useful insights. Embedding our low-level controller inside of an MPC path planner to avoid undesired collisions with cells and counteract Brownian diffusion in-situ is another critical next step for this work. Finally, expanding our approach to control multiple $\mu$bots simultaneously would advance the state of the art, and bring us one step closer to solving fundamental challenges in emerging medical applications.

## REFERENCES

[1] T. Honda, K. I. Arai, and K. Ishiyama, "Micro swimming mechanisms propelled by external magnetic fields," *IEEE Trans. Magn.*, vol. 32, no. 5, pp. 5085–5087, Sep. 1996.
[2] M. Sitti et al., "Biomedical applications of untethered mobile milli/microrobots," *Proc. IEEE*, vol. 103, no. 2, pp. 205–224, Feb. 2015.
[3] J. Troccaz and R. Bogue, "The development of medical microrobots: A review of progress," *Ind. Robot: An Int. J.*, vol. 35, no. 4, pp. 294–299, 2008.
[4] C. Bárcena, A. K. Sra, and J. Gao, "Applications of magnetic nanoparticles in biomedicine," in *Nanoscale Magnetic Materials and Applications*. Boston, MA, USA: Springer, 2009, pp. 591–626.

[5] S. Guo and Q. Pan, "Mechanism and control of a novel type microrobot for biomedical application," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2007, pp. 187–192.
[6] M. S. Sakar, E. B. Steager, A. Cowley, V. Kumar, and G. J. Pappas, "Wireless manipulation of single cells using magnetic microtransporters," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 2668–2673.
[7] E. W. H. Jager, O. Inganäs, and I. S. Lundström, "Microrobots for micrometer-size objects in aqueous media: Potential tools for single-cell manipulation," *Science*, vol. 288, no. 5475, pp. 2335–2338, 2000.
[8] S. Kim et al., "Fabrication and characterization of magnetic microrobots for three-dimensional cell culture and targeted transportation," *Adv. Mater.*, vol. 25, no. 41, pp. 5863–5868, 2013.
[9] E. B. Steager, M. S. Sakar, C. Magee, M. Kennedy, A. Cowley, and V. Kumar, "Automated biomanipulation of single cells using magnetic microrobots," *Int. J. Robot. Res.*, vol. 32, no. 3, pp. 346–359, 2013.
[10] H. Kim and M. J. Kim, "Electric field control of bacteria-powered microrobots using a static obstacle avoidance algorithm," *IEEE Trans. Robot.*, vol. 32, no. 1, pp. 125–137, Feb. 2016.
[11] D. Palima and J. Glückstad, "Gearing up for optical microrobotics: Micromanipulation and actuation of synthetic microstructures by optical forces," *Laser Photon. Rev.*, vol. 7, no. 4, pp. 478–494, 2013.
[12] S. Chowdhury, W. Jing, and D. J. Cappelleri, "Towards independent control of multiple magnetic mobile microrobots," *Micromachines*, vol. 7, no. 1, 2016, Art. no. 3.
[13] E. Y. Erdem et al., "Thermally actuated omnidirectional walking microrobot," *J. Microelectromech. Syst.*, vol. 19, no. 3, pp. 433–442, Jun. 2010.
[14] B. Behkam and M. Sitti, *Bacteria Integrated Swimming Microrobots (Lecture Notes in Computer Science Series)*, vol. 4850. New York, NY, USA: Springer, 2007.
[15] R. Pieters, H.-W. Tung, S. Charreyron, D. F. Sargent, and B. J. Nelson, "RodBot: A rolling microrobot for micromanipulation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 4042–4047.
[16] D. Rivas, S. Mallick, M. Sokolich, and S. Das, "Cellular manipulation using rolling microrobots," in *Proc. IEEE Int. Conf. Manipulation, Automat. Robot. Small Scales.*, 2022, pp. 1–6.
[17] S. Das, E. B. Steager, M. A. Hsieh, K. J. Stebe, and V. Kumar, "Experiments and open-loop control of multiple catalytic microrobots," *J. Micro-Bio Robot.*, vol. 14, no. 1/2, pp. 25–34, 2018.
[18] B. M. Vinagre, I. Tejado, and J. E. Traver, "There's plenty of fractional at the bottom, I: Brownian motors and swimming microrobots," *Fractional Calculus Appl. Anal.*, vol. 19, no. 5, 2016, Art. no. 1282.
[19] T. Xu, J. Yu, X. Yan, H. Choi, and L. Zhang, "Magnetic actuation based motion control for microrobots: An overview," *Micromachines*, vol. 6, no. 9, pp. 1346–1364, 2015.
[20] X. Tang, Y. Li, X. Liu, D. Liu, Z. Chen, and T. Arai, "Vision-based automated control of magnetic microrobots," *Micromachines*, vol. 13, no. 2, 2022, Art. no. 337.
[21] Z. Zou, Y. Liu, Y.-N. Young, O. S. Pak, and A. C. Tsang, "Gait switching and targeted navigation of microswimmers via deep reinforcement learning," *Commun. Phys.*, vol. 5, no. 1, 2022, Art. no. 158.
[22] L. Amoudruz and P. Koumoutsakos, "Independent control and path planning of microswimmers with a uniform magnetic field," *Adv. Intell. Syst.*, vol. 4, no. 3, 2022, Art. no. 2100183.
[23] S. Muiños-Landin, A. Fischer, V. Holubec, and F. Cichos, "Reinforcement learning with artificial microswimmers," *Sci. Robot.*, vol. 6, no. 52, 2021, Art. no. eabd9285.
[24] M. R. Behrens and W. C. Ruder, "Smart magnetic microrobots learn to swim with deep reinforcement learning," *Adv. Intell. Syst.*, vol. 4, no. 10, 2022, Art. no. 2200023.
[25] M. Greeff and A. P. Schoellig, "Exploiting differential flatness for robust learning-based tracking control using Gaussian processes," *IEEE Control Syst. Lett.*, vol. 5, no. 4, pp. 1121–1126, Oct. 2021.
[26] L. Yang, Y. Zhang, Q. Wang, K.-F. Chan, and L. Zhang, "Automated control of magnetic spore-based microrobot using fluorescence imaging for targeted delivery with cellular resolution," *IEEE Trans. Automat. Sci. Eng.*, vol. 17, no. 1, pp. 490–501, Jan. 2020.
[27] D. Chang, W. Wu, C. R. Edwards, and F. Zhang, "Motion tomography: Mapping flow fields using autonomous underwater vehicles," *Int. J. Robot. Res.*, vol. 36, no. 3, pp. 320–336, 2017.
[28] L. E. Beaver, B. Wu, S. Das, and A. A. Malikopoulos, "A first-order approach to model simultaneous control of multiple microrobots," in *Proc. IEEE Int. Conf. Manipulation, Automat. Robot. Small Scales*, 2022, pp. 1–7.
[29] H. Sira-Ramirez and S. K. Agrawal, *Differentially Flat Systems*, 1st ed. Boca Raton, FL, USA: CRC Press, 2018.
[30] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.