IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2023, London, UK. Cite as RA-L paper.

Learning Deep Neural Network Controller for Path Following of Unicycle Robots

Priyabrata Saha[1], Luis Guerrero-Bonilla[2], Magnus Egerstedt[3], and Saibal Mukhopadhyay[1]

***Abstract*—This paper investigates the scope of deep neural network (DNN) based controller in the path following task for unicycle mobile robots. A DNN-based controller is trained to follow paths with arbitrary curvature in two-dimensional space. The training process does not require initialization or supervision from any other known expert controller. Rather, the training of the DNN controller is guided by another predictive neural network that represents a path following error dynamics which is exponentially stable at the origin. The two DNNs are trained jointly in a simulated environment. The learned DNN controller is then employed as a standalone controller in a real unicycle robot for the tasks of following various linear and curved paths.**

***Index Terms*—Mobile robots, path following, deep learning methods, machine learning for robot control**

I. Introduction

MOBILE robots are being widely used in different areas like manufacturing industries, transportation, surveillance, and domestic applications, just to name a few. Planning a path and then following the prescribed path are two crucial functionalities, among others, of autonomous mobile robots. This paper focuses on developing a path-following controller for unicycle mobile robots. The primary goal of path-following control is to design a control law that drives a mobile robot to follow a planned geometric path [1]. Unlike trajectory tracking, the planned path in path-following task does not prescribe any temporal specification [2]. The path following task allows the robot to track the geometric curve with any feasible speed profile [3]. Usually, path following achieves a smoother convergence to the desired path, compared to trajectory-tracking control [2].

Path following control has been heavily researched. The primary approaches for path following [3]–[5] are based on the classical concept [4] that executes an orthogonal or non-orthogonal projection of the current robot position on the desired path to compute a distance (or crosstrack) and a heading error. A controller takes these errors as input and actuates on robot's orientation to steer it to the path, while the linear velocity of the robot conforms to a predefined profile. The second types of method define the error state using polar coordinates with respect to a goal frame [6] or virtual vehicle [7] that has its own dynamics for describing the motion. Recently, vector-field guided path following algorithms have gained popularity among researchers [8], [9]. In these algorithms, a vector field is carefully designed, such that its integral curves approach the path asymptotically [9]. Morro et al. [10] introduced another approach, the level curve (LC) method, that computes the path following error using the implicit equation of the desired path.

This paper studies the purview of deep neural network (DNN) based controllers in the path following task for unicycle robots, utilizing the level curve approach. Single hidden-layer neural networks have been used for a long time in path following control of mobile robots to approximate the dynamics [11], [12] and/or to assist classical backstepping [11] or PID [13], [14] controller. Recently, the success of reinforcement learning (RL), aided by DNNs, in various difficult robotic tasks in virtual environments has motivated many researchers to apply RL in path-following tasks as well. Consequently, RL is being utilized, with [15] and without [16] DNNs, to assist classical controllers for better performance in path following of practical robots. Although the use of a standalone neural network has shown promising results in simulation [17], [18], its application in path following control of practical robots is rare. Learning DNN-based controller either requires supervision from other expert controllers or a large amount of data that may not be feasible to collect in practical scenarios.

In this paper, we investigate the scope of learning a DNN-based controller in simulation without any supervision from any expert classical controller and employing it as a standalone controller to drive a practical unicycle robot on paths with arbitrary curvature. Instead of using supervision from system behavior under a known expert controller, we use another DNN that hypothesizes a target dynamics, which is exponentially stable at the origin, for the path following error. The controller DNN and the DNN that represents a dynamical system, which is stable at the origin, are trained jointly, guiding each other in the learning process. Path following error dynamics generally depends on the curvature of the path in consideration or its implicit equation. As a consequence, a controller trained using the error dynamics of a specific path will perform well for that specific path only and will likely fail in following other paths. To avoid retraining the

Manuscript received: June, 09, 2022; Revised August, 29, 2022; Accepted October, 27, 2022.

This paper was recommended for publication by Editor Dana Kulic upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by the Army Research Office under Grant W911NF-19-1-0447.

[1]Priyabrata Saha and Saibal Mukhopadhyay are with School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. priyabratasaha@gatech.edu, saibal.mukhopadhyay@ece.gatech.edu

[2]Luis Guerrero-Bonilla is with School of Engineering and Sciences, Tecnológico de Monterrey, Monterrey 64700, NL, Mexico. lguerrero@tec.mx

[3]Magnus Egerstedt is with Samueli School of Engineering, University of California, Irvine, CA 92697, USA. magnus@uci.edu

Digital Object Identifier (DOI): see top of this page.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2023, London, UK. Cite as RA-L paper.

controller every time the path changes, which is inefficient if not infeasible in practical scenarios, we propose to train the controller using multiple randomly generated paths. It is important to note that the training of the DNNs happens offline using simulated data from the nominal kinematic model of the robot and therefore, does not involve data collection using the actual robot in different paths. Control inputs of real robots are generally bounded. A DNN controller trained without any regulations in simulation is likely to violate those bounds and cause unexpected behavior. On the other hand, directly applying the hard bounds on the controller during training reduces its flexibility to learn arbitrary functions. Therefore, we propose to apply soft constraints on the control inputs, linear and rotational velocity, by adding regulatory terms to the training objective.

We use the robotarium platform [19] to demonstrate the performance of the learned controller in the tasks of following various linear and curved paths. We show that the performance of the controller, learned in simulation, can be improved by re-tuning some design parameters based on its behavior in experiment. Furthermore, we observe that controlling linear speed along with rotational velocity reduces the deviation from the desired path at points with sharp orientation changes. We also compare the performance of the proposed DNN controller with a PD controller. PD controller tracks better when the path does not have a sharp turn, but the opposite is true when the path involves a sharp orientation change.

The rest of this paper is organized as follows. Section II presents the problem and the approach we used to address it. Section III describes our experimental setup. Section IV presents the experiments we used to evaluate the proposed method and analyzes the results. Finally, Section V concludes the paper.

II. Problem Statement and Methodology

A. Problem Statement

Consider a unicycle robot, as shown in Fig. 1a. The state of the robot is described by (x, y, θ), where $(x, y) \in \mathbb{R}^2$ corresponds to its position in the two-dimensional plane and $\theta \in [-\pi, \pi]$ denotes its orientation. The kinematics of the system is given by

$$\begin{aligned} \dot{x} &= v\cos\theta, \\ \dot{y} &= v\sin\theta, \\ \dot{\theta} &= \omega\ . \end{aligned} \tag{1}$$

Here, v and ω stand for the linear and rotational velocity, respectively, which are the control inputs of the mobile robot.

For such a unicycle robot, we consider the problem of learning a neural network-based controller which drives the robot to follow a generic 2-D curve path that can be represented in the implicit form $f(x, y) = 0$.

The path following problem requires a framework to compute the error state and dynamics. We follow the level curve method proposed by Morro et al. [10] to compute a path-dependent error dynamics for its simplicity of implementation. Using the level curve framework to compute the error state and dynamics, we formulate a learning task to train a DNN that can drive the robot to follow a generic 2D-path. The following two subsections describe the process of computing the error dynamics and learning a neural network controller, respectively.

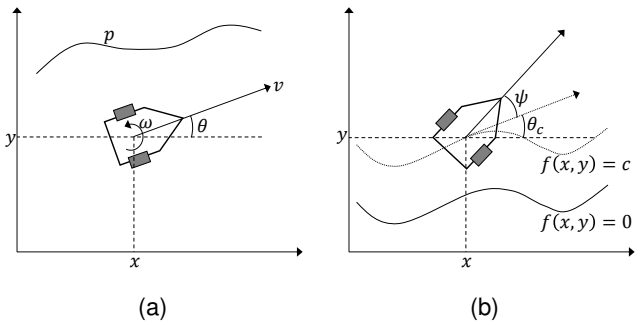

Fig. 1. (a): Schematic of a unicycle mobile robot and its state variables. (b): Path following errors for following a generic curve $f(x, y) = 0$.

B. Computing the Error Dynamics: Level Curve Approach

The level curve (LC) method uses the implicit equation of the desired path to compute the path following error. When the robot is on the path, the equation is satisfied and the robot's position belongs to the zero-level curve. On the other hand, points outside of the path do not satisfy the equation and belong to level curves with nonzero heights. This nonzero value is considered as a signed measure of a robot distance from the desired path. This method neither requires projection to the path nor virtual moving robot for error computation. The error state for a generic curve $f(x, y) = 0$ at state (x, y, θ) is shown in Fig. 1b. Instead of the Euclidean distance to the nearest path point, the value of function f at the robot position (x, y) is considered as the distance from the desired path. The heading error is computed as the angle between robot's orientation and the tangent to the level curve at (x, y), i.e.,

$$\psi = \theta - \theta_c\ . \tag{2}$$

Assuming f is differentiable and $\|\nabla f\|^2 = f_x^2 + f_y^2 > 0$, the angle θ_c can be computed as

$$\theta_c = \arg(f_y - if_x), \tag{3}$$

where $\arg$ denotes the complex argument, and f_x and f_y are the first order partial derivatives of f. Using (1) the time derivative of the distance error f can be written as

$$\dot{f} = f_x v\cos\theta + f_y v sin\theta. \tag{4}$$

Assuming f is twice differentiable and its first and second order partial derivatives are bounded in any bounded domain $D \subset \mathbb{R}^2$, the time derivative of the heading error ψ can be computed as

$$\dot{\psi} = \omega - \frac{(f_x f_{xy} - f_y f_{xx})v\cos\theta + (f_x f_{yy} - f_y f_{xy})v\sin\theta}{\|\nabla f\|^2}. \tag{5}$$

Here, f_{xx}, f_{yy}, and f_{xy} represent the second order partial derivatives of f.

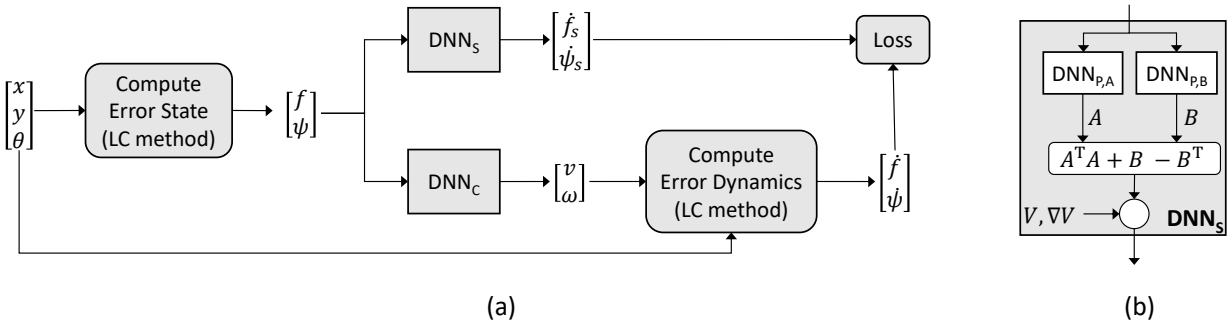

Fig. 2. (a): Proposed method to jointly train DNN_C and the DNN_S. DNN_S is a predictive neural network that predicts a dynamics, which is exponentially stable at the origin, for the path following error $[f, \psi]^\top$. DNN_C is responsible for learning a policy to control the linear speed v and the rotational velocity ω. (b): Block-level structure of DNN_S, guided by Lyapunov energy function V.

C. Learning a Neural Network Controller

To learn a neural network controller, we adapt our previous work [20] to the arbitrary path following problem. In [20], we showed that a control law that stabilizes a system at the origin and the corresponding closed-loop dynamics can be learned jointly by training two neural networks. One neural network, DNN_S, learns to generate a hypothesis for a dynamical system that is exponentially stable at the origin and the other, DNN_C, learns to produce a control input such that the closed-loop dynamics will match the behavior of the dynamical system hypothesized by DNN_S. Control input obtained from DNN_C is applied to a known or learned model of the system to estimate the behavior of the system under that control input. Both networks are trained jointly by minimizing the difference between the time-derivative of the system state subjected to the control input and the time-derivative of the state of the hypothesized dynamical system, which is exponentially stable at the origin.

In the path following setup, we want to stabilize the error dynamics at the origin. The neural network DNN_S takes the error state $[f, \psi]^\top$ as input and predicts a vector field $[\dot{f}_s, \dot{\psi}_s]^\top$ that represents a dynamical system exponentially stable at the origin. On the other hand, DNN_C takes the error state $[f, \psi]^\top$ as input and predicts a control vector of linear and rotational velocity $[v, \omega]^\top$ that should drive the system in such a way that the closed-loop dynamics match the behavior of the dynamical system hypothesized by DNN_S. The time-derivative $[\dot{f}, \dot{\psi}]^\top$ of the system state, subjected to the control vector $[v, \omega]^\top$ is computed using (4) and (5). Note, it is possible to keep the linear speed v constant and learn only the rotational velocity ω. However, while driving through a path that requires sharp change in orientation, a high linear speed can cause large deviation from the path leading to potential instability. To avoid such cases, we constrain our DNN_C to learn to reduce linear speed when the heading error ψ is large.

The neural networks DNN_S and DNN_C are trained by minimizing the following objective function:

$$\mathcal{L} = \|[\dot{f}_s, \dot{\psi}_s]^\top - [\dot{f}, \dot{\psi}]^\top\|^2 + \lambda_1 \frac{v|\psi|}{v_{\max}} - \lambda_2 \log(v/v_{\max}). \quad (6)$$

Here, λ_1 and λ_2 are regularization hyperparameters and $v_{\max}$ denotes the maximum allowed linear speed. The first term of (6) is responsible for minimizing the difference between the dynamics hypothesis from DNN_S and the system dynamics subjected to the control law from DNN_C, whereas the second term penalizes high linear speed when heading error is large. A trivial solution of minimizing the first two terms of (6) is $v = 0$, $\dot{f}_s = 0$, and $\dot{\psi}_s = \omega$. To avoid this trivial solution and push the robot to move at maximum allowed linear velocity whenever feasible, we add a third term that discourages $v < v_{\max}$. The rotational velocity ω is also thresholded by a maximum allowed rotational velocity, i.e., $|\omega| \leq \omega_{\max}$. Block diagram of training forward path is shown in Fig. 2(a).

Dynamics generated by a standard neural network are not stable in general. We follow the method proposed in [20] and [21] to design a inherently stable neural network for DNN_S. Specifically, we use a neural network whose output is proportional to the negative gradient of a given Lyapunov energy function, i.e.,

$$\begin{bmatrix}\dot{f}_s \\ \dot{\psi}_s\end{bmatrix} = -\mathbf{P}(f, \psi)\nabla V - \frac{\text{ReLU}(-\nabla V^\top \mathbf{P}(f, \psi)\nabla V + \alpha V(f, \psi))}{\|\nabla V\|^2}\nabla V. \quad (7)$$

Here, V is a Lyapunov function, and $\nabla V = [V_f, V_\psi]^\top$ represents the gradient of V with respect to the error state vector. The proportionality matrix $\mathbf{P}(f, \psi) \in \mathbb{R}^{2\times 2}$ is defined as

$$\mathbf{P}(f, \psi) = \mathbf{A}(f, \psi)^\top \mathbf{A}(f, \psi) + \mathbf{B}(f, \psi) - \mathbf{B}(f, \psi)^\top, \quad (8)$$

so that $\mathbf{x}^\top \mathbf{P}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^2$, given any arbitrary matrices $\mathbf{A}(f, \psi), \mathbf{B}(f, \psi) \in \mathbb{R}^{2\times 2}$. α is a positive constant, and $\text{ReLU}(z) = \max(0, z), z \in \mathbb{R}$. Stability analysis of the dynamics defined by (7) is shown in [20]. The first term of the right hand side of (7) only ensures stability [20] but the addition of the second term guarantees exponential stability [21].

The neural network DNN_S is designed based on (7). The output neurons of a neural network constitute the entries of matrices $\mathbf{A}$ and $\mathbf{B}$, which are arranged according to (8) to provide the matrix $\mathbf{P}$, which is connected with V and ∇V to provide the final output of DNN_S (Fig. 2(b)).

Note that, DNN_S is only required during training. The goal of the training is to minimize the difference between dynamics generated by DNN_S and the closed-loop dynamics subjected to the controller DNN_C. During evaluation, DNN_C works independently to provide control inputs given a system state.

III. Experimental Setup

A. Experiment platform

We use the robotarium platform [19] for experiments. Constrained by the maximum rotational speed of the motors, the linear speed v and rotational velocity ω of the robot are clipped by $v_{\max} = 8$ cm/s and $\omega_{\max} = 2$ rad/s, respectively.

B. DNN architecture and other hyperparameters

The controller DNN_C is implemented as a multilayer perceptron (MLP) comprising three hidden layers each having 60 neurons with ReLU activation. The number of input and output neurons of DNN_C are same as the dimension of the error state and the control input, respectively. DNN_S is implemented using an MLP of three hidden layers each having 100 neurons with ReLU activation. The number of input neurons of this MLP is same as the dimension of the error state, i.e, 2, whereas the number of output neurons equals the total number of elements of matrices $\mathbf{A}$ and $\mathbf{B}$, i.e., 8. The outputs of the MLP are arranged to form matrices $\mathbf{A}$ and $\mathbf{B}$ which are combined according to (8) to provide the matrix $\mathbf{P}$. Finally, matrix $\mathbf{P}$ is connected with V and ∇V according to (7) to generate $[\dot{f}_s, \dot{\psi}_s]^\top$. We use quadratic Lyapunov function $V(f, \psi) = f^2 + q\psi^2$, where $q > 0$. For a given α (in eqn. (7)), varying q provides a trade-off between the tracking error and the oscillation of the control signal. $\alpha = 0.03$ and $q = 0.06$ are used for all experiments, unless otherwise mentioned. We use $\lambda_1 = 0.7$ and $\lambda_2 = 0.5$ for the regularization hyperparameters in (6).

C. Dataset and DNN training

To train the DNNs, we need data of the error dynamics as input-output pairs $\{[f, \psi]^\top, [\dot{f}, \dot{\psi}]^\top\}$. The error dynamics is path dependent. Therefore, a model trained with data from a specific path, is unlikely to perform well in task of following a different path. On the other hand, training a model each time the desired path changes is inefficient and may not be feasible in many scenarios. To generalize the learned model, we train it with data from multiple randomly generated paths. Specifically, we consider sinusoidal paths in the form of truncated Fourier series, i.e.,

$$f(x, y) = y + c + \sum_{n=1}^{N} a_n \cos(nx) + b_n \sin(nx) = 0, \quad \text{(9a)}$$

and,

$$f(x, y) = x + c + \sum_{n=1}^{N} a_n \cos(nx) + b_n \sin(ny) = 0. \quad \text{(9b)}$$

The coefficients $a_n, b_n \sim U(-0.5, 0.5)$ and $c \sim U(-1, 1)$, where U stands for the uniform distribution. We used $N = 7$. For each path, we randomly sample some points from $U([-2, 2] \times [-2, 2] \times [-\pi, \pi])$ as robot's position and orientation, and obtain the corresponding error state $[f, \psi]^\top$. The time derivative of the error state is computed using (4) and (5) for maximum linear speed $v_{\max}$ and zero rotational velocity ($\omega = 0$). The pairs $\{[f, \psi]^\top, [\dot{f}|_{v=v_{\max}}, \dot{\psi}|_{v=v_{\max}, \omega=0}]^\top\}$ for 400 different paths and 20000 points form the training dataset. Note, the training of the DNNs is performed using simulated data from the nominal kinematic model (i.e. eqn. (1)) of the robot and therefore does not involve data collection using the actual robot in different paths.

During training, the time derivative of the error state for a control input $[v, \omega]^\top$, generated by DNN_C, is computed by

$$\begin{aligned} \dot{f} &= \frac{v}{v_{\max}} \dot{f}|_{v=v_{\max}}, \\ \dot{\psi} &= \omega + \frac{v}{v_{\max}} \dot{\psi}|_{v=v_{\max}, \omega=0}. \end{aligned} \quad \text{(10)}$$

The DNNs are trained using Adam optimizer in mini-batches of 400 samples for 20000 steps with a learning rate of 10^{-4} and weight decay of 5×10^{-6}.

D. Evaluation metrics

To measure the performance of a learned control law in the path following task, we define the following metrics.

Mean Absolute Tracking Error (f_{mean}). Mean absolute tracking error measures the average deviation of the robot from the desired path and is defined as

$$f_{\text{mean}} = \frac{1}{T} \sum_{t=1}^{T} |f(x_t, y_t)|, \quad \text{(11)}$$

where (x_t, y_t) denotes the robot's position at timestep t, and T is the number of timesteps observed.

Maximum Absolute Tracking Error ($f_{\max}$). Maximum absolute tracking error measures the maximum deviation of the robot from the desired path and is defined as

$$f_{\max} = \max \left(f_{\text{mean}}, \max_{t \in [t_0, T]} |f(x_t, y_t)| \right). \quad \text{(12)}$$

If the initial position of the robot is far away from the path, then the value of $f_{\max}$ would be the value of f at the initial position. To avoid this case, we take the maximum after the robot crosses the f_{mean} for the first time, which we denote as t_0.

Mean Rotational Speed(ω_{mean}). Mean rotational speed measures the average control effort and is defined as

$$\omega_{\text{mean}} = \frac{1}{T} \sum_{t=1}^{T} |\omega_t|, \quad \text{(13)}$$

where ω_t denotes the rotational velocity at timestep t. A lower value is better for all the metrics.

IV. Experimental Results

We evaluate the learned controller in the path following task using different paths including sinusoidal, elliptical, linear spline, and cubic spline paths.

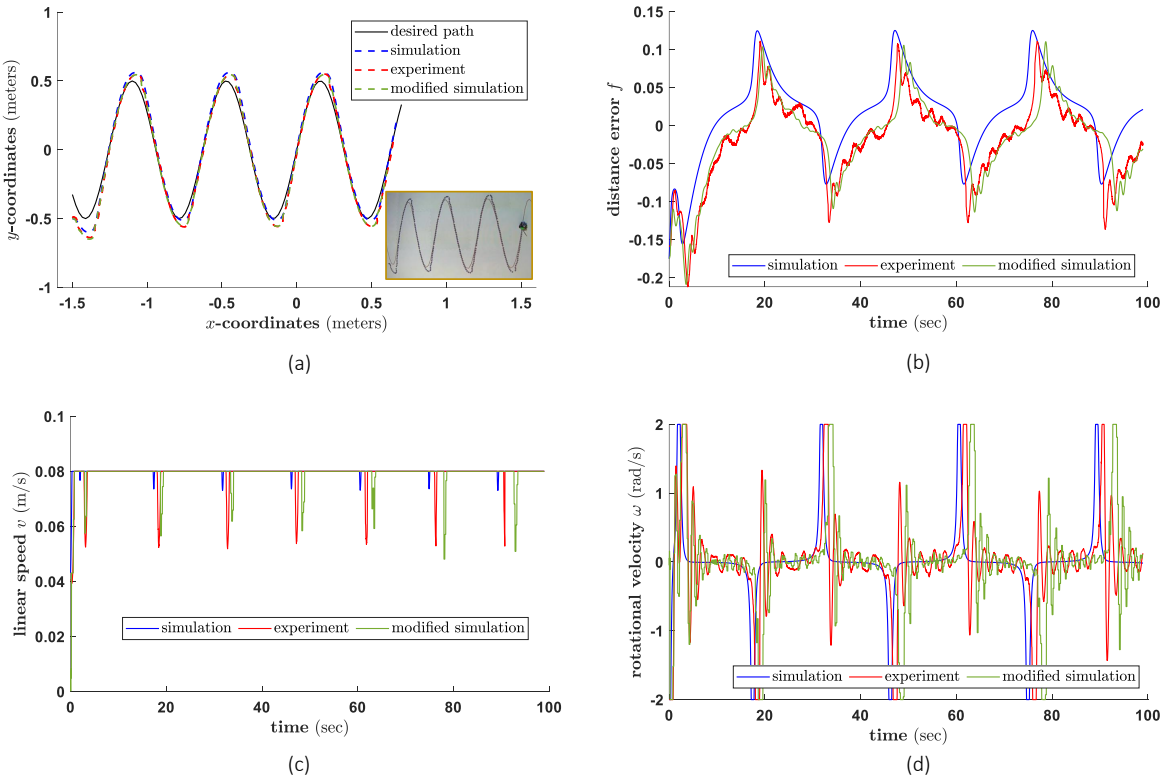

Fig. 3. Visual comparison (best visualized in color) of performance in simulation, experiment and modified simulation in the task of following a sinusoidal path. (a) Desired path and robot's trajectories in three cases. A snapshot from the experiment is shown in the inset. (b) Plot of distance error f over time. Note, the distance error in level curve method is not Euclidean distance and may have different units for different paths represented by different implicit equations. Therefore, following existing literature [10], [22] on level curve-based method, we do not include any units for the LC distance error. (c): Plot of linear speed v over time. (d): Plot of rotational velocity ω over time.

A. *Experiment-aware training*

As mentioned earlier, the initial training of the DNNs is performed using simulated data from the nominal kinematic model of the robot. The learned controller is then employed in a real robot to follow any arbitrary path $f(x, y) = 0$. However, we observe significant difference between the simulation and experiment, subjected to the learned controller. Fig. 3 compares the trajectory, distance error f, and control inputs: linear speed v and rotational velocity ω, over time for simulation and experiment in task of following a sinusoidal path. Simulation is performed using the robotarium simulator, which uses the nominal kinematic model of the robot. In experiment, we observe some ripples in distance error and rotational velocity (Fig. 3(b,d)), which are not present in simulation. Real systems often involve different sources of uncertainties that contribute to the discrepancy between the simulation and experiment. We investigate adding several sources of uncertainties in simulation, for example, delay, sensor noise, actuator noise etc. We observe that a modified simulation involving a combination of delay, reduced actuation rate, sensor offset and noise partially imitates the behavior observed in experiment (Fig. 3). Though the modified model of simulation does not imitate the real system accurately, it facilitates controller selection, saving interactions with the actual robot with suboptimal controllers. We use this modified simulation model to choose the best candidate among several controllers trained with different settings of hyperparameters. The overall process is summarized as follows. First, we train multiple controllers with different settings of hyperparameters and choose a controller based on their performance in the simulator. The chosen controller is then employed to the real robot to collect data on its experimental behavior. Next, we develop a modified simulation model that (partially) mimics the behavior observed in experiment. Now, we repeat the controller selection using the modified simulation model to finalize a controller for the rest of the experiments. The rest of the results shown are from experiments only, unless otherwise mentioned.

B. *Impact of non-constant linear speed*

As mentioned earlier, decreasing the linear speed at a sharp turn of the path can reduce the deviation from the path. To evaluate the impact of non-constant linear speed, we compare the performance of two scenarios. In one case, only the rotational velocity ω is controlled, while the linear speed v is kept constant at $v_{\max}$ (output of DNN_C corresponding to v is tied to $v_{\max}$, and $\lambda_1 = \lambda_2 = 0$ is used in (6)). In the other case, both the rotational velocity ω and the linear speed v are controlled using DNN_C. Fig. 4 compares the trajectory, distance error f, and control inputs: linear speed v and rotational velocity ω, over time for the two cases in the task of following a linear spline with a turn of $3\pi/4$. Larger

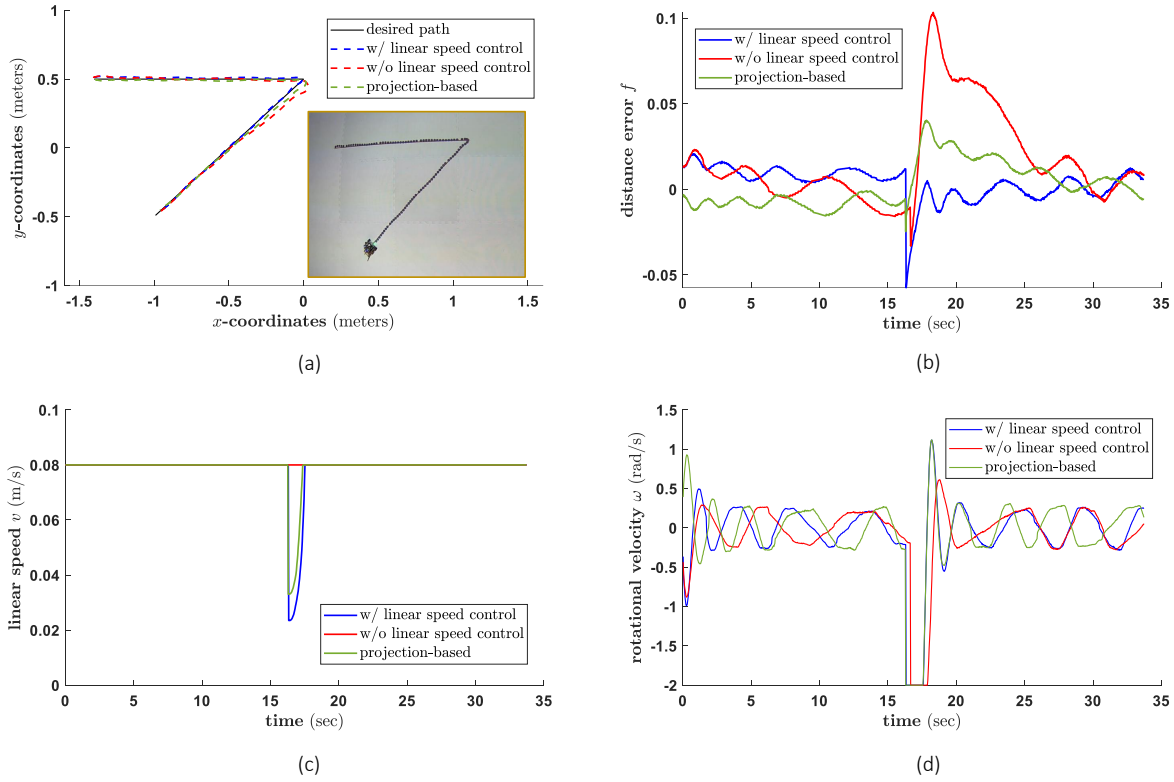

Fig. 4. Visual comparison (best visualized in color) of performance with and without linear speed control in the task of following a linear spline path with a single turn of $3\pi/4$. Performance of the controller learned using projection-based framework is shown as well (details are given in section IV-E). (a) Desired path and robot's trajectories in all three cases. A snapshot from the experiment with linear speed control is shown in the inset. (b) Plot of distance error f (crosstrack error d for the projection-based case) over time. (c): Plot of linear speed v over time. (d): Plot of rotational velocity ω over time.

deviation from the path right after the turning point is observed when linear speed is fixed at $v_{\max}$ (Fig. 4(a,b)). We compare the two cases quantitatively, in terms of the metrics defined in section III-D, for different turn angles in Table I. Notice that the equations of the two line segments are different, and the robot has to switch from one path segment to the subsequent one. For this, we follow the approach as suggested in [10] by allowing the robot to pursue the next path before it reaches the turning point. We use a distance threshold of 5 cm when approaching the turning point to make the switch.

TABLE I
EVALUATION METRICS, WITH AND WITHOUT LINEAR SPEED CONTROL, FOR FOLLOWING LINEAR SPLINE PATHS WITH A SINGLE TURN OF DIFFERENT ANGLES

Turn angle	f_{mean}		$f_{\max}$		ω_{mean}(rad/s)	
	w/o	w/	w/o	w/	w/o	w/
$\pi/4$	0.0122	0.0053	0.0453	0.0366	0.1842	0.2099
$\pi/2$	0.0090	0.0053	0.0475	0.0468	0.2140	0.2701
$3\pi/4$	0.0207	0.0081	0.1034	0.0574	0.2491	0.2724
π	0.0324	0.0154	0.0826	0.0518	0.2519	0.2030

Fig. 5 shows the contour plots of the rotational velocity and the linear speed for the controller when trained to control both. The linear speed remains at $v_{\max}$ for a region around $\psi = 0$ and declines as the heading error ψ increases beyond that region (Fig. 5(a)). From Fig. 5(b), it can be observed that the partial gain of rotational velocity controller with respect to the distance error f is higher when the heading error ψ is small compared to the case when ψ is large.

C. Trade-off between oscillation and tracking error

The amplitude of ripples in distance error f and rotational velocity ω can be reduced by increasing the coefficient of heading error q in the Lyapunov function $V(f, \psi) = f^2 + q\psi^2$. However, increasing q leads to higher mean tracking error.

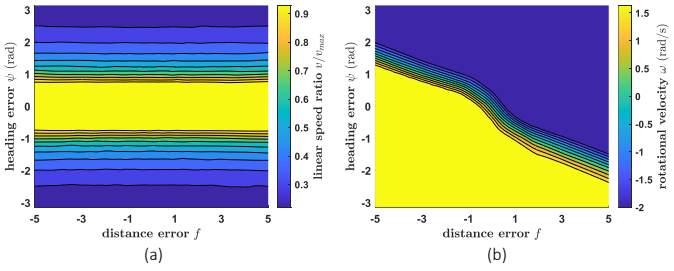

Fig. 5. Contour plots (best visualized in color) of the learned (a) linear speed controller and (b) rotational velocity controller.

The trade-off between oscillation and the mean tracking error is shown in Fig. 6 and Table II for a pair of values of q in the task of following a cubic spline path. The robot switches from one cubic path segment to the subsequent one using the same approach mentioned in IV-B. $f_{\max}$ error and mean rotational speed (control effort) ω_{mean} decreased for a higher value of q, but mean error f_{mean} is increased.

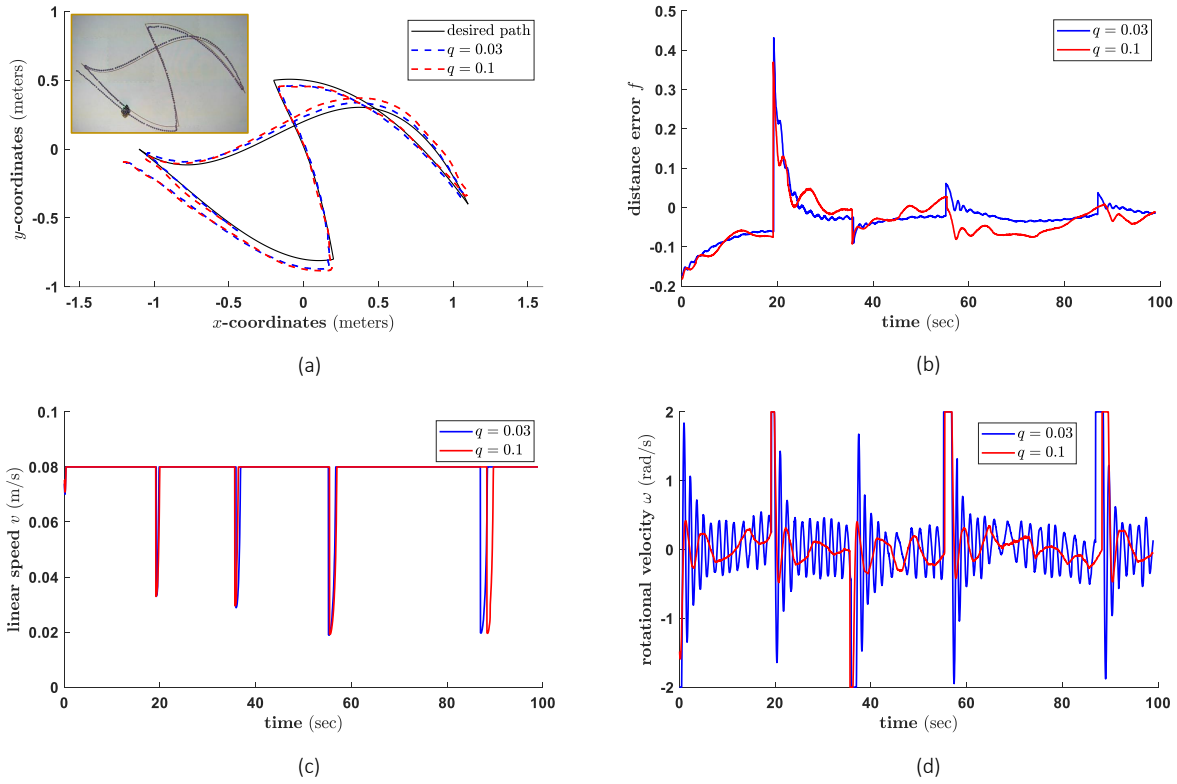

Fig. 6. Visual comparison (best visualized in color) of performance for two different values of q in the task of following a cubic spline path. (a) Desired path and robot's trajectories in both cases. A snapshot from the experiment with $q = 0.03$ is shown in the inset. (b) Plot of distance error f over time. (c): Plot of linear speed v over time. (d): Plot of rotational velocity ω over time.

TABLE II
EVALUATION METRICS IN THE TASK OF FOLLOWING A CUBIC SPLINE PATH FOR TWO DIFFERENT VALUES OF q

q	f_{mean}	$f_{\max}$	ω_{mean}(rad/s)
0.03	0.0427	0.4319	0.4143
0.1	0.0502	0.3700	0.2416

D. Comparison with PD controller

We compare the DNN controller and a PD controller in task of following a linear path with no turn versus linear spline paths with a single turn of different angles (same as the one used in section IV-B). Note, the derivative of the distance error can be analytically computed as a function of the heading error: $\dot{f} = \|\nabla f\| v \sin(\psi)$ [10]. Consequently, the state-feedback controller $\omega = -K_p f - K_d \psi$ effectively works as a PD controller and performs better than the usual PD controller that requires numerical derivatives from noisy measurements. We use $K_p = 5, K_d = 5$. Table III shows the quantitative comparison. The PD controller has lower tracking error for the path with no turn, whereas the opposite is true for the paths with turn. Furthermore, the mean rotational speed ω_{mean} of the DNN controller is lower than that of the PD controller in each cases. Fig. 7 shows the simulated trajectories (with actuation bound of the real robot applied) of the robot from different initial positions under the PD controller and the DNN controller in the task of following an elliptical path. We kept the initial orientation of the robot aligned with the target orientation for these plots. The DNN controller has larger region of attraction compared to the PD controller.

TABLE III
COMPARISON WITH PD CONTROLLER IN TASK OF FOLLOWING LINEAR SPLINE PATHS WITH A SINGLE TURN OF DIFFERENT ANGLES

Turn angle	f_{mean}		$f_{\max}$		ω_{mean}(rad/s)	
	PD	NN	PD	NN	PD	NN
0	0.0046	0.0099	0.0100	0.0202	0.2320	0.1357
$\pi/4$	0.0192	0.0053	0.0596	0.0366	0.2759	0.2099
$\pi/2$	0.0077	0.0053	0.0444	0.0468	0.3092	0.2701
$3\pi/4$	0.0202	0.0081	0.0840	0.0574	0.3077	0.2724
π	0.0248	0.0154	0.0977	0.0518	0.3293	0.2030

E. Projection-based path following

The proposed DNN-based method does not specifically require the level curve-based framework for error computation and can be applied to the projection-based framework as well. In projection-based path following, the desired path is characterized by its curvature $\kappa(s)$, where s denotes the curvilinear abscissa along the path [3], [4]. The distance/crosstrack error d in this case is defined as the distance between the actual robot and its orthogonal projection on the path. The heading error ψ is computed as the angle between robot's orientation and the tangent to the path at the projection point. The path

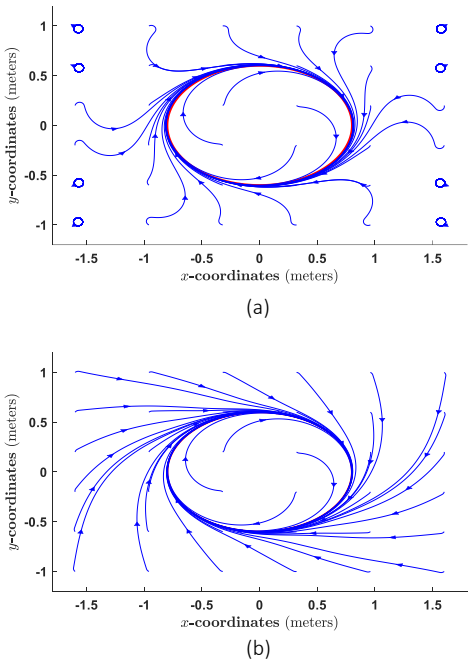

Fig. 7. Simulated trajectories of the robot from different initial positions subjected to (a) the PD controller and (b) the DNN controller when the target path is an ellipse.

following error dynamics in this framework is given by [3]

$$\dot{d} = v \sin \psi,$$
$$\dot{\psi} = \omega - \frac{\kappa(s) v \sin \psi}{1 - \kappa(s) d}. \tag{14}$$

The time derivative of the error state is computed using (14) for the predicted (by DNN_C) control vector $[v, \omega]^\top$. DNN_S predicts a vector field $[\dot{d}_s, \dot{\psi}_s]^\top$ that represents a dynamical system exponentially stable at the origin. Since the error dynamics of (14) depends on the path curvature $\kappa(s)$, we train the networks for different values of $\kappa(s) \in [-0.5, 0.5]$. We use $\alpha = 0.03$, $q = 2$, $\lambda_1 = 0.1$ and $\lambda_2 = 0.1$ for this experiment. We evaluate the learned controller in the task of following the linear spline path with a turn of $3\pi/4$ used in section IV-B. Trajectory, distance error and control inputs for this controller are shown in Fig. 4 with green lines.

V. Conclusion

In this paper, we experimentally demonstrated the applicability of DNN-based controller in the path following task for unicycle robots. We presented a learning method for DNN-based controller that can be trained without initialization or supervision from any other controller known a priori and can be used solely during operation. The proposed method jointly learns an error dynamics hypothesis that is stable at the origin and a control law to realize that hypothesized dynamics in closed-loop. This paper illustrates the practicality of a DNN controller, learned in a model-based simulation framework, in real system using the path following example. Similar strategy can be applied in other practical systems as well. The proposed method can be extended to three-dimensional path following for aerial and underwater vehicles by representing the path as the intersection of two surfaces.

References

[1] A. P. Aguiar, D. B. Dačić, J. P. Hespanha, and P. Kokotović, "Path-following or reference tracking?: An answer relaxing the limits to performance," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 167–172, 2004.

[2] P. Walters, R. Kamalapurkar, L. Andrews, and W. E. Dixon, "Online approximate optimal path-following for a mobile robot," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 4536–4541.

[3] J. Płaskonka, "Different kinematic path following controllers for a wheeled mobile robot of (2, 0) type," *Journal of Intelligent & Robotic Systems*, vol. 77, no. 3, pp. 481–498, 2015.

[4] A. Micaelli and C. Samson, "Trajectory tracking for unicycle-type and two-steering-wheels mobile robots," Ph.D. dissertation, INRIA, 1993.

[5] K. Do and J. Pan, "Global output-feedback path tracking of unicycle-type mobile robots," *Robotics and Computer-Integrated Manufacturing*, vol. 22, no. 2, pp. 166–179, 2006.

[6] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino, "Closed loop steering of unicycle like vehicles via lyapunov techniques," *IEEE robotics & automation magazine*, vol. 2, no. 1, pp. 27–35, 1995.

[7] M. Egerstedt, X. Hu, and A. Stotsky, "Control of mobile platforms using a virtual vehicle approach," *IEEE transactions on automatic control*, vol. 46, no. 11, pp. 1777–1782, 2001.

[8] M. Michalek and K. Kozlowski, "Vector-field-orientation feedback control method for a differentially driven vehicle," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 1, pp. 45–65, 2009.

[9] W. Yao, H. G. de Marina, and M. Cao, "Vector field guided path following control: Singularity elimination and global convergence," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 1543–1549.

[10] A. Morro, A. Sgorbissa, and R. Zaccaria, "Path following for unicycle robots with an arbitrary path curvature," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 1016–1023, 2011.

[11] R. Fierro and F. L. Lewis, "Control of a nonholonomic mobile robot using neural networks," *IEEE transactions on neural networks*, vol. 9, no. 4, pp. 589–600, 1998.

[12] G. Zhang, X. Zhang, and Y. Zheng, "Adaptive neural path-following control for underactuated ships in fields of marine practice," *Ocean Engineering*, vol. 104, pp. 558–567, 2015.

[13] J. Velagic, N. Osmic, and B. Lacevic, "Neural network controller for mobile robot motion control," *World Academy of Science, Engineering and Technology*, vol. 47, pp. 193–198, 2008.

[14] T. Hu and S. X. Yang, "Real-time motion control of a nonholonomic mobile robot with unknown dynamics," in *Proceedings of the Computational Kinematics Conference, Seoul*, 2001.

[15] W. Zhu, F. Raza, and M. Hayashibe, "Reinforcement learning based hierarchical control for path tracking of a wheeled bipedal robot with sim-to-real framework," in *2022 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2022, pp. 40–46.

[16] W. Zhu, X. Guo, Y. Fang, and X. Zhang, "A path-integral-based reinforcement learning algorithm for path following of an autoassembly mobile robot," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, pp. 4487–4499, 2019.

[17] H. Shen and C. Guo, "Path-following control of underactuated ships using actor-critic reinforcement learning with mlp neural networks," in *2016 Sixth International Conference on Information Science and Technology (ICIST)*. IEEE, 2016, pp. 317–321.

[18] Y. Sun, X. Ran, G. Zhang, X. Wang, and H. Xu, "Auv path following controlled by modified deep deterministic policy gradient," *Ocean Engineering*, vol. 210, p. 107360, 2020.

[19] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, and M. Egerstedt, "The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems," *IEEE Control Systems Magazine*, vol. 40, no. 1, pp. 26–44, 2020.

[20] P. Saha, M. Egerstedt, and S. Mukhopadhyay, "Neural identification for control," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4648–4655, 2021.

[21] J. Z. Kolter and G. Manek, "Learning stable deep dynamics models," *Advances in neural information processing systems*, vol. 32, 2019.

[22] K. D. Do, "Global path-following control of stochastic underactuated ships: A level curve approach," *Journal of dynamic systems, measurement, and control*, vol. 137, no. 7, p. 071010, 2015.