Developing an Embodied Gait on a Compliant Quadrupedal Robot

Jonas Degrave[1], Ken Caluwaerts[2], Joni Dambre[1], Francis wyffels[1]

***Abstract*— Incorporating the body dynamics of compliant robots into their controller architectures can drastically reduce the complexity of locomotion control. An extreme version of this embodied control principle was demonstrated in highly compliant tensegrity robots, for which stable gait generation was achieved by using only optimized linear feedback from the robot's sensors to its actuators. The morphology of quadrupedal robots has previously been used for sensing and for control of a compliant spine, but never for gait generation. In this paper, we successfully apply embodied control to the compliant, quadrupedal Oncilla robot. As initial experiments indicated that mere linear feedback does not suffice, we explore the minimal requirements for robust gait generation in terms of memory and nonlinear complexity. Our results show that a memory-less feedback controller can generate a stable trot by learning the desired nonlinear relation between the input and the output signals. We believe this method can provide a very useful tool for transferring knowledge from open loop to closed loop control on compliant robots.**

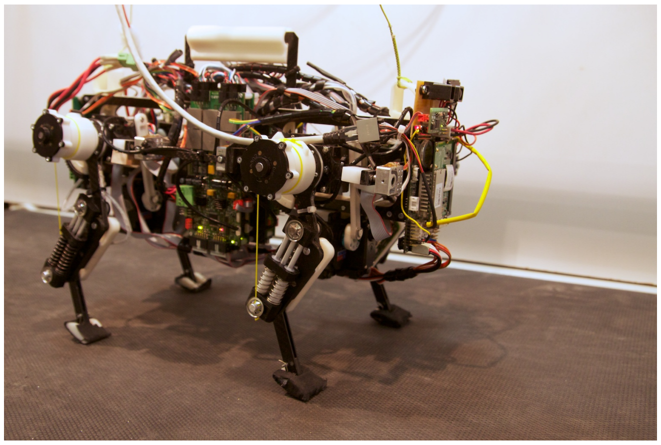
Fig. 1. The Oncilla robot on the treadmill.

I. Introduction

Compliant robots have steadily been gaining interest due to their increased ability to interact with the environment and unexpected disturbances. One way to implement compliance is by controlling impedance and joint torque [1], often referred to as active compliance. Successful implementations of this approach can be found in the well-known Big-Dog quadrupedal robot [2] or the SARCOS humanoid [3].However, robots with active compliance rely on inticrate software solutions and often complex sensors to make stiff actuation modules compliant. To reduce the complexity of the sensing and control modules, and motivated by energy efficiency and a safer robot-human-world interaction, recent trends in robotics tend to use compliant actuation modules, which can have either fixed or regulated compliance [4], [5]. The evolution towards compliance also extends to other parts of the robot as flexible materials are being used for structural parts of the robot. Examples of such robots are the quadrupedal robots Oncilla [6] and StarlETH [7], the i-HY hand, which consists of flexible fingers that can manipulate a wide variety of objects [8], and the tensegrity robot ReCTeR [9].

Unfortunately, compliant robots are harder to control due to the increased non-linear behaviour of the elastic elements. For this reason, the focus on the control algorithms has been shifted towards the morphology. The idea is that control tasks, such as locomotion control, can be partially outsourced to the compliant body elements and their interaction with the environment. This concept is known as morphological computation [10]. Recently, it was shown that certain compliant structures such as spring-mass networks have universal computing power [11]. This is highly related to the field called *physical reservoir computing*, in which the principles of reservoir computing [12]–[14] are applied to physical systems. While physical implementations exist ranging from a water bucket [15] to integrated photonics devices [16], robotic implementations are rare. Nevertheless, recent work [9], [17] illustrated that locomotion control can be outsourced to the body of a tensegrity robot: a structure composed of compression elements held together by a compliant tensile network. Stable gait generation was achieved by using only optimized linear feedback from the robot's stretch sensors to its actuator control signals.

In this paper, we test the same principle on a much less compliant robot, the quadrupedal robot Oncilla [6], shown in Fig. 1. In earlier research, the morphology of quadruped robots has been used for sensing [18] and for control of a compliant spine [19], but never for gait generation. We not only show that robust locomotion control by a simple mapping from the rotary encoders in the motors is possible, but we also investigate under which conditions this can be achieved.

When physical reservoir computing is applied to robotics, the robot body is a highly specific dynamical system, to which, in general, existing proofs of computational universality do not apply. This implies that a mismatch can exist

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement No 248311 (AMARSi), from the European Commission (EC) Human Brain Project under grant agreement No 604102, and from the Agency for Innovation by Science and Technology in Flanders (IWT).

[1]Electric and Information Systems (ELIS), Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium `Jonas.Degrave@UGent.be`

[2]NASA Ames Research Center/Oak Ridge Associated Universities, Mail Stop 269-3, Moffett Field, California 94035, United States

between the actually observed robot states and the dynamical transformations that are required for the task. It has been shown that, when keeping a fixed number of observed states (i.e., sensor readouts), there is a trade-of between memory and non-linear computing power [20]. In this paper, we propose the introduction of an additional transformation between the physical body and the linear combination layer. We investigate the requirements to such a transformation by tuning two dimensions of its complexity: memory and nonlinearity. By doing so, we can investigate which dynamics are desired in order to outsource locomotion control for the Oncilla quadrupedal robot platform.

The remainder of this paper is structured as follows. We first describe our robot and describe the control architecture we have used. We subsequently describe the experimental setup and present and discuss the results obtained with our approach on the Oncilla robot. We end our paper by presenting our conclusions.

II. The Oncilla Robot

The quadrupedal Oncilla robot is the compliant platform used in this work, see Fig. 1. Each of the robot's legs has a three-segmented pantographic system to achieve similar dynamical to those of felines. The robot has actuated hip and shoulder joints that can perform both an abduction and a flexion motion. The knee joint is actuated with a short thread so it can only perform actuated flexion, while an opposite pushing spring does the extension. For a complete overview of this robot platform, see [6].

The robot has 12 actuated degrees of freedom and a variety of sensors. In this paper, we only use the 8 rotary encoders on the hip and knee joints of the robot. These sensors are chosen as they are found in nearly all quadrupedal robots, which will allow for a broader application of this approach. The time between consecutive updates of the sensor readings and motor actuations is on average $\Delta t \approx 8.2\,\mathrm{ms}$. However, this period can vary as much as $15\,\%$ depending on the computational complexity of the controller. The robot can operate fully autonomously, but for the sake of this paper, we power the robot with a power cable and process the signals on a remote computer.

In order to run experiments without being constrained by the space available in our lab, we put our robot on a treadmill. The robot is equipped with a distance sensor on its head, such that the treadmill can adjust speed to keep the robot in the middle of the treadmill, as shown in Fig. 2. An assistant sits next to the setup to intervene when the robot would put its own safety in jeopardy.

III. Controller Architecture

A. Embodied Computation

In earlier work [9], [17], it was shown that for a tensegrity robot, a linear transformation from the sensor signals to the motor signals was enough to generate stable locomotion. The idea behind this is that the body of the robot itself has computing power, and that this power is being harvested by using it as a reservoir. This *embodiment* of computation

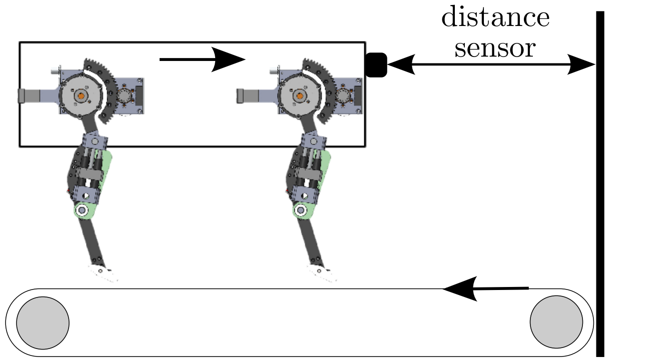

Fig. 2. The Oncilla robot on the treadmill. The distance sensor measures the distance from the robot to the front of the treadmill. The treadmill then adjusts speed to keep the robot in the center of the treadmill. This setup allows the robot to run for minutes at end while not having to deviate from a straight line.

allows the robot to generate stable locomotion without the requirement to explicitly use the state of its compliant elements in a digital control algorithm.

On our robot however, we found that we could indeed generate a gait this way, but that it was not stable and did not always return to its limit cycle. In other words, the robot's internal dynamics in response to the environment do not exactly match the required dynamics for stable gait generation. However, the fact that they suffice to generate a close but unstable approximation indicates that the mismatch is not very large.

Therefore, we propose to digitally add additional transformations to the sensor signals. In this way, the computations are still partially embodied in the morphology of the robot. In this setting, we want to quantify the minimal complexity of these transformations as expressed by their memory and nonlinear complexity. The next sections describe its parts in more detail.

B. Linear Transformation

The aim of the linear transformation is to find the $M \times N$ transformation matrix $\boldsymbol{W}$ that optimally maps the $N \times 1$ vector of the N normalized input signals $\boldsymbol{x}$ to the $M \times 1$ vector of the M output signal $\hat{\boldsymbol{y}}$:

$$\hat{\boldsymbol{y}} = \boldsymbol{W} \cdot \boldsymbol{x}.$$

Optimality is defined as the minimisation of the mean squared error (MSE) between the output signals $\hat{\boldsymbol{y}}$ and the target output signals $\boldsymbol{y}$. This can be achieved by using linear regression:

$$\boldsymbol{W} = \boldsymbol{y} \cdot \boldsymbol{x}^{+}$$

To achieve the right bias, we add a constant signal to the inputs $\boldsymbol{x}$.

This approach is limited to one-shot learning. In order to continue optimizing this relation while running, we will use the recursive least squares (RLS) algorithm [21], an online method for linear regression. In what follows, we introduce the vector $\boldsymbol{x_{rls}}$ to indicate the inputs for the RLS-algorithm, because in the remainder of this work these will be transformed versions of the sensor outputs $\boldsymbol{x}$. In RLS, the weight matrix $\boldsymbol{W_{rls}}$ is updated at each time step according to the following equations:

$$\boldsymbol{L_{rls}}(t) = \frac{\boldsymbol{P_{rls}}(t) \cdot \boldsymbol{x_{rls}}(t)}{1 + \boldsymbol{x_{rls}}^{\mathrm{T}}(t) \cdot \boldsymbol{P_{rls}}(t) \cdot \boldsymbol{x_{rls}}(t)}$$

$$\boldsymbol{P_{rls}}(t + \Delta T) = \boldsymbol{P_{rls}}(t) - \frac{\boldsymbol{P_{rls}}(t) \cdot \boldsymbol{x_{rls}}(t) \cdot \boldsymbol{x_{rls}}^{\mathrm{T}}(t) \cdot \boldsymbol{P_{rls}}(t)}{1 + \boldsymbol{x_{rls}}^{\mathrm{T}}(t) \cdot \boldsymbol{P_{rls}}(t) \cdot \boldsymbol{x_{rls}}(t)}$$

$$\boldsymbol{e_{rls}}(t) = \boldsymbol{y}(t) - \boldsymbol{W_{rls}}(t - \Delta t) \cdot \boldsymbol{x_{rls}}(t)$$

$$\boldsymbol{W_{rls}}(t) = \boldsymbol{W_{rls}}(t - \Delta t) + \boldsymbol{e_{rls}}(t) \cdot \boldsymbol{L_{rls}}^{\mathrm{T}}(t).$$

Here, Δt is the controller time step. $\boldsymbol{P_{rls}}$ is the $N \times N$ precision matrix, which is initialized with the identity matrix, such that the noise on the input signals is initially assumed uncorrelated. $\boldsymbol{e_{rls}}$ represents the $M \times 1$ a priori error vector. $\boldsymbol{W_{rls}}$ is the matrix that represents the linear transformation, which is initialized with zeros.

C. Adding Non-Linear Dynamics

Between the robot body and the linear transformation, we now add an additional transformation layer in order to increase the richness of signals received by the linear transformation. In order to be able to separately explore the need for memory and nonlinearity, we introduce two separate modules: a nonlinear layer and a memory buffer.

The nonlinear transformations are generated by introducing a hidden layer of H nonlinear neurons, each of which receives a random mixture of the sensor signals (again augmented with a constant bias signal):

$$\boldsymbol{x_{nl}}(t) = \tanh(\boldsymbol{W}_{hidden} \cdot \boldsymbol{x}(t) + \boldsymbol{w}_{bias}).$$

This technique is known as Extreme Learning Machines (ELM) [22]. In our paper, we initialize all elements in the matrix $\boldsymbol{W}_{hidden}$ and the vector $\boldsymbol{w}_{bias}$ by sampling them from the standard normal distribution. These elements are not optimized.

The memory buffer of length B is added to each nonlinearly transformed signal, such that the RLS algorithm obtains direct access to the signals from previous time steps in $\boldsymbol{x_{rls}}(t)$. $\boldsymbol{x_{rls}}$ thus contains all signal values from a small time window in the past. This allows us to explore a further richness of the dynamics which were added by the morphology of the robot:

$$\boldsymbol{x_{rls}}(t) = \left\{ \begin{matrix} \boldsymbol{x_{nl}}(t) \\ \boldsymbol{x_{nl}}(t - \Delta t) \\ \boldsymbol{x_{nl}}(t - 2\Delta t) \\ \vdots \\ \boldsymbol{x_{nl}}(t - B\Delta t) \end{matrix} \right\}.$$

The resulting detailed controller architecture is schematically represented in Figure 3.

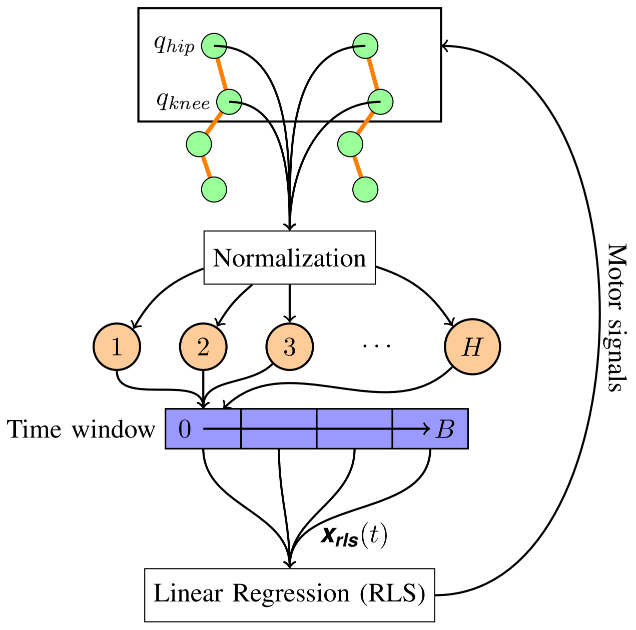

Fig. 3. A schematic of the control system. The signals measured in the motor encoders are first normalized, are then send through a layer of hidden, untrained neurons, the outputs of these neurons are buffered and a linear transformation is performed on these buffered signals to generate the motor signals. It is only this linear transformation which is learned through a linear regression method (RLS).

IV. Experimental Setup

We want our robot to realise a stable gait based on the feedback from the 8 rotary encoders. Using these sensors, the proposed controller must be trained to derive a motor command which is sent to the end-effectors. In sequential operation, this should result in the robot moving with a stable gait. The training procedure outlined in this section enables the controller to discover the relation between the received sensor signals and the output it needs to generate at that moment.

As target signals, we use motor signals for stable gaits resulting from previous work [23], [24]. We have the robot trot at a frequency of $1.7\,\mathrm{Hz}$, corresponding to a speed of about $0.76\,\mathrm{m/s}$. Before each experiment, the robot runs for 5 seconds with this gait, using the desired signal as input, in order to reach steady state. During these 5 seconds, we measure the mean and the standard deviation of the sensor signals. These are used to normalize the input signals such that they have a mean of 0 and a standard deviation of 1. After normalization, we add Gaussian noise with an amplitude of 0.01 to each input signal for regularization during training.

In the first training phase, the linear combination is optimized using RLS. As a result of this training, the control system finds a relation between the input and the output signals, but it fails to find a stable attractor. Every time the robot has a small error in the output signal, this error is reflected in the input signals of the next time step. Since the controller has never learned to handle those errors, they accumulate and destabilize the attractor.

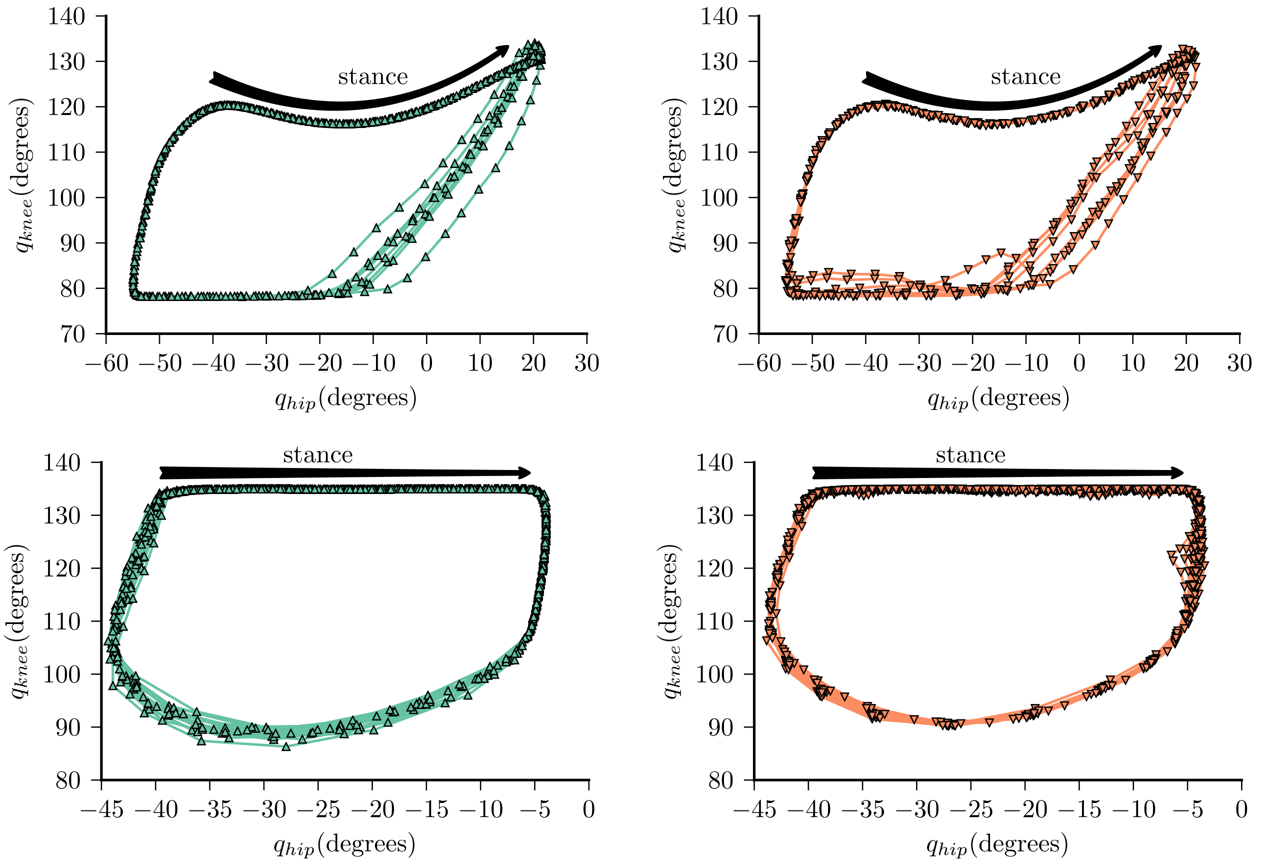

Fig. 4. On the left (▲), the trajectory of the robot's right hind leg in the joint space is shown while the robot is performing a trotting gait (top) or walking gait (bottom), with q_{hip} and q_{knee} being the angle measured over time by the hip and knee encoder respectively. These were recorded during the first 5 seconds of the first training stage. A higher angle means the leg is moved to the front or the knee is extended. On the right (▼), the same trajectory is shown, but with the learned controller, with $B = 5$ and $H = 50$ on top and $B = 12$ and $H = 50$ on the bottom. These were recorded between 25 and 30 seconds into the running stage.

For the controller to learn to deal with its own errors, we add a second training phase, with RLS still active. In this stage, the output signals sent to the motors are mixtures of the target signals and the signals generated by the linear transformation [17]. The fraction of the target signals is reduced over time until it becomes zero. After this phase, the RLS learning is switched off and the resulting gait is evaluated.

We thus split up the learning process into multiple phases:

1) **The normalization phase**: We wait for transient effects caused by starting from standstill to fade out, and when we record the average and variance of each sensor to normalize them. This stage takes 5 s.
2) **The first training phase**: We send the teacher signal to the motors, and use RLS to learn the relation between these outputs and the normalized inputs from the sensors. This stage takes 10 s, unless noted differently.
3) **The second training phase**: The motor signals are mixed between the teacher signal and the signals generated by the linear transformation. The RLS-algorithm still updates $\boldsymbol{W_{rls}}$. This stage takes 10 s, unless noted differently.
4) **The running phase**: The robot stops optimizing the linear transformation, but continues to run and where we test the stability of the attractor.

In the controller described in the previous section, there are three parameters: the number of hidden neurons H, the number of time steps in the time window B and the training time t_{train}. As the hidden neuron layer is the only nonlinear part of the controller, the number of hidden neurons H gives an indication of the amount of nonlinearity that is needed in the system. The number of time steps in the time window B is a measure for how much the system depends on time information. Finally, the training time t_{train} is a measure for the complexity of the learning task. When a stable attractor is found quickly, this suggests that we may be able reduce the number of input signals for linear regression, at the cost of a longer training time. Vice versa, if it takes long to find a stable attractor, the training time could probably be reduced by adding more input signals to the linear regression.

In order to evaluate the importance of each of these three parameters, we conducted three experiments. These were aimed at determining the interaction between these parameters and at finding their minimal values before the attractors become unstable. We are however limited by computation speed. The RLS-algorithm scales with $O(N^2)$, with N being

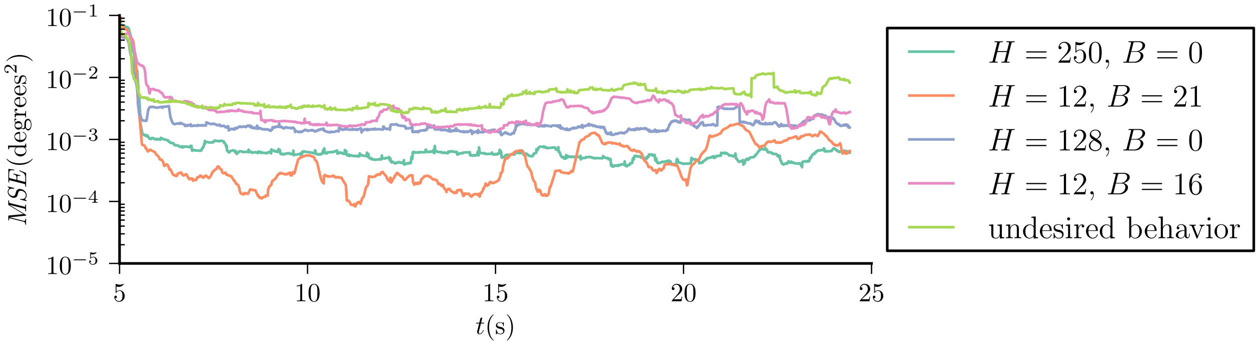

Fig. 5. Evolution of the MSE during training, in which the noise has been filtered by a moving average filter with the length of one gait period: without time window (turkoise and blue), with reduced number of hidden neurons (orange and pink) and a result in which the controller failed to find a stable attractor because it did not have enough signals (light green).

the number of signals it receives. In our case, this is equal to $(B+1)H+1$. When this number becomes too large, Δt needs to increase to allow enough time for computations, which is bad for the accuracy of the movements of the robot.

We identify three types of undesired behavior the trained controller can display:

- the output signals die out and the robot motion freezes, because the attractor has a stable, fixed point equilibrium;
- the robot moves erratically because the attractor has the wrong limit cycle or no limit cycle;
- the robot is stable while in gait, but cannot return to the limit cycle once it has left it, because the attractor has a limit cycle for which the basin of attraction is too small.

In this paper, we therefore look for the minimal values of the parameters in our control system that allow our robot to run stably for 30 seconds, and to return to its gait after the motors have been powered off and are powered back on. This way we test the controller's robustness to these three problems.

V. Results

To evaluate our approach, we optimized a controller with $H = 50$ hidden nodes and a time window B of 5 time steps ($44\,\mathrm{ms}$ or $7.5\,\%$ of the gait period). For these settings, an attractor was found that generated a stable gait and was able to return to its limit cycle after stopping. The resulting attractor is depicted in Fig. 4.

In order to prove that this result is reproducible on different setups, we trained the Oncilla robot to perform a walking gait. For this situation, we again used $H = 50$ hidden nodes, but we had to increase the time window B to 12 time steps ($103\,\mathrm{ms}$ or $17.5\,\%$ of the gait period). The resulting attractor is shown in Fig. 4. We also needed to increase the length of the first and second training stage to $30\,\mathrm{s}$.

The fact that we needed to increase both B and t_{train} is explained by the increased complexity of a walking gait. In this gait, each leg has a different phase which means that there is less dependence between the motor signals that need to be generated. Therefore, we had to increase the number of inputs to the RLS layer to the maximum we could compute in real time. Additionally, we needed to increase the training time to find the proper relation between the inputs and the outputs.

Since our approach was reproducible, we consequently tried to reduce the parameters for the easier trot gait to identify the point at which the controller fails to find a stable attractor. We first reduced the number of hidden neurons H. It makes little sense to have $H < 12$, since we have 12 independent outputs to generate. We found that with 12 hidden neurons, we needed B to be at least 16 time steps ($115\,\mathrm{ms}$ or $20\,\%$ of the gait period) for a stable gait.

Secondly, we removed the buffer ($B = 0$), and searched for the minimal number of hidden neurons required for finding a stable attractor. We found that without a time window, we need at least $H = 128 \pm 32$ hidden neurons.

Thirdly, we tried to reduce the training time. For this, we used a controller without a buffer and with $H = 250$ hidden neurons. We found that $1.18\,\mathrm{s}$, or two gait periods are enough for both the first and second training stage, or $2.36\,\mathrm{s}$ in total.

VI. Discussion

We can explain the observations in the previous section by looking at the MSE during the optimization process. At each time step, we can evaluate the difference between the output generated by the linear transformation and the output of the teacher signal. In Fig. 5 we plot the evolution of the MSE over time for different optimizations, filtered by a moving average with a time window of one gait period.

The figure shows that a lower number of signals $N = (B+1)H+1$ received by the RLS algorithm results in a higher value of the MSE during training. Moreover, when the MSE is too high during training, the system fails to find a stable attractor. This implies that we can use the MSE as a dynamic stopping criterion for training. As long as the it is too high, the controller will not be able to compensate its own errors which accumulate over time. The fact that we were able to considerably reduce the training time also supports this. The figure also shows that the MSE drops rapidly during the first seconds.

When we compare the relative importance of memory and nonlinearity, we find that both contribute to the performance of the controller, but that they are more or less interchangeable. This can possibly be explained by the nonlinearity of the body dynamics, which results in sensor signals being to some extent correlated to nonlinearly transformed versions of their own history. The setup of our research did not allow to find a significant difference in relative importance between their contribution. Since Δt can vary as much as $15\,\%$ and that $\boldsymbol{W}_{hidden}$ and $\boldsymbol{w}_{bias}$ were arbitrarily sampled from a standard normal distribution, both contributions could be further optimized. However, this would unnecessarily complicate the setup. Moreover, it is an indication that this approach is robust and we believe that it could be more broadly applicable.

We want to stress that allthough the controller might seem complex, it does not eliminate the existence of morphological computation in our setup. In the rudementary case where $H = 128, B = 0$, the linear regression might receive 129 linearly independent signals. These signals are however nonlinearly dependent, and have at most 8 statistically independent components, namely the 8 sensor signals. We have undergoing research to show more clearly that morphological computing plays an essential role in this approach.

VII. Conclusion

In this paper, we demonstrated how an embodied control system with memory-less nonlinear feedback can generate a dynamically balanced trot on a compliant quadrupedal robot. Our feedback controller, based on extreme learning machines, learns the desired relation between the input and the output signals in the time span of only a couple of strides.

We have shown that this method can be extended to other gaits, such as a walk, when increasing the training time and the model complexity. The incorporation of either additional memory or additional non-linearities contribute approximately equally to the controller performance. The parameter that mainly determines the performance is the number of signals that is fed into the the linear transformation.

As our controller was trained directly on the actual robot, we did not have to rely on a simulation model, which is often unreliable on a compliant robot. In addition, the controller optimisation was fast, happened entirely online and automatically. We believe that the proposed method can provide a useful tool for transferring knowledge from open loop to closed loop control.

References

[1] A. Albu-Schäffer, C. Ott, and G. Hirzinger, "A unified passivity-based control framework for position, torque and impedance control of flexible joint robots," *The International Journal of Robotics Research*, vol. 26, no. 1, pp. 23–39, 2007.

[2] R. Playter, M. Buehler, and M. Raibert, "Bigdog," in *Defense and Security Symposium*. International Society for Optics and Photonics, 2006, pp. 62 302O–62 302O.

[3] S. Hyon, J. G. Hale, and G. Cheng, "Full-body compliant human–humanoid interaction: balancing in the presence of unknown external forces," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 884–898, 2007.

[4] R. v. Ham, T. G. Sugar, B. Vanderborght, K. W. Hollander, and D. Lefeber, "Compliant actuator designs," *IEEE Robotics & Automation Magazine*, vol. 16, no. 3, pp. 81–94, 2009.

[5] B. Vanderborght, A. Albu-Schäffer, A. Bicchi, E. Burdet, D. G. Caldwell, R. Carloni, M. Catalano, O. Eiberger, W. Friedl, G. Ganesh, *et al.*, "Variable impedance actuators: A review," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1601–1614, 2013.

[6] A. Sproewitz, L. Kuechler, A. Tuleu, M. Ajallooeian, M. D'Haene, R. Moeckel, and A. J. Ijspeert, "Oncilla robot: a light-weight bioinspired quadruped robot for fast locomotion in rough terrain," in *Procedures of the Fifth International Symposium on Adaptive Motion on Animals and Machines*, 2011.

[7] M. Hutter, C. Gehring, M. Bloesch, M. Hoepflinger, C. D. Remy, and R. Siegwart, "Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion," in *15th International Conference on Climbing and Walking Robot-CLAWAR 2012*, no. EPFL-CONF-181042, 2012.

[8] L. U. Odhner, L. P. Jentoft, M. R. Claffee, N. Corson, Y. Tenzer, R. R. Ma, M. Buehler, R. Kohout, R. D. Howe, and A. M. Dollar, "A compliant, underactuated hand for robust manipulation," *The International Journal of Robotics Research*, vol. 33, no. 5, pp. 736–752, 2014.

[9] K. Caluwaerts, J. Despraz, A. Işçen, A. P. Sabelhaus, J. Bruce, B. Schrauwen, and V. SunSpiral, "Design and control of compliant tensegrity robots through simulation and hardware validation," *Journal of The Royal Society Interface*, vol. 11, no. 98, p. 20140520, 2014.

[10] R. Pfeifer and F. Iida, "Morphological computation: Connecting body, brain and environment," *Japanese Scientific Monthly*, vol. 58, no. 2, pp. 48–54, 2005.

[11] H. Hauser, A. J. Ijspeert, R. M. Füchslin, R. Pfeifer, and W. Maass, "Towards a theoretical foundation for morphological computation with compliant bodies," *Biological cybernetics*, vol. 105, no. 5-6, pp. 355–370, 2011.

[12] D. Verstraeten, B. Schrauwen, M. dHaene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.

[13] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, p. 34, 2001.

[14] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[15] C. Fernando and S. Sojakka, "Pattern recognition in a bucket," in *Advances in artificial life*. Springer, 2003, pp. 588–597.

[16] K. Vandoorne, P. Mechet, T. Van Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman, "Experimental demonstration of reservoir computing on a silicon photonics chip," *Nature communications*, vol. 5, 2014.

[17] K. Caluwaerts, M. D'Haene, D. Verstraeten, and B. Schrauwen, "Locomotion without a brain: physical reservoir computing in tensegrity structures," *Artificial life*, vol. 19, no. 1, pp. 35–66, 2013.

[18] J. Degrave, R. Van Cauwenbergh, F. wyffels, T. Waegeman, and B. Schrauwen, "Terrain classification for a quadruped robot," in *International Conference on Machine Learning and Applications*, 2013.

[19] Q. Zhao, K. Nakajima, H. Sumioka, H. Hauser, and R. Pfeifer, "Spine dynamics as a computational resource in spine-driven quadruped locomotion," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 1445–1451.

[20] J. Dambre, D. Verstraeten, B. Schrauwen, and S. Massar, "Information processing capacity of dynamical systems," *Scientific Reports*, vol. 2, 2012.

[21] J. M. Cioffi and T. Kailath, "Fast, recursive-least-squares transversal filters for adaptive filtering," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 32, no. 2, pp. 304–337, 1984.

[22] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[23] J. Degrave, M. Burm, T. Waegeman, F. wyffels, and B. Schrauwen, "Comparing trotting and turning strategies on the quadrupedal oncilla robot," in *IEEE International Conference on Robotics and Biomimetics (ROBIO-2013)*, 2013.

[24] J. Degrave, M. Burm, P.-J. Kindermans, J. Dambre, and F. wyffels, "Transfer learning of gaits on a quadrupedal robot," *Adaptive Behavior*, Prepublished January 20 2015.