Learning for Control of Rolling μbots

Logan E. Beaver[1], Max Sokolich[2], Suhail Alsalehi[1], Ron Weiss[3], Sambeeta Das[2], Calin Belta[1*]

[1*]Division of Systems Engineering, Boston University, Boston, 02215, MA, USA.
[2]Department of Mechanical Engineering, University of Delaware, Newark, 19716, DE, USA.
[3]Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, 02142, MA, USA.

*Corresponding author(s). E-mail(s): cbelta@bu.edu;
Contributing authors: lebeaver@bu.edu; sokolich@udel.edu;
alsalehi@bu.edu; rweiss@mit.edu; samdas@udel.edu;

Abstract

Micron-scale robots ($\boldsymbol{\mu}$bots) have recently shown great promise for emerging medical applications, and accurate control of $\boldsymbol{\mu}$bots is a critical next step to deploying them in real systems. In this work, we develop the idea of a nonlinear mismatch controller to compensate for the mismatch between the disturbed unicycle model of a rolling $\boldsymbol{\mu}$bot and trajectory data collected during an experiment. We exploit the differential flatness property of the rolling $\boldsymbol{\mu}$bot model to generate a mapping from the desired state trajectory to nominal control actions. Due to model mismatch and parameter estimation error, the nominal control actions will not exactly reproduce the desired state trajectory. We employ a Gaussian Process (GP) to learn the model mismatch as a function of the desired control actions, and correct the nominal control actions using a least-squares optimization. We demonstrate the performance of our online learning algorithm in simulation, where we show that the model mismatch makes some desired states unreachable. Finally, we validate our approach in an experiment and show that certain error metrics are reduced by up to **40%**.

Keywords: Microrobot, Gaussian Process, Magnetic Actiation, Online Learning

1 Introduction

Interest in micron-scale robots (μbots) 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 [6–9]. Despite these advances, there are numerous challenges associated with the control of μbots. The extremely small scale of μbots incentivizes novel actuation techniques, such as electrophoretic actuation [10], optical actuation [11], magnetic field actuation [12], thermal actuation [13], or by attachment to swimming microorganisms [14]. The small size of μbots also implies that Brownian motion plays a significant role in their dynamics (see [15, 16]).

In this article, we explore the dynamic behavior of rolling μbots (see [17, 18] for further details); our μbots consist of microspheres coated with strips of nickel, which cause them to rotate and align with an externally applied magnetic field. The 3D magnetic field system induces a rotating moment on the μbot, which causes it to roll along the substrate surface during experiments. Using a magnetic field to induce rolling requires significantly less energy than other actuation methods, e.g., translating particles using strong magnetic gradients. Our μbots are non-toxic to living cells, and can be embedded within cells without damaging them (see [17]). This makes the rolling μbots an ideal candidate for emerging medical applications involving cellular manipulation.

A similar vision-based control system to manipulate rolling μbots was presented in [19], where the authors use visual feedback to navigate through an environment with impurities and obstacles. In contrast, we propose an open-loop strategy that pre-processes a reference control signal to minimize tracking error. Thus, our approach is easily extensible to receding horizon control and other closed-loop feedback strategies that periodically re-plan to compensate for drift. Another system for manipulation and control with the micron scale "rod-bot" was introduced in [20]. In that work, a rod-shaped robot was magnetized to roll subject to a global magnetic field. A learning-based approach to controlling swimming micron-scale robots is discussed in [21]. The swimming robot has a corkscrew-like shape, which propels it through the fluid environment. The authors use a neural network to learn a set of control actions that drive the agent in a circle; in contrast, we propose a model-based learning technique that compensates for model mismatch and parameter estimation errors. A recent survey [22] further details different actuation techniques and resulting motion control strategies for magnetically-actuated micron-scale robots.

Inspired by [23], in this work we derive a *nonlinear mismatch* controller to compensate for model mismatch and disturbances in the tracking controller for an individual μbot. This technique can be used with feedback linearizable and differentially flat systems, where a nominal model is used to generate control actions based on a reference trajectory. In addition to learning the model parameters from data, our nonlinear mismatch approach learns the model's error as a function of the state and control variables. Unlike the approach of [23], we correct the control signal generated by our model rather than updating the reference state. We also show that this minimizes the tracking error of the μbots, which implies significantly better performance compared to parameter estimation alone.

In addition to the novelty of our experimental testbed and application, we make the following contributions to learning for control systems:

- we adapt the *inverse nonlinear mismatch* approach of [23] from a 1D LQR minimization to a *nonlinear mismatch* approach for the 2D tracking problem,
- we derive an explicit functional form of the μbot's velocity error, which demonstrates the nonlinear effect of model mismatch error on the dynamics,
- we explicitly correct the model mismatch through a least-squares optimization, and
- we demonstrate improvement in the μbot's tracking capability in simulation and experiment, and show that our online learning approach is real-time implementable.

The remainder of this article is organized as follows: We formulate the tracking problem in Section 2 and present our learning approach in Section 3. Simulation and experimental results are included in Section 4, and we draw conclusions and discuss future work in Section 5. Finally, information on the experimental testbed and μbot fabrication is presented in the Appendix.

2 Problem Formulation and Approach

We model the μbot as a unicycle that navigates a 2D environment by rolling along a planar surface subject to a disturbance term,

$$\dot{\boldsymbol{p}} = a_0 f \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix} + \boldsymbol{D}, \tag{1}$$

where $\boldsymbol{p} \in \mathbb{R}^2$ is the position, $a_0 \in \mathbb{R}_{>0}$ is an empirically determined constant, $f, \alpha \in \mathbb{R}$ are the control inputs that determine the robot's rolling frequency and heading, respectively, and $\boldsymbol{D} \in \mathbb{R}^2$ is a disturbance term. This is illustrated in Fig. 1, along with microscope images of the μbot.

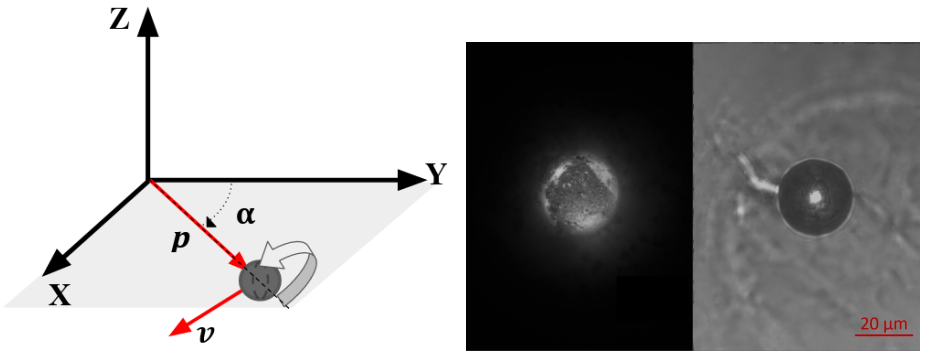

Fig. 1 Left to right: Schematic illustrating the effect of the magnetic field angle α on the motion of the μbot, fluorescent and brightfield microscopy images of μbot.

Our objective is to design a low-level controller that tracks a desired open loop trajectory $\boldsymbol{v}_d(t)$ over some time interval $t \in [0, T] \subset \mathbb{R}$. A control block diagram

outlining our nonlinear mismatch controller is presented in Fig. 2, and we present our working assumptions next.

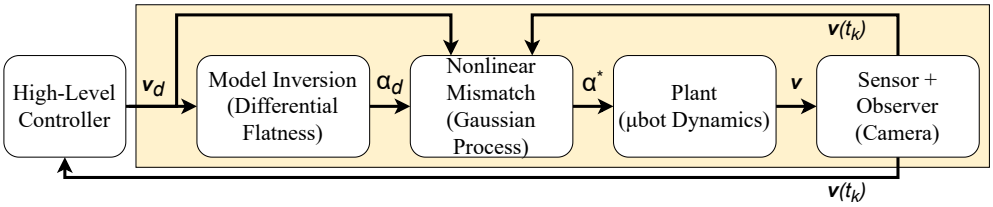

Fig. 2 A control block diagram showing how our proposed system (yellow box) transforms the desired velocity signal ($\boldsymbol{v}_d$) into a desired rolling speed (α_d) using the learned model; the Nonlinear Mismatch block yields the corrected output α^* that is sent to the system.

Assumption 1. *The environmental disturbances and model mismatch are isotropic and do not depend on the μbot's position* $\boldsymbol{p}$.

Assumption 2. *The error in aligning the μbot to the magnetic field is negligible.*

Assumptions 1 and 2 simplify the learning process and are reasonable for the laboratory environment. The disturbance acting on the μbots is primarily Brownian motion, which acts uniformly at random to disturb the velocity. Assumption 1 could be relaxed by having the μbots infer hydrodynamic disturbances caused by heat, density, and chemical concentration differences, e.g., using an approach similar to [24]. In previous work, we have also found that the alignment of μbots to the global magnetic field is nearly instantaneous (see [25]), which motivates Assumption 2.

Assumption 3. *The μbot is controlled to roll at a constant rate, i.e.,* $f(t)$ *is a constant.*

Assumption 3 does not affect the derivation of our controller, and it is not restrictive on our analysis; it can be relaxed by including $f(t)$ as an argument in the Gaussian Process (GP). This increases the amount of training data and learning time, but not prohibitively so. Furthermore, the μbots operate orders of magnitude faster than most medical applications require, and operating them with a constant rolling frequency is a common practice [17].

3 Learning-Based Controller

The μbot's dynamics (1) are differentially flat with the output variables $\boldsymbol{y} = [p^x, p^y]^T$ (see [26]). A system is differentially flat if and only if the state and control variables $\boldsymbol{x}(t) \in \mathcal{X} \subset \mathbb{R}^N, \boldsymbol{u}(t) \in \mathcal{U} \subset \mathbb{R}^M$ can be written as an explicit function of *output variables* $\boldsymbol{y} \subset \mathbb{R}^M$ and a finite number of their derivatives (see [27]). This implies the existence of a diffeomorphism (smooth bijective mapping) between the original space $\mathcal{X} \times \mathcal{U}$ and an *output space* $\mathcal{Y}$. Thus, we can analyze the simpler integrator system in $\mathcal{Y}$ to generate control actions that map back to the original nonlinear system in $\mathcal{X} \times \mathcal{U}$. The differential flatness property enables us to write the μbot's desired control

actions as an explicit function of a desired velocity,

$$\begin{aligned} \alpha_d &= \arctan\left(\frac{v_d^y - \hat{D}^y}{v_d^x - \hat{D}^x}\right), \\ f_d &= \frac{||\boldsymbol{v}_d - \hat{\boldsymbol{D}}||}{\hat{a}_0}, \end{aligned} \tag{2}$$

where $\hat{\cdot}$ denotes model parameter estimates, and the superscripts x, y denote the corresponding Cartesian vector components.

In general, the parameter a_0 cannot be measured directly (see [18]), and it is likely a function of the rolling frequency f, properties of the workspace fluid, local chemical concentrations, temperature, and asymmetry in the μbot. Similarly, the disturbance $\boldsymbol{D}$ includes model mismatch terms, stochastic Brownian motion, and other micron-scale disturbances. Substituting the model (2) directly into the dynamics (1) yields the realized velocity,

$$\boldsymbol{v} = \frac{a_0}{\hat{a}_0}\left(\boldsymbol{v}_d - \hat{\boldsymbol{D}}\right) + \boldsymbol{D}, \tag{3}$$

which is a function of the model parameter mismatch $\frac{a_0}{\hat{a}_0}$ and disturbance mismatch $\hat{\boldsymbol{D}} - \boldsymbol{D}$. Note that (3) is a nonlinear function of the model parameter $\hat{a}_0$ and the disturbance model $\hat{\boldsymbol{D}}$, which explains why a model parameter estimation alone is insufficient to achieve a desired trajectory.

To enhance our ability to track the desired trajectory beyond the capability of parameter estimation, we follow the approach outlined by [23] and explicitly define a velocity error $\boldsymbol{v}_e$,

$$\boldsymbol{v}_e \coloneqq \boldsymbol{v} - \boldsymbol{v}_d, \tag{4}$$

which has the explicit functional form,

$$\boldsymbol{v}_e = \boldsymbol{v}_d\left(\frac{a_0}{\hat{a}_0} - 1\right) + \boldsymbol{D} - \frac{a_o}{\hat{a}_o}\hat{\boldsymbol{D}}. \tag{5}$$

Thus, our objective is to learn the velocity error $\boldsymbol{v}_e$ as a function of α, f (both come from $\boldsymbol{v}_d$ through (2)) and the arguments of $\boldsymbol{D}$. Then, given an estimate of $\boldsymbol{v}_e$, we update the control inputs α and f to compensate for the model mismatch and disturbances. Note that if we replace the desired velocity $\boldsymbol{v}_d$ in (4) with a velocity command $\boldsymbol{v}_{cmd}$ meant to minimize our tracking error, the resulting velocity is,

$$\boldsymbol{v}_{cmd}(\alpha) + \boldsymbol{v}_e(\alpha) = \boldsymbol{v}, \tag{6}$$

where we seek to select α such that the difference between $\boldsymbol{v}$ and $\boldsymbol{v}_d$ is minimzied. We achieve this by first learning the velocity error $\boldsymbol{v}_e$ as a function of α under Assumptions 1–3 using GP regression. This determines, in a probabilistic manner, what underlying function fits the velocity error for a given μbot.. Then, we substitute our model into $\boldsymbol{v}_{cmd}$, and select the value of α that minimizes our μbot's deviation from the desired velocity.

3.1 Optimization-Based Controller

To learn the velocity error for our μbot, we employ a GP. A GP is completely defined by its mean $\mu(\alpha)$ and kernel (or covariance) $K(\alpha, \alpha')$ functions. 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 $\boldsymbol{v}_e(\alpha)$. In this work, we learn the velocity error $\boldsymbol{v}_e$ from experimental data, where the two Cartesian axes of $\boldsymbol{v}_e$ are each captured by a GP. This yield a Gaussian distribution at each point α^*, with a mean that predicts the expected value of $\boldsymbol{v}_e(\alpha)$ and a standard deviation that predicts the uncertainty in that estimate. We exploit this property to predict model mismatch and correct our control input before acting on the system, and the resulting error bounds could also be exploited by a high-level planner as a measure of the trajectory's robustness.

Substituting our learned model into the velocity command $\boldsymbol{v}_{cmd}$ of (6), and assuming the GP has sufficiently learned the velocity error function $\boldsymbol{v}_e$, implies

$$\hat{a}_0 \, f \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix} + \hat{\boldsymbol{D}} + \boldsymbol{\mu}(\alpha) = \boldsymbol{v}(\alpha), \tag{7}$$

where $\boldsymbol{\mu}$ is the GPs' estimate for each component of the velocity error. To minimize the velocity error of our μbot, we perform a least squares minimization of (7) from the desired velocity, i.e.,

$$\min_{\alpha} ||\boldsymbol{v}(\alpha) - \boldsymbol{v}_d||^2. \tag{8}$$

Applying Assumptions 1–3 and expanding (8) yields a one-dimensional least-squares cost function,

$$J(\alpha) = \left(\hat{a}_0 f \cos(\alpha) + \mu^x(\alpha) + \hat{D}^x - v_d^x\right)^2 + \left(\hat{a}_0 f \sin(\alpha) + \mu^y(\alpha) + \hat{D}^y - v_d^y\right)^2. \tag{9}$$

Expanding the cost function and applying the Pythagorean identity yields,

$$J(\alpha) = (a_0 f)^2 + ||\boldsymbol{\mu}(\alpha) + \hat{\boldsymbol{D}} - \boldsymbol{v}_d||^2 + 2a_0 f \left(\boldsymbol{\mu}(\alpha) + \hat{\boldsymbol{D}} - \boldsymbol{v}^d\right) \cdot \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix}. \tag{10}$$

Thus, at each time instant, we seek the value of α that minimizes the least squares error, i.e.,

$$J^*(\alpha) = \min_{\alpha \in [-\pi, \pi]} (a_0 f)^2 + ||\boldsymbol{\mu}(\alpha) + \hat{\boldsymbol{D}} - \boldsymbol{v}_d||^2 + 2a_0 f \left(\boldsymbol{\mu}(\alpha) + \hat{\boldsymbol{D}} - \boldsymbol{v}^d\right) \cdot \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix}, \tag{11}$$

which is a 1D, bounded, continuous, and differentiable optimization problem.

3.2 Online Learning

We train the GP during an initial *learning phase*, where the μbot is given a sequence of control inputs, either from a human operator or open-loop control sequence. We

collect position and control action data for the μbot 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 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 achieved velocity of the μbot 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,

$$\dot{\boldsymbol{p}} = \boldsymbol{D}. \tag{12}$$

Taking the expectation of both sides yields the mean disturbance,

$$\frac{1}{|\mathcal{V}|} \sum_{\boldsymbol{v}(t_k) \in \mathcal{V}} ||\boldsymbol{v}(t_k)|| = \mathbb{E}\big[\boldsymbol{D}\big] = \hat{\boldsymbol{D}}, \tag{13}$$

where $|\cdot|$ is set cardinality. Note that for systems subject to purely Brownian motion, $\hat{\boldsymbol{D}} = \boldsymbol{0}$.

Next, we determine $\hat{a}_0$ using data from an open loop control sequence; taking the expectation of (1) and squaring both sides yields,

$$\mathbb{E}\big[||\boldsymbol{v} - \hat{\boldsymbol{D}}||\big]^2 = a_0^2 f^2. \tag{14}$$

We estimate the expectation of $\boldsymbol{v}$ using the data set $\mathcal{V}$; this yields the best statistical estimate for $\hat{a}_0$,

$$\hat{a}_0 = \frac{1}{|\mathcal{V}|} \sum_{\boldsymbol{v}(t_k) \in \mathcal{V}} \frac{||\boldsymbol{v}(t_k) - \hat{\boldsymbol{D}}||}{f}. \tag{15}$$

Finally, the resulting set of velocity errors is

$$\mathcal{Y} = \Big\{ \boldsymbol{v}_e(t_k) \ : \ \boldsymbol{v}_e = \boldsymbol{v}(t_k) - \boldsymbol{v}_d(t_k) \Big\}, \tag{16}$$

where $\boldsymbol{v}_d(t_k)$ is the desired velocity of the μbot at each time t_k. We use (16), in conjunction with our empirical model (2), 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.

4 Experimental Results

We validated our learning approach in silico and in situ[1]; we present our simulation results in Subsection 4.1 and experimental findings in Subsection 4.2. 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

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

sequence of control actions in open-loop with and without the Nonlinear Mismatch module (Fig. 2). This 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 [28]) 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, and this provided powerful 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 don't describe the true behavior of the rolling μbot's velocity error. We found that a linear combination of a radial basis function and white noise yielded a kernel that adequately captures the velocity error of the rolling μbot,

$$K(\alpha, \alpha') = \exp \frac{||\alpha - \alpha'||^2}{2\sigma} + \eta, \tag{17}$$

where σ is a length hyperparameter and η is drawn from a normal distribution where the mean is zero and the variance is another hyperparameter.

4.1 In Silico Experiment

We developed a μbot 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 3 as follows. First, we applied zero input over 100 time steps (3 seconds) with the 'model-mismatch' flag to estimate the mean disturbance using (13). 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 (15) and (16), 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 (11) to generate the *corrected* trajectory with the 'model mismatch' flag.

Fig. 3 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 model mismatch–even in an environment with no noise. The velocity error estimate for one trial is presented in Fig. 4, 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 (7). This stems from correcting the x and y components of the velocity error while only controlling α.

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

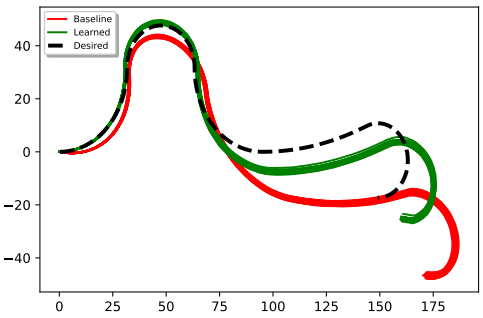

Fig. 3 The desired, (blue, dashed), corrected (green), and baseline (orange) trajectories from 100 different trials of the in silico experiment.

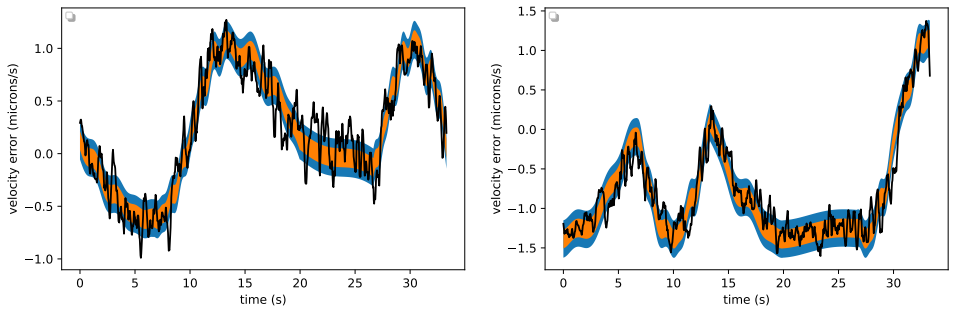

Fig. 4 The GP's prediction of the x (left) and y (right) axis velocity errors at each time step; orange bands correspond to one standard deviation (65%), and the blue band corresponds to two standard deviations (95%) of uncertainty.

4.2 In Situ Experiment

We repeated the same procedure from Section 4.1 using 24um μbots over 300 time steps (10 seconds) at the experimental facility at the University of Delaware (the setup is described in the Appendix). The resulting desired, baseline, and corrected trajectories are presented in Fig. 5, along with the drift error for the baseline and corrected cases; photos of an experiment with the trajectories overlaid are presented in Fig. 6.

The left image in Fig. 5 shows significant improvement in the μbot's ability to track the open-loop trajectory, and the right image shows the that the μbot's drift was reduced by at least 6 pixels (3.7 microns) along each axis for the majority of the experiment. To calculate the μbot's drift, we subtracted the desired and achieved velocity of the particle along each axis to calculate the velocity error. Next, we performed a cumulative trapezoidal integration on the absolute value of the velocity error, which quantified the worst-case scenario for how far the μbot could drift.

The median velocity error along each axis is presented in Table 1, along with the error in the μbot'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 μbot by matching the desired open-loop control policy and brings the μbot closer to

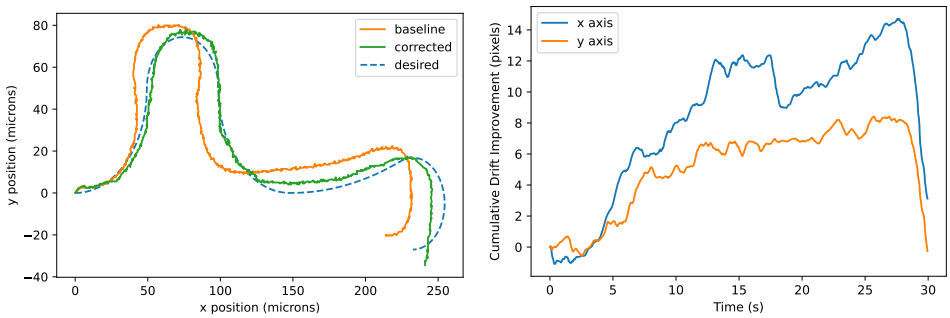

Fig. 5 Left: a comparison of the baseline (orange), corrected (green), and desired (blue, dashed) trajectories from the in situ experiment. Right: improvement in the cumulative drift of the μbot between the baseline and corrected caeses.

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 μbot's trajectory in a way that our tracking controller cannot compensate for (see the Appendix). These exogenous factors are the source of error in the last 3 seconds of the *corrected* experiment.

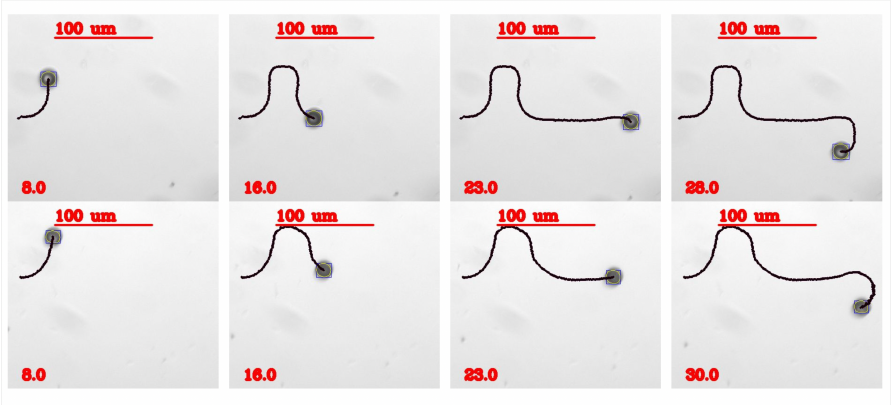

Fig. 6 Snapshots of another *baseline* (top) and *corrected* (bottom) experiment, taken approximately every 8 seconds apart and showing the μbot with the position history overlaid.

	Baseline		Corrected		Improvement
Final Position Error	19.85 px	(12.56 microns)	11.28 px	(7.14 microns)	43 %
Median v_x Error	1.88 px/s	(1.19 microns/s)	1.13 px/s	(0.72 microns/s)	40 %
Median v_y Error	1.26 px/s	(0.80 microns/s)	0.97 px/s	(0.61 microns/s)	23 %

Table 1 Errors for the physical μbot experiment; the median uses the absolute value of the error.

5 Conclusion

In this article, 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 μbots.

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 μbots simultaneously would advance the state of the art, and bring us one step closer to solving fundamental challenges in emerging medical applications.

Appendix: Experimental Setup

The experimental setup is a 3D Helmholtz system based on the Jetson Xavier NX as described in [29]. We read frames from the FLIR BFS-U3-28S5M-C USB 3.1 Blackfly® S Monochrome Camera, which extracts the μbot's positions from a Zeiss Axiovert 100 inverted microscope. Similar to [18], we convert our control commands into a rotating 3D magnetic field via,

$$\boldsymbol{B} = \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = ||\boldsymbol{B}|| \begin{bmatrix} \cos(\gamma)\cos(\alpha)\cos(2\pi ft) + \sin(\alpha)\sin(2\pi ft) \\ -\cos(\gamma)\sin(\alpha)\cos(2\pi ft) + \cos(\alpha)\sin(2\pi ft) \\ \sin(\gamma)\cos(2\pi ft) \end{bmatrix}, \tag{18}$$

where $\boldsymbol{B} \in \mathbb{R}^3$ is the magnetic field strength, $\gamma \in \mathbb{R}$ is the constant inclination (azimuthal angle) of the magnetic field, and $\alpha, f \in \mathbb{R}$ are the control actions, which control the heading angle and rolling frequency of the μbot. These action commands are then sent from the Jetson Xaiver to an Arduino Mega via UART to be executed, which allows the 3 phase sine waves to be generated independent of the computer vision software. This allows for higher frequencies, and thus, faster rolling speeds.

The μbots 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 μbot's magnetic moment. Due to the inherent surface properties of the μbot and the substrate surface, there are often very large attractive forces that result in the μbot 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 μbot to more easily roll across the surface. Secondly, instead of suspending the μbot'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.

References

[1] Honda, T., Arai, K., Ishiyama, K.: Micro swimming mechanisms propelled by external magnetic fields. IEEE Transactions on Magnetics **32**(5), 5085–5087 (1996)

[2] Sitti, M., Ceylan, H., Hu, W., Giltinan, J., Turan, M., Yim, S., Diller, E.: Biomedical applications of untethered mobile milli/microrobots. Proceedings of the IEEE **103**(2), 205–224 (2015)

[3] Troccaz, J., Bogue, R.: The development of medical microrobots: a review of progress. Industrial Robot: An International Journal (2008)

[4] Bárcena, C., Sra, A.K., Gao, J.: Applications of magnetic nanoparticles in biomedicine. In: Nanoscale Magnetic Materials and Applications, pp. 591–626. Springer, ??? (2009)

[5] Guo, S., Pan, Q.: Mechanism and control of a novel type microrobot for biomedical application. In: Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 187–192. IEEE, ??? (2007)

[6] Sakar, M.S., Steager, E.B., Cowley, A., Kumar, V., Pappas, G.J.: Wireless manipulation of single cells using magnetic microtransporters. In: 2011 IEEE International Conference on Robotics and Automation, pp. 2668–2673. IEEE, ??? (2011)

[7] Jager, E.W.H., Inganäs, O., Lundström, I.S.: Microrobots for micrometer-size objects in aqueous media: potential tools for single-cell manipulation **288**(5475), 2335–2338 (2000)

[8] Kim, S., Qiu, F., Kim, S., Ghanbari, A., Moon, C., Zhang, L., Nelson, B.J., Choi, H.: Fabrication and characterization of magnetic microrobots for three-dimensional cell culture and targeted transportation. Advanced Materials **25**(41), 5863–5868 (2013)

[9] Steager, E.B., Selman Sakar, M., Magee, C., Kennedy, M., Cowley, A., Kumar, V.T.I.J.o.R.R.: Automated biomanipulation of single cells using magnetic microrobots **32**(3), 346–359 (2013)

[10] Kim, H., Kim, M.J.: Electric field control of bacteria-powered microrobots using

a static obstacle avoidance algorithm. IEEE Transactions on Robotics **32**(1), 125–137 (2015)

[11] Palima, D., Glückstad, J.: Gearing up for optical microrobotics: micromanipulation and actuation of synthetic microstructures by optical forces. Laser & Photonics Reviews **7**(4), 478–494 (2013)

[12] Chowdhury, S., Jing, W., Cappelleri, D.J.: Towards independent control of multiple magnetic mobile microrobots. Micromachines **7**(1), 3 (2016)

[13] Erdem, E.Y., Chen, Y.-M., Mohebbi, M., Suh, J.W., Kovacs, G.T., Darling, R.B., Böhringer, K.F.: Thermally actuated omnidirectional walking microrobot. Journal of Microelectromechanical Systems **19**(3), 433–442 (2010)

[14] Behkam, B., Sitti, M.: Bacteria integrated swimming microrobots. In: Lecture Notes in Computer Science, vol. 4850 LNAI, pp. 154–163 (2007)

[15] Das, S., Steager, E.B., Hsieh, M.A., Stebe, K.J., Kumar, V.: Experiments and open-loop control of multiple catalytic microrobots. Journal of Micro-Bio Robotics **14**(1-2), 25–34 (2018)

[16] Vinagre, B.M., Tejado, I., Traver, J.E.: There's plenty of fractional at the bottom, i: Brownian motors and swimming microrobots. Fractional Calculus and Applied Analysis **19**(5), 1282 (2016)

[17] Rivas, D., Mallick, S., Sokolich, M., Das, S.: Cellular manipulation using rolling microrobots. In: 2022 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS), pp. 1–6 (2022). https://doi.org/10.1109/MARSS55884.2022.9870486

[18] Yang, L., Zhang, Y., Wang, Q., Chan, K.-F., Zhang, L.: Automated control of magnetic spore-based microrobot using fluorescence imaging for targeted delivery with cellular resolution. IEEE Transactions on Automation Science and Engineering **17**(1), 490–501 (2020) https://doi.org/10.1109/TASE.2019.2937232

[19] Tang, X., Li, Y., Liu, X., Liu, D., Chen, Z., Arai, T.: Vision-based automated control of magnetic microrobots. Micromachines **13**(2), 337 (2022)

[20] Pieters, R., Tung, H.-W., Charreyron, S., Sargent, D.F., Nelson, B.J.: Rodbot: A rolling microrobot for micromanipulation. In: 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 4042–4047 (2015). IEEE

[21] Behrens, M.R., Ruder, W.C.: Smart magnetic microrobots learn to swim with deep reinforcement learning. Advanced Intelligent Systems **4**(10), 2200023 (2022)

[22] Xu, T., Yu, J., Yan, X., Choi, H., Zhang, L.: Magnetic actuation based motion control for microrobots: An overview. Micromachines **6**(9), 1346–1364 (2015)

[23] Greeff, M., Schoellig, A.P.: Exploiting Differential Flatness for Robust Learning-Based Tracking Control using Gaussian Processes. IEEE Control System Letters **5**(4), 1121–1126 (2021)

[24] Chang, D., Wu, W., Edwards, C.R., Zhang, F.: Motion tomography: Mapping flow fields using autonomous underwater vehicles. The International Journal of Robotics Research **36**(3), 320–336 (2017) https://doi.org/10.1177/0278364917698747

[25] Beaver, L.E., Wu, B., Das, S., Malikopoulos, A.A.: A first-order approach to model simultaneous control of multiple microrobots. In: 2022 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS), pp. 1–7 (2022)

[26] Sira-Ramirez, H., Agrawal, S.K.: Differentially Flat Systems, 1st edn. CRC Press, ??? (2018)

[27] Fliess, M., Levine, J., Martin, P., Rouchon, P.: Flatness and defect of non-linear systems: Introductory theory and examples. International Journal of Control **61**(6), 1327–1361 (1995)

[28] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[29] Sokolich, M., Rivas, D., Duey, M., Borsykowsky, D., Das, S.: Modmag: A modular magnetic micro-robotic manipulation device. arXiv preprint arXiv:2211.01173 (2022)