General Dynamic Neural Networks for the Adaptive Tuning of an Omni-Directional Drive System for Reactive Swarm Robotics

Hanqing Zhao
IRIDIA
Université libre de Bruxelles
Brussels, Belgium
hanqing.zhao@ulb.ac.be

Marco Dorigo
IRIDIA
Université libre de Bruxelles
Brussels, Belgium
mdorigo@ulb.ac.be

Michael Allwright
IRIDIA
Université libre de Bruxelles
Brussels, Belgium
michael.allwright@ulb.ac.be

***Abstract*— We demonstrate the use of general dynamic neural networks (GDNNs) for the online tuning of an omni-directional drive system for reactive swarm robots. The drive system used in this work consists of four motor-encoder-microcontroller modules each constituting a single-input single-output (SISO) proportional, integral, and differential (PID) control system. For a given target velocity, a neural network generates the parameters for each PID control system. In this paper, we evaluate and compare two different network structures for generating the PID parameters for the control systems using a hardware platform that we also presented in this paper. We analyze the performance of the system with respect to ISO performance indicators, our results show that both network structures are able to learn and tune the parameters for each PID control system to increase the accuracy of the drive system in comparison to fixed untuned PID parameters that are close to the output of a randomly initialized network.**

I. Introduction

The use of differential drive in mobile robot platforms is often selected due to low cost and simplicity of control. However, the use of differential drive can limit a robot's agility, often requiring multiple maneuvers to reach a given pose. To further complicate matters, these maneuvers must often be performed while respecting other constraints such as avoiding obstacles in the environment.

Performing these maneuvers is especially problematic in swarm robotics systems, where many robots may be cooperating to perform a given task in a confined space. For example, in the work of Allwright et al., swarms of robots search an environment for unused blocks that can be attached to one or more structures [1]. In the event that a robot has approached a structure with an unsuitable pose for deposition, the robot must reverse and adjust its pose while not colliding with other parts of the structure or other robots. Furthermore, since the described system aims to be reactive, performing such a maneuver is not ideal since it requires a form of planning.

One solution to these problems is to use omni-directional drive instead of differential drive. Although more expensive and more complex to control, the advantage of using omni-directional drive is that the robot can move in any direction and in many cases may eliminate the requirement of performing multiple maneuvers to reach a target pose. The tuning of omni-directional drive systems is however, more complex than that for a differential drive system due to issues such as slippage and interference between the wheels.

In this paper, we study how general dynamic neural networks [2] can be used to automatically learn the relationship between the target velocity of an omni-directional drive system and a set of parameters for its control systems. We conduct this study using an omni-directional drive system which we have designed for this research. Our system consists of a platform and four equally-spaced omni wheels, each driven by an independent motor-encoder-microcontroller module, in which the microcontroller generates the pulse-width modulation (PWM) control signals for the motor driver and tracks the output signals from the shaft encoder. Each microcontroller implements a proportional, integral, and differential (PID) control system [3], whose parameters are continuously updated by a GDNN with respect to the target velocity and the network's current training. Using separated motor-encoder-microcontroller modules for each omni wheel enables a non-lossy capture of high-frequency shaft encoder output thus gives the system a better scalability in the number of wheels.

We compare in this paper two network structures for generating the parameters of the PID control systems. We refer to these network structures as the *ensemble* structure and *separated* structure. The key difference between these network structures is that the ensemble structure uses a single network for all of the control systems, while the separated structure has an independent network for each control system. The primary hypothesis is that the ensemble structure with an appropriate architecture will outperform the separated structure, since the former will be able to learn and to encode the influence of each control system on the other control systems. To test this hypothesis, we evaluate the performance of our system with respect to the indicators specified in ISO 9283:1998. To encourage further research in this area, we have made the hardware and software of our platform open source and available as a project on the Open Science Foundation [4].

II. State of the Art

The rotation speed of each wheel in an omni-directional drive system is not independent and a change in the velocity of one wheel may influence the other wheels. In other words, the relationship between the velocities of the wheels in an omni-directional drive system is non-linear. As previously described, the speed of each wheel in our platform is regulated by a PID control system. If the parameters to these PID controllers are fixed, the controllers are linear and have limited ability to handle the non-linearity of the system.

A widely used method for applying PID control to a non-linear system is to tune the parameters of the PID controller online [5]. Online tuning of PID controllers has been thoroughly studied and major approaches include fuzzy inference [6], stochastic approximation from data [7], heuristic optimization [8], and their combinations [9]. More recently, machine learning techniques such as reinforcement learning [10] and supervised learning [11] have been applied to tune PID controllers.

In related work on the tuning of multiple-input multiple-output (MIMO) systems, Sun et al. proposed a fuzzy inference approach that tunes each SISO PID controller separately [12]. This approach, however, is not suitable to an omni-directional drive system since the method cannot compensate for the interference between the individual PID controllers. Boyd et al. constructed a single MIMO PID controller, where the PID parameters were determined by convex-concave optimization [13]. This approach, however, has the drawback of being difficult to tune without a transfer function of the system being controlled. The PID parameters for a MIMO controller can also be evolved using a genetic algorithm [14]. This method, however, requires evaluating a large number of candidate solutions, resulting in a long convergence time and a possible overfitting to a particular instance of the system.

In recent work, Reichensdörfer et al. demonstrated a neural network architecture which was proved to be suitable for the online tuning of PID controllers [2]. This architecture is known as a general dynamic neural network (GDNN) and is characterized by the possibility of having arbitrary time-delayed recurrent connections between any two neurons. In this paper, we aim to advance the state of the art by demonstrating how GDNNs can be used to learn the relationship between an input velocity to an omni-directional drive system and the corresponding real time tuning for its PID controllers. We also compare two different network structures to test our hypothesis that using a single network to provide the parameters for all the PID controllers will enable a GDNN to learn the non-linearity of the system more effectively.

III. Hardware Description

The omni-directional platform in this work consists of four 38 millimeter omni wheels each connected to an EMG-30 motor and shaft encoder module via a gear pair with a 6:5 reduction ratio. The EMG-30 motor is driven by a DRV8871 motor driver which takes a pulse-width modulation signal from a dedicated ATMega328P as its input. The output of the shaft encoder is connected to the port change interrupt pins of the same microcontroller, allowing it to measure the rotation of the output shaft and to implement a closed loop PID controller. Each microcontroller communicates with a

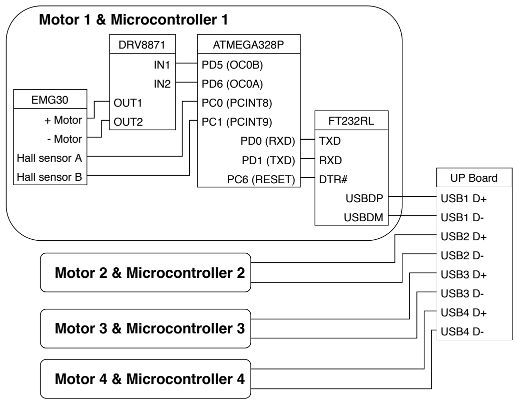

Fig. 1. High-level schematic of the electronic hardware.

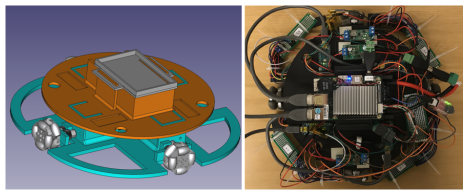

Fig. 2. Structural design (left) and photo (right) of the drive platform

main microprocessor board using a FT232RL UART to USB bridge. A high-level schematic of these components is shown in Fig. 1.

The electronic components and wheels are mounted to a circular platform. The design of this platform was based on two principles: (i) minimize the overall size of the platform and (ii) keep the center of gravity of the platform as close as possible to the geometric center. Fig. 2 shows the fully-assembled platform.

IV. Software Description

A. Microcontroller Firmware

The microcontroller for each motor runs the same firmware. This firmware implements two main components. The first component is a command-based interface, which executes commands from the microprocessor and returns their results to the microprocessor over a UART interface. These commands enable setting the velocity of the motor, reading back the velocity error, and setting the parameters of the PID controllers.

The second component is the motor controller. The motor controller measures the speed of the motor's shaft, executes the PID controllers, and controls the duty cycle of the PWM signal to the motor. Measuring the speed of the motor is implemented using a port change interrupt, which increments or decrements a counter depending on whether the motor's shaft is rotating forwards or backwards. The execution of the PID controller is triggered by a timer interrupt, at each interruption the counter for measuring the position of the motor's shaft is reset to zero and the duty cycle of the PWM signal is updated.

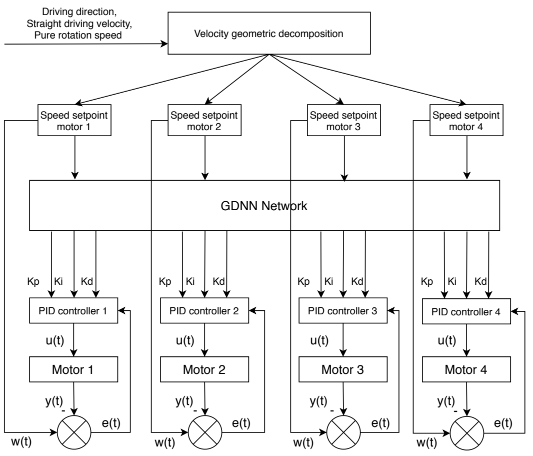

Fig. 3. The control system with the ensemble network structure.

On timer interrupt t, the output of the PID controller $p(t)$ is updated according to the previous output $p(t-1)$ and the rotation speed error $e(t)$ during the last three time steps:

$$\begin{aligned} p(t) = p(t-1) &+ K_p\big(e(t) - e(t-1)\big) \\ &+ K_i e(t) \\ &+ K_d\big(e(t) - 2e(t-1) + e(t-2)\big) \end{aligned} \tag{1}$$

Where K_p, K_i, K_d are the parameters of the PID controller.

B. Microprocessor Software

The microprocessor in our design is provided by an Intel atom-based UP Board running Ubuntu Linux. The board communicates with the four microcontrollers using UART to USB bridges. The software running on the microprocessor is written in C++ and is responsible for communicating with the microcontrollers and update its neural network parameters at a rate of 3.33 Hz.

C. Neural Network Structure and Architecture

For our system, we have investigated two different network structures: the ensemble structure and separated structure. For the ensemble structure (Fig. 3), the network contains four neurons in its input layer (for the target velocity of each motor) and twelve neurons in its output layer (three parameters for each of the four PID controllers). This configuration is trained on-the-fly with the goal of minimizing the error at all four motors. In contrast to the ensemble structure, the separated structure consists of four separate networks each of which with a single neuron in its input layer (for the target velocity of the corresponding motor) and three neurons in its output layer (for the parameters of the PID controller of the corresponding motor controller). These four networks are trained simultaneously but independently with the goal of minimizing the error at their corresponding motor.

To study the performance of the GDNN architecture in the ensemble and separated network setups, we define three configurations for each structure. The first configuration is the baseline configuration. The baseline configuration is a four-layer GDNN where the first hidden layer consists of 15 neurons and the second hidden layer has the same number of neurons as the output layer (12 for ensemble and 3 for separated structure). This number of neurons has achieved relatively good performance in our experiments, the relationship between the number of neurons and system performance, however, will need to be examined more thoroughly in future work. Fig. 4 shows the baseline configuration applied to the separated network structure.

We study the influence of the number of hidden layers by removing the second hidden layer from the baseline configuration to form a three-layer GDNN, which we refer to as the reduced-layer configuration. This configuration is the most similar to the work of Reichensdörfer [2]. To study the role of the recurrent connections in the architecture, we removed all the recurrent connections from the baseline configuration to create the feed-forward configuration.

With the exception of the feed-forward configuration, the position of the recurrent connections in our GDNN architectures is identical to that described in the work of Reichensdörfer where each neuron in the input layer has a recurrent delayed connection of one time step from all of the neurons in the first hidden layer.

Referring to Fig. 4, the $t-1$ neurons output the sum of all input values with a delay of one time step. In contrast to normal neurons, the $t-1$ neurons have no weight, bias, nor activation functions. The other neurons in the input and hidden layers use the hyperbolic tangent activation function. The neurons in the output layer use the sigmoid activation function with a scale factor.

D. Neural Network Training

GDNNs are trained using the Truncated Back Propagation Through Time (TBPTT) algorithm which is defined by two parameters (k_1, k_2) [15]. At each step of the training, given an error function $E(t)$, the GDNN parameters are updated on every ${k_1}^{\text{th}}$ step using TBPTT so that the accumulated error from the k_2 steps is minimized. For each connection weight w_{ij} and bias b_i, the change of weight Δw_{ij} and bias Δb_i can be calculated as follows, where η is the learning weight:

$$\Delta w_{ij} = -\eta \sum_{h=0}^{k_2} \frac{\partial E(t-h)}{\partial w_{ij}} \tag{2a}$$

$$\Delta b_i = -\eta \sum_{h=0}^{k_2} \frac{\partial E(t-h)}{\partial b_i} \tag{2b}$$

The error function $E(t)$ is related to the PID controller error and different for the two control system structures. For the separated structure, $e(t)$ refers to the error of the corresponding PID controller and the corresponding $E(t)$ is defined in (3a). For the ensemble structure, $e_n(t)$ refers to the error of the n^{th} PID controller and $E(t)$ is defined in (3b).

$$E(t) = \frac{1}{2}e(t)^2 = \frac{1}{2}(w(t) - y(t))^2 \tag{3a}$$

$$E(t) = \frac{1}{2}e_1(t)^2 + \frac{1}{2}e_2(t)^2 + \frac{1}{2}e_3(t)^2 + \frac{1}{2}e_4(t)^2 \tag{3b}$$

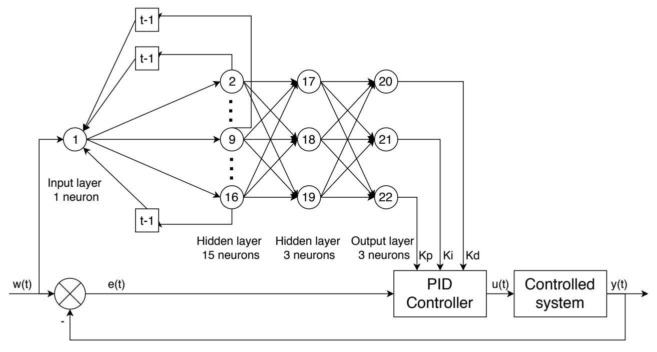

Fig. 4. Diagram of a GDNN used in the baseline separated structure.

In the remainder of this section, we show how we back propagate the error in (3a) for the separated structure in its baseline configuration (Fig. 4).

The output neurons of the network are the K_p, K_i and K_d parameters of the PID controller. In order to obtain the partial derivation of $E(t)$ with respect to each connection weight and neuron bias, we must first obtain the partial derivative $E(t)$ with respect to the three PID parameters. For parameter K_p we have:

$$\frac{\partial E(t)}{\partial K_p} = -e(t)\frac{\partial y(t)}{\partial u(t)}\frac{\partial u(t)}{\partial K_p} \tag{4}$$

Where $\frac{\partial y(t)}{\partial u(t)}$ is the Jacobian information of the controlled system, which is assumed to be constant and equal to 1 and $\frac{\partial u(t)}{\partial K_p}$ is the partial derivation of the output $u(t)$ with respect to the parameter K_p.

According to (1), we have $\frac{\partial u(t)}{\partial K_p} = e(t) - e(t-1)$. Using the same technique, we can write the partial derivative of the error function with respect to each parameter as:

$$\frac{\partial E(t)}{\partial K_p} = -e(t)(e(t){-}e(t{-}1))\frac{\partial y(t)}{\partial u(t)} \tag{5a}$$

$$\frac{\partial E(t)}{\partial K_i} = -e(t)^2\frac{\partial y(t)}{\partial u(t)} \tag{5b}$$

$$\frac{\partial E(t)}{\partial K_d} = -e(t)(e(t){-}2e(t{-}1) + e(t{-}2))\frac{\partial y(t)}{\partial u(t)} \tag{5c}$$

Using these partial derivatives, the connection weight and neuron bias of each neuron can be updated as follows, where $O_t(N_i)$ is the output value of neuron i at time step t:

$$\begin{aligned}\Delta w_{ij} &= -\eta\sum_{h=0}^{k_2}\frac{\partial E(t-h)}{\partial w_{ij}} = -\eta\sum_{h=0}^{k_2}\Bigg[\frac{\partial E(t-h)}{\partial K_p}\frac{\partial O_{t-h}(N_{22})}{\partial w_{ij}}\\ &+\frac{\partial E(t-h)}{\partial K_i}\frac{\partial O_{t-h}(N_{21})}{\partial w_{ij}} + \frac{\partial E(t-h)}{\partial K_d}\frac{\partial O_{t-h}(N_{20})}{\partial w_{ij}}\Bigg]\end{aligned} \tag{6a}$$

$$\begin{aligned}\Delta b_i &= -\eta\sum_{h=0}^{k_2}\frac{\partial E(t-h)}{\partial b_i} = -\eta\sum_{h=0}^{k_2}\Bigg[\frac{\partial E(t-h)}{\partial K_p}\frac{\partial O_{t-h}(N_{22})}{\partial b_i}\\ &+\frac{\partial E(t-h)}{\partial K_i}\frac{\partial O_{t-h}(N_{21})}{\partial b_i} + \frac{\partial E(t-h)}{\partial K_d}\frac{\partial O_{t-h}(N_{20})}{\partial b_i}\Bigg]\end{aligned} \tag{6b}$$

Due to the existence of delayed connections, it is difficult to find an analytical solution to the partial derivative of a neuron's output with respect to its internal weight or bias $\frac{\partial O_{t\text{-}h}(N_i)}{\partial w_{ij}}$. However, these partial derivatives can be estimated using the following numerical approach:

$$\frac{\partial O_{t-h}(N_i)}{\partial w_{ij}} = \frac{O_{t-h}(N_i, I, W, b) - O_{t-h}(N_i, I, W_{w_{ij}-\epsilon}, b)}{\epsilon} \tag{7}$$

Where $O_{t-h}(N_i, I, W, b)$ represents the network's output from neuron N_i at h steps prior to current time step t, given an input vector I which contains all the previous inputs from $t-h$ to t, a connectivity weight matrix W and a bias matrix b. $W_{w_{ij}-\epsilon}$ is a modified weight matrix where the small value ϵ has been subtracted from each element in the connectivity weight matrix. Note that in our experiments we use TBPTT configuration $k_1 = k_2$ thus weight matrix W does not change in the error bootstrapping window between the current step t and the step $t - k_2$. Moreover, we have set the value of ϵ to be the square of our data type's precision level following the recommendation of Reichensdörfer [2]. In our case, ϵ is set to 10^{-7}.

V. Discussion and Results

A. Test Trajectory and Performance Criteria

The performance of the drive platform's control system is evaluated using a test trajectory, which is a set of velocity vectors $\overrightarrow{v}$ and times t at which each velocity vector should be applied.

ISO 9283:1998 defines several performance indicators for evaluating a robot's ability to traverse a given trajectory. These performance indicators include the Positioning Path Accuracy (PPA) and Path Velocity Characteristics (PVC). PPA is defined as the ability for a robot to move along a given path n times. For a given test trajectory, the corresponding PPA can be calculated as follows:

$$\begin{aligned}\text{PPA} &= \frac{\sum_{j=1}^{n}\text{PPA}_j}{n}, j \in [1, n]\\ \text{PPA}_j &= \frac{1}{m}\sum_{i=1}^{m}\sqrt{(x_{ij} - x_{cij})^2 + (y_{ij} - y_{cij})^2}\end{aligned} \tag{8}$$

Where m denotes the number of position measurements on the test trajectory, n is the number of repetitions, (x_{ij}, y_{ij}) is the i^{th} position of the platform measured during the j^{th} repetition, and (x_{cij}, y_{cij}) is the corresponding command position on the test trajectory of measured position (x_{ij}, y_{ij}).

In addition to PPA, we also consider two PVC indicators: the Path Velocity Accuracy (PVA) and the Path Velocity Repeatability (PVR).

The PVA is defined as the error between the command velocities and the mean value of the measured velocities during n traverses of a test trajectory and is expressed as a percentage of the command velocity. The PVA used in our experiments is calculated as follows:

$$\begin{aligned}\text{PVA} &= \frac{1}{m}\sum_{i=1}^{m}\frac{\overline{v_i} - v_{ci}}{v_{ci}} \times 100\%\\ \overline{v_i} &= \frac{1}{n}\sum_{j=1}^{n}v_{ij}\end{aligned} \tag{9}$$

Where m is the number of velocity measurements along the test trajectory, n is the number of repetitions, v_{ij} is the velocity for i^{th} measurement during the j^{th} run, and v_{ci} is the command velocity at the time of the i^{th} velocity measurement. The PVR is a measure of the error between each velocity command and the actual velocity obtained, which is defined as follows:

$$\begin{aligned} \mathrm{PVR} &= \pm(\frac{3S_v}{v_c} \times 100\%) \\ S_v &= \sqrt{\frac{\sum_{j=1}^{n}(\overline{v_j} - \overline{v})^2}{n-1}} \\ \overline{v} &= \frac{1}{n}\sum_{j=1}^{n}\overline{v_j}, \quad \overline{v_j} = \frac{1}{m}\sum_{i=1}^{m}v_{ij} \end{aligned} \tag{10}$$

Where m, n, v_{ij} share their meaning with the PVA indicator and v_c denotes the command velocity. While the PPA from ISO 9283:1998 reports the error between the command trajectory and the average trajectory from all traversals, the PPA used in our evaluation is the average of the errors between the command trajectory and each traversal of the trajectory. This departure from the standard was required due to the clock in our tracking system is not synchronized with the clock in the driving platform.

B. Evaluation

The system is evaluated using two different test trajectories, a cross trajectory with uniform velocity (CTUV) and a rectangle trajectory with variable velocity (RTVV). The CTUV trajectory drives the robot in four directions each separated by 90 degrees. The robot moves in each direction and returns to the origin before moving in the next direction so that the trajectory forms a cross shape. The command velocity for each direction is 14.92 cm/s and is executed for 4 seconds. In the RTVV trajectory, the robot drives in four directions each separated by 90 degrees (without returning to the origin) to form a rectangle. On each side of the rectangle, the robot is instructed to accelerate at 1.658 cm/s^2 from 0 cm/s to 14.92 cm/s over 9 seconds and then to decelerate at the same rate back to 0 cm/s.

The CTUV trajectory is used for evaluating the PPA and PVC of the robot at constant speed, while the RTVV trajectory is used for PVC evaluation with variable speed. In the PPA calculation, the corresponding command position on the CTUV trajectory x_{cij} of measured position (x_{ij}, y_{ij}) in the sequence is assumed to be the closest point on the x-axis or y-axis of the normalized position coordination system.

Before each experiment, the weights in the network are randomly initialized following a uniform distribution between 0 and 1. Following initialization, for each target trajectory the network is pre-trained by traversing the trajectory three times. After pre-training, the experiment starts and the robot attempts to move along the specified trajectory (12 times for CTUV, 5 times for RTVV). The initial position and orientation of the robot is normalized at the beginning of each experiment, following the pre-training traversals. During the experiments, the speed error is recorded by the robot itself and the position sequence is recorded by an overhead tracking system [16].

C. Results

For discussing our results, we will denote a system structure and architecture using the scheme [Structure]-[Network Type]-[Layers]. For example, E-F-4 represents the ensemble structure with a 4-layer feed-forward network (without recurrent connections), while S-R-4 represents the separated structure with a 4-layer network (with recurrent connections).

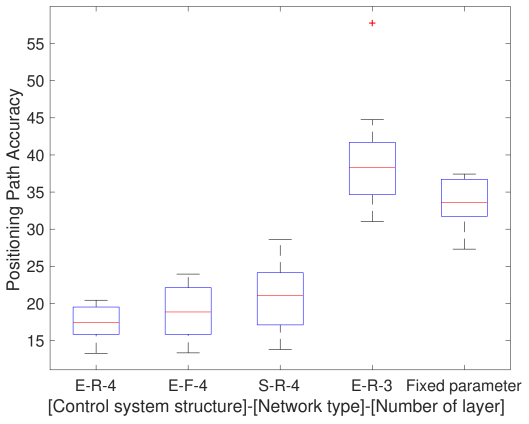

Fig. 5. Position Path Accuracy (PPA) of the system structures/architectures for the CTUV trajectory.

To demonstrate the ability of different structures and architectures to learn appropriate PID parameters from randomly initialized state, the GDNN network-based systems are also compared with a system using fixed untuned parameters, where K_p, K_i, K_d have been fixed to 0.5 which reflects the initial state of the controller, as the PID parameters generated from a randomly initialized network are around 0.5.

This comparison serves as a baseline that demonstrates the ability of the GDNN-based systems to learn their parameters. The PPA results for the different systems and architectures are shown in Fig. V-C. The presented data is from five runs of each of these systems/architectures using the CTUV trajectory. These results show that the ensemble structure in its baseline configuration (E-R-4) is the most accurate (lowest average PPA) and has the smallest variance.

To further study the velocity accuracy, we consider the Path Velocity Accuracy (PVA) indicator with 12 traversals of the CTUV trajectory. We analyze the data from these runs over (i) the whole trajectory (WT) and (ii) the part of the trajectory (PT) where the robot has accelerated to 90% of its target velocity and before it begins to decelerate. The WT-PVA and the PT-PVA and their standard deviations (in brackets) for each system/architecture are reported in Table I.

TABLE I
PATH VELOCITY ACCURACY FOR THE CTUV TRAJECTORY

	Command = 21 counts/itr		Command = - 21 counts/itr	
	WT-PVA	PT-PVA%	WT-PVA%	PT-PVA%
E-R-4	−4.17 (0.23)	0.09 (0.22)	4.39 (0.29)	0.08 (0.28)
E-F-4	−4.31 (0.22)	−0.05 (0.21)	4.42 (0.27)	0.04 (0.25)
S-R-4	−5.25 (0.32)	−0.01 (0.31)	6.08 (0.33)	−0.14 (0.28)
E-R-3	−14.10 (0.52)	−0.14 (0.25)	14.3 (0.53)	−0.11 (0.28)
Fixed	−14.54 (0.25)	−0.03 (0.22)	14.61 (0.24)	−0.04 (0.19)

The results in Table I show that the E-R-4 and E-F-4 have significant better velocity accuracy during the whole traversal

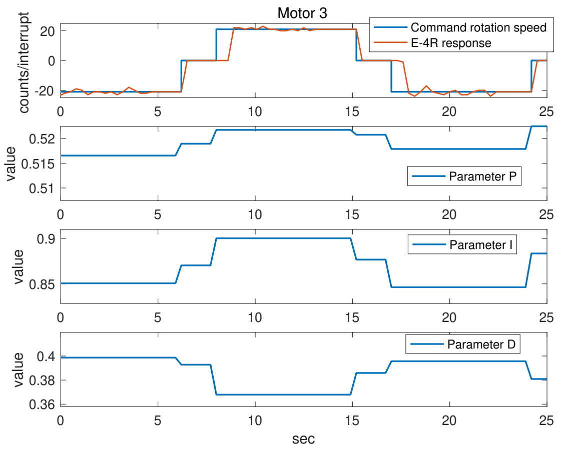

Fig. 6. Output of the PID parameters from the E-4-R configuration for a single motor during the CTUV trajectory.

of CTUV trajectory compared to the other configurations. While E-R-4 had slightly better accuracy than E-F-4, the difference is not significant if we consider the PVA during the part of the trajectory where the robot had already accelerated to 90% of its target velocity.

To further study the performance differences between E-R-4 and E-F-4 system, we consider the PVA and PVR indicators during traversals of the RTVV trajectory. The results are summarized in Table II, where E-R-4 had the best stability in low and high speed ranges. In the medium speed range, however, although E-R-4's speed accuracy (PVA) was the best, its speed stability (PVR) was the worst.

TABLE II
PATH VELOCITY CHARACTERISTICS FOR THE RTVV TRAJECTORY

	Averaged PVA		
	1-11 counts/itr	12-16 counts/itr	17-21 counts/itr
E-R-4	18.9(±2.95)%	0.843(±0.646)%	1.61(±0.485)%
E-F-4	17.4(±3.39)%	1.03(±0.563)%	1.35(±0.52)%
Fixed	27.8(±3.76)%	5.45(±0.598)%	1.11(±0.51)%
	Averaged PVR		
	1-11 counts/itr	12-16 counts/itr	17-21 counts/itr
E-R-4	43.3%	7.81%	6.0%
E-F-4	47.0%	7.23%	6.59%
Fixed	48.0%	7.34%	6.37%

We have included Fig. 6 to provide an insight into how the PID parameters evolve during a traversal of a trajectory.

VI. Conclusion

In this paper, we have implemented and compared four different GDNN-based systems/architectures for controlling an omni-directional driving platform for swarm robotics applications and have compared their performance using ISO 9283:1998. Our results showed that overall the GDNN-based ensemble structure with four layers (E-R-4) traverses the test trajectories most accurately and with the lowest variation. In the future, we intend to investigate different training schemes and to validate our approach on a more diverse range of trajectories.

To encourage other researchers to build upon the results presented in this paper, we have released the design files, a bill of materials, and the firmware for the presented work under open source licences. This content can be accessed by visiting the project repository on the Open Science Foundation [4].

Acknowledgment

This research was partially supported by the Program of Concerted Research Actions (ARC) of the Université libre de Bruxelles and by the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 846009. Marco Dorigo acknowledges support from the Belgian F.R.S.-FNRS.

References

[1] M. Allwright, N. Bhalla, and M. Dorigo, "Structure and markings as stimuli for autonomous construction," in *Proceedings of the Eighteenth International Conference on Advanced Robotics*. IEEE, 2017, pp. 296–302.
[2] E. Reichensdörfer, J. Günther, and K. Diepold, "Recurrent neural networks for PID auto-tuning," Technische Universität München, Tech. Rep., 2017. [Online]. Available: http://mediatum.ub.tum.de/doc/1381851/534530033346.pdf
[3] D. Rivera, M. Morari, and S. Skogestad, "Internal model control: PID controller design," *Industrial & Engineering Chemistry Process Design and Development*, vol. 25, no. 1, pp. 252–265, 1986.
[4] H. Zhao, M. Allwright, and M. Dorigo, "An omni-directional drive platform for experiments in GDNN-based controller architecture," 2019. [Online]. Available: http://osf.io/enw26/
[5] K. J. Åström, T. Hägglund, C. C. Hang, and W. K. Ho, "Automatic tuning and adaptation for PID controllers – a survey," *Control Engineering Practice*, vol. 1, no. 4, pp. 699–714, 1993.
[6] A. Visioli, "Tuning of PID controllers with fuzzy logic," *IEE Proc. – Control Theory and Applications*, vol. 148, no. 1, pp. 1–8, 2001.
[7] N. S. A. Shukor, M. A. Ahmad, and M. Z. M. Tumari, "Data-driven pid tuning based on safe experimentation dynamics for control of liquid slosh," in *2017 IEEE 8th Control and System Graduate Research Colloquium (ICSGRC)*. IEEE, 2017, pp. 62–66.
[8] B. Nagaraj and N. Murugananth, "A comparative study of PID controller tuning using GA, EP, PSO and ACO," in *2010 International Conference On Communication Control And Computing Technologies*. IEEE, 2010, pp. 305–313.
[9] A. Kumar and V. Kumar, "A novel interval type-2 fractional order fuzzy pid controller: Design, performance evaluation, and its optimal time domain tuning," *ISA transactions*, vol. 68, pp. 251–275, 2017.
[10] A. el Hakim, H. Hindersah, and E. Rijanto, "Application of reinforcement learning on self-tuning PID controller for soccer robot multi-agent system," in *2013 Joint International Conference on Rural Information & Communication Technology and Electric-Vehicle Technology*. IEEE, 2013, pp. 1–6.
[11] R. Hernández-Alvarado, L. García-Valdovinos, T. Salgado-Jiménez, A. Gómez-Espinosa, and F. Fonseca-Navarro, "Neural network-based self-tuning PID control for underwater vehicles," *Sensors*, vol. 16, no. 9, pp. 1429–1447, 2016.
[12] Z. Sun, R. Xing, C. Zhao, and W. Huang, "Fuzzy auto-tuning PID control of multiple joint robot driven by ultrasonic motors," *Ultrasonics*, vol. 46, no. 4, pp. 303–312, 2007.
[13] S. Boyd, M. Hast, and K. J. Åström, "MIMO PID tuning via iterated LMI restriction," *International Journal of Robust and Nonlinear Control*, vol. 26, no. 8, pp. 1718–1731, 2016.
[14] W.-D. Chang, "A multi-crossover genetic approach to multivariable PID controllers tuning," *Expert Systems with Applications*, vol. 33, no. 3, pp. 620–626, 2007.
[15] I. Sutskever, "Training recurrent neural networks," *Ph.D. Thesis, University of Toronto Toronto, Canada*, 2013.
[16] A. Stranieri, A. Turgut, M. Salvaro, L. Garattoni, G. Francesca, A. Reina, M. Dorigo, and M. Birattari, "IRIDIA's arena tracking system," IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, Tech. Rep. TR/IRIDIA/2013-013, January 2013.