Learning Robot Gait Stability using Neural Networks as Sensory Feedback Function for Central Pattern Generators

Sébastien Gay[1,2] José Santos-Victor[2] and Auke Ijspeert[1]

***Abstract*— In this paper we present a framework to learn a model-free feedback controller for locomotion and balance control of a compliant quadruped robot walking on rough terrain. Having designed an open-loop gait encoded in a Central Pattern Generator (CPG), we use a neural network to represent sensory feedback inside the CPG dynamics. This neural network accepts sensory inputs from a gyroscope or a camera, and its weights are learned using Particle Swarm Optimization (unsupervised learning). We show with a simulated compliant quadruped robot that our controller can perform significantly better than the open-loop one on slopes and randomized height maps.**

I. INTRODUCTION

Balance control during locomotion is critical for legged robots to move in a rough or dynamic environment. Being able to locomote next to a human wherever he/she goes while ensuring the robot and human integrity is a challenge that remains unsolved to this day. One of the precursor of this field is the MIT LegLab which has produced self stabilizing monopod, biped and quadruped robots([12], [14]). The control behind this self-stabilization was composed of a small set of simple PID-like laws. An extension of this work has been implemented in the BigDog robot from Boston Dynamics which achieved impressive results at walking outdoors in rough environments. [13] This approach however is very specific and relies strongly on powerful actuators, precise sensors and accurate state estimation.

Another well known approach in the field is the crossing of very rough terrains with the robot Little Dog, also from Boston Dynamics [10]. In [5], a model of the world was extracted from external cameras, and a footstep planner was in charge of finding an achievable path from one end of the terrain to the other. The robot then achieved that path by generating motions using inverse models. The robot was successful at crossing this kind of very rough terrain, but its movements were basically a succession of discrete movements for precise foot placement. The robot was always in a statically stable posture, and had perfect knowledge of the whole environment.

In this paper we also want to allow a four legged robot to cross a rough terrain, but we want to do so while keeping a dynamic gait. One popular line of work aims at decoupling the rhythmic motion and the feedback, by using central pattern generators (CPG), i.e. network of coupled oscillators, as an open-loop controller, which is then modified by sensory feedback. Sensory feedback can be readily integrated in CPGs. In [11], the oscillators are decoupled and feedback from force sensors actually generates the inter-limb coordination. In [7], two very basic feedback loops implementing contact feedback and phase resetting for swing stance transitions have shown to increase the robot stability on slopes.

In [6], robot stabilization was achieved on the Tekken Robot controlled with a CPG by implementing a set of reflexes inspired by biology inside the equations of the CPG. An extension of this work, with an aim at biologically plausible control was presented in [8]. The controller was composed of different kinds of neurons, some responsible for the generation of the rhythmic joint trajectories for the legs and others responsible for shaping these trajectories using sensory information. The feedback functions were also inspired from biology.

Implementing feedback in CPGs has recently been investigated on the HyQ robot ([15]). In [4] the authors use a kinematic model of the robot to adjust the foot locus according to the robot orientation, and inverse dynamics to stabilize it on challenging terrain. However, the feedback is simply superimposed to the trajectories generated by the CPG, and thus the stability properties of CPGs are not exploited.

In [2] and [3] reflexes are integrated inside a CPG enhanced using virtual model control. This time kinematic information is used to include feedback inside the CPG dynamics. The trajectories with sensory feedback corrections are directly generated by the CPG.

Our work is in a sense similar to these latest lines of work. We use a central pattern generator to generate the rhythmic motion for the legs and modify this CPG using sensory feedback so as to stabilize the robot when walking on rough terrain. The main difference between our approach and the pieces of work presented before is that we do not explicitly define the feedback functions modifying the CPG. Instead we want the robot to learn how to modify its own CPG using its sensors in order to maintain balance. Here we choose to use the gyroscope velocities and the optical flow from the camera as sensor information. As both these sensors represent speed rather than absolute orientations, they provide precious information about the robot dynamics. To enable this learning of stability we choose to represent the mapping between sensor values and perturbations applied to the radius, phase and offset of the CPG using an artificial neural network. The weights of this neural network are then optimized in simulation using Particle Swarm Optimization (PSO). The choice of CPG for the generation of rhythmic

[1]Biorobotics Laboratory, Ecole Polytechnique Fédérale de Lausanne
[2]VisLab, Institute for systems and robotics, Instituto Superior Técnico, Lisbon

motions was motivated by a crucial property: its global stability (with finite time perturbations) which ensures the smoothness of the generated trajectories. The limit cycle of the CPG that we use is globally stable, causing a trajectory modified by feedback for some amount of time to return to the limit cycle when the feedback disappears. The other main interest of CPG for this work is that it decreases the control problem dimensionality to a small set of variables. This means that the number of outputs of our neural network, so the number of parameters to optimize to learn its weights is also limited. Thus we believe that CPGs are a good basis to learn a model-free mapping from sensory information to joint trajectories. Moreover, we chose to learn gait stability here rather than explicitly writing the equations for it, so as to investigate if optimization could find different strategies to what a researcher would implement, or even what animals would do. To our knowledge, this work is the first to attempt to learn a feedback controller for robot gait stability using CPGs in such a direct way. It is also the first to link optical flow with CPGs to control walking robot balance.

The robot used for this work is called Oncilla [17]. It is a quadruped robot mimicking cat properties, with in-series compliance on each knee joint. After describing the framework used to control this robot in open-loop, we show how to include sensory feedback to modify the gait in order to prevent the robot from falling when the terrain is changing, and our learning procedure. We finish by presenting experiments with the robot in simulation on different terrains and discuss the results.

II. CONTROL FRAMEWORK

In this section, we present our control framework for learning the mapping between sensor values and central pattern generator commands. The goal here is to adapt an already existing open-loop gait to cope with changing terrains. We consider two different kinds of terrains: slopes and randomized height maps, but our control framework could be applied to any terrain in theory. Figure 5 shows these terrains in simulation. Our goal is to be able to cross these terrains by slightly modifying our existing gait, and thus keeping a dynamic rhythmic motion, rather than by performing a series of discrete movements for careful foot placement. Since obtaining this open-loop gait is not the main purpose of this paper, we will only briefly describe the methodology in Section II-A. Next we will present how to introduce a neural network as sensory feedback function for the CPG, and the learning procedure.

A. Open-Loop Central Pattern Generator

The robot is controlled by a network of coupled non linear oscillators, a central pattern generator (CPG). The unit oscillator has been modified from [16]. The general idea of this oscillator is to be able to control the duty factor of the gait - the ratio of the duration of the stance phase and the total stride duration - by applying a skewed sine wave to the protraction-retraction joint of the hips. Furthermore the shape of the foot locus can be tuned by applying a double peak trajectory to the knee joint, the duration of each peak being defined by the duty factor. The main motivation to use CPGs here is to exploit their natural properties of robustness to perturbations and smoothness, critical features when introducing sensory feedback. The abduction-adduction joint of the hips is not used for the open-loop gait, and only for discrete movements using feedback. The main difference between the oscillator used here and the one in [16] is that we use a Hopf-like convergence behavior for the amplitude, which is useful when introducing feedback. The equations of the unit oscillators used for the hip and knee are given below:

$$\dot{r}_h = \gamma(\mu_h - r_h^2)r_h \quad (1)$$
$$\dot{\phi}_h = \omega \quad (2)$$
$$\theta_h = r_h \cos(\phi_L) + o_h \quad (3)$$

where ϕ_L is a filter applied on the phase given by:

$$\phi_L = \begin{cases} \frac{\phi_{2\pi}}{2d} & \text{if } \phi_{2\pi} < 2\pi d \\ \frac{\phi_{2\pi}+2\pi(1-2d)}{2(1-d)} & \text{otherwise} \end{cases}$$

and $\phi_{2\pi} = \phi \pmod{2\pi}$

$$\dot{r}_{k1} = \gamma(\mu_{k1} - r_{k1}^2)r_{k1} \quad (4)$$
$$\dot{r}_{k2} = \gamma(\mu_{k2} - r_{k2}^2)r_{k2} \quad (5)$$
$$\theta_k = r_k\Gamma_k + o_k \quad (6)$$

with:

$$r_k = \begin{cases} r_{k1} & \text{if } \phi_{2\pi} < \pi \\ r_{k2} & \text{otherwise} \end{cases} \quad (7)$$

$$\Gamma_k = \begin{cases} -16\phi_N^3 + 12\phi_N^2 & \text{if } \phi_N < \frac{1}{2} \\ 12(\phi_N - \frac{1}{2})^3 - 12(\phi_N - \frac{1}{2})^2 + 1 & \text{otherwise} \end{cases} \quad (8)$$

$$\phi_N = 2(\frac{\phi_k}{2\pi} \pmod{0.5}) \quad (9)$$

r_h and r_k are the radiuses of the hip and knee oscillators, μ_h is the hip target amplitude, μ_{k1} and μ_{k2} the knee stance and swing amplitudes, ω their frequency, ϕ_h and ϕ_k their phases, o_h and o_k their offsets and θ_h and θ_k their outputs. γ is a positive gain defining the speed of convergence of the radiuses to the target amplitudes μ_h, μ_{k1} and μ_{k2}. d is the virtual duty factor, the actual duty factor depending on the robot dynamics and on parameters of the gait. Hip and knee are coupled so that $\phi_k = \phi_h + \psi_{hk}$, where ψ_{hk} is the desired phase shift between hip and knee. Figure 1 shows the commands sent to hip and knee for three different values of the virtual duty factor.

The four hips of the robot are also phase-coupled in order to synchronize them, to achieve different gaits. The coupling between hip oscillators i and j is obtained by adding a term to Equation 2 as follows:

$$\dot{\phi}_{hi} = \omega + w_{ij}sin(\phi_{hj} - \phi_{hi} - \psi_{ij}) \quad (10)$$

where ψ_{ij} is the desired phase difference between the oscillators controlling hips i and j and w_{ij} is a positive gain

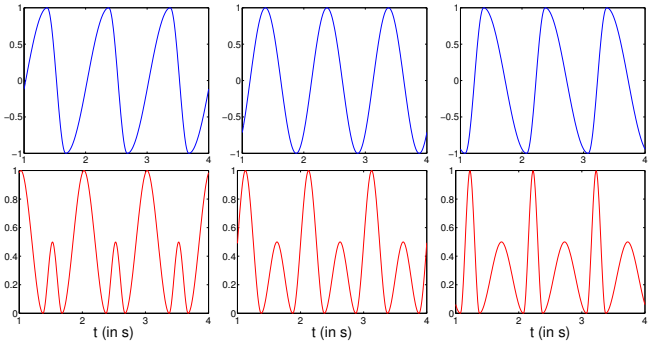


Fig. 1: The hip (top, blue) and knee (bottom, red) commands for different values of the virtual duty factor: $d = 0.3$ (left), $d = 0.5$ (middle) and $d = 0.7$ (right)

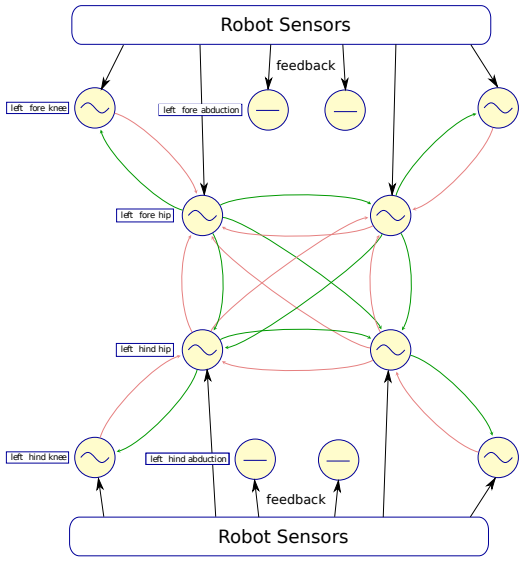


Fig. 2: The CPG used for the Oncilla robot. The hip and knee joints are controlled by the oscillators presented in Section II-A and coupled. The ablation joints are idle in the open-loop case, and only perform discrete movements controlled by the simple integrator described in Section II-B. All oscillators and integrator accept sensory feedback which modify their outputs.

defining the coupling strength. Figure 2 shows the general structure of our CPG.

The different parameters of this CPG (amplitudes, offsets, frequency, duty factor, coupling weights) have been tuned in simulation using Particle Swarm Optimization, with a fitness function aiming at minimizing the pitching and rolling angles of the robot and maximizing the speed of locomotion. These parameters have then been implemented on the real robot for validation and hand-tuned. The obtained gait has then been ported back to the simulator to carry out the work described in this paper.

B. Including Sensory Feedback in the CPG

The main point of the paper is to use the CPG presented in Section II-A and introduce sensory feedback to enable the robot to adapt to changing environments. Our controller is modular and the CPG remains fully operational if the feedback is disabled.

We modify the equations presented in Section II-A by introducing feedback on the radius, phase and offset variables of the hip oscillator and on the radius and offset of the knee oscillator (the phase being shared with the hip). The new equations for the hip and knee are given below:

$$\dot{r}_h = \gamma(\mu_h + \kappa_r \mathcal{F}_h^r(\mathbf{S}) - r_h^2) r_h \quad (11)$$

$$\dot{\phi}_h = \omega + w_{ij} sin(\phi_j - \phi_i - \psi_{ij}) + \kappa_\phi \mathcal{F}_h^\phi(\mathbf{S}) \quad (12)$$

$$\dot{o}_h = \kappa_o \mathcal{F}_h^o(\mathbf{S}) \quad (13)$$

$$\dot{r}_{k1} = \gamma(\mu_{k1} + \kappa_r \mathcal{F}_k^{r1}(\mathbf{S}) - r_{k1}^2) r_{k1} \quad (14)$$

$$\dot{r}_{k2} = \gamma(\mu_{k1} + \kappa_r \mathcal{F}_k^{r2}(\mathbf{S}) - r_{k2}^2) r_{k2} \quad (15)$$

$$\dot{o}_k = \kappa_o \mathcal{F}_k^o(\mathbf{S}) \quad (16)$$

where $\mathcal{F}_k^{r1}(\mathbf{S}) = \begin{cases} \mathcal{F}_k^r(\mathbf{S}) & \text{if } \phi_{2\pi} < \pi \\ 0 & \text{otherwise} \end{cases}$

and $\mathcal{F}_k^{r2}(\mathbf{S}) = \begin{cases} \mathcal{F}_k^r(\mathbf{S}) & \text{if } \phi_{2\pi} \geq \pi \\ 0 & \text{otherwise} \end{cases}$

The abduction/adduction joints are also used to exploit the sensory feedback and increase the robot stability. They are decoupled from the other joints and perform only discrete movements given by the following equations:

$$\dot{r}_a = \gamma(\kappa_r \mathcal{F}_a^r(\mathbf{S}) - r_a^2) r_a \quad (17)$$

$$\dot{o}_a = \kappa_o \mathcal{F}_a^o(\mathbf{S}) \quad (18)$$

$\mathcal{F}_h^r$, $\mathcal{F}_h^\phi$, $\mathcal{F}_h^o$, $\mathcal{F}_k^r$, $\mathcal{F}_k^o$, $\mathcal{F}_a^r$, $\mathcal{F}_a^o$ are functions of the sensor values $\mathbf{S}$ to be defined in Section II-C., and κ_r, κ_ϕ and κ_o are positive scaling factors. As described in Section II-A, the oscillator controlling the knee has two radiuses, r_{k1} for the stance phase and r_{k2} for the swing phase. The feedback for the knee radius is thus decoupled into $\mathcal{F}_k^{r1}(\mathbf{S})$ acting on r_{k1} only during the stance phase and $\mathcal{F}_k^{r2}(\mathbf{S})$ acting on r_{k2} only during the swing phase. Figure 3 shows the effect of these feedback functions on their respective variable and the output of the oscillator. Here the oscillator of the hip is shown.

The feedback on the radius $\mathcal{F}^r$ increases or decreases the amplitude of the oscillations temporarily. As soon as the feedback disappears the amplitude of the oscillations converges back to its default amplitude μ. This feedback can be used for instance to simulate leg retraction and extension reflexes.

The feedback on the offset $\mathcal{F}^o$ simply changes the setpoint of the oscillations, and stays encoded in the system when the feedback disappears. This feedback can be used for instance to change the posture of the robot when the slope of the ground changes.

The feedback on the phase $\mathcal{F}^\phi$ temporarily increases or decreases the oscillation frequency, by accelerating or decelerating the phase. When the feedback disappears, the apparent frequency of the oscillator goes back to its intrinsic frequency (the phase increases at the same speed as before the feedback arrived). This feedback can be used for instance to stop, or slow down temporarily the oscillations or to entrain the oscillator with an external signal. It is worth noting, as shown in the bottom right graph of Figure 3,

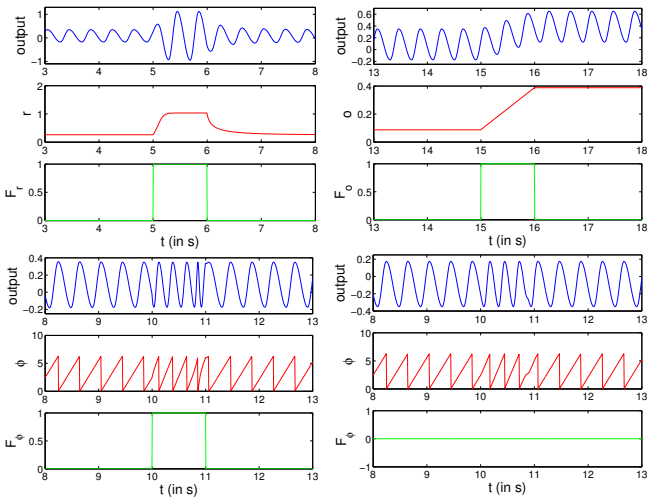


Fig. 3: Influence of the feedback functions on the output. At $t = 5$ we set $\mathcal{F}^r = 1$ until $t = 6$, at $t = 10$ we set $\mathcal{F}^\phi = 1$ until $t = 11$, at $t = 15$ we set $\mathcal{F}^o = 1$ until $t = 16$. Top Left: influence of the feedback on the radius. Top Right: influence of the feedback on the offset. Bottom left: influence of the feedback on the phase. Bottom right: influence of the feedback on the phase of one hip on another hip, through the phase coupling.

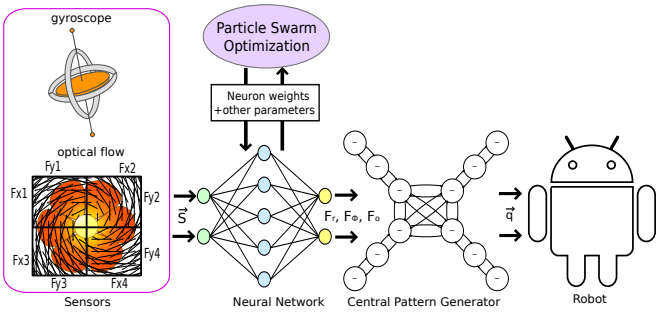


Fig. 4: General idea of our framework for learning stability. Sensor values are processed from various sensors. Optical flow is computed from the images of the camera and averaged in four quarters of the image. The sensor values are then fed to a fully connected neural network which weights are optimized using Particle Swarm Optimization (PSO). This neural network outputs the feedback values for the CPG controlling the robot

that applying feedback to the phase of one hip influences the phase of the hip of the other legs. The amount of this influence is determined by the weight of the phase coupling w_{ij}.

C. Learning the feedback functions

Now that the low level CPG has been defined the feedback functions $\mathcal{F}^r$, $\mathcal{F}^\phi$ and $\mathcal{F}^o$ need to be designed such that they map the sensor values to the right CPG modifications to stabilize the robot. One option is to design these feedback functions by hand, but this can be very complicated, especially when using multiple sensor values at once. Furthermore the feedback functions for the different legs might be strongly correlated. For instance, when standing on a platform, which starts tilting to the left, the robot might want to fold its right knees at the same time as it extends its left ones. Correlating these outputs is non trivial to do by hand. A linear mapping might not be sufficient with dynamic gaits on a compliant robot. For this work, we decided to represent these feedback functions with an artificial neural network, and unsupervised learning to tune its weights. The motivations behind are :

1) Neural networks can, with sufficient number of neurons, represent any mapping from an N-dimensional input vector to M-dimensional outputs.
2) Inputs and outputs of a neural network are, by essence, correlated.
3) Learning a non linear mapping from sensors representing velocities (gyroscope, optical flow etc.) to joint speeds (variables of the CPG), should approximate aspects of the robot dynamics which a pure kinematic model cannot.
4) By learning, the robot might find different strategies to increase its stability from what engineers would imagine, or even from what animals would do.

Figure 4 shows the full control framework including the learning process. The first step is sensor processing. We get rotational speeds from the gyroscope and images from the camera. The optical flow is then computed from the images of the camera and down-sampled by splitting the image (typically in four quarters) and averaging it in each part. The sensor vector $\mathbf{S}$ is thus composed of all or a subpart of the three gyroscope values and $2K$ values for the camera (x and y component of each vector, K being the number of parts the image is split into for the down-sampling).

Theses sensor values are used as input to a fully connected neural network with sigmoid activation, which outputs the values for $\mathcal{F}^r(\mathbf{S})$, $\mathcal{F}^\phi(\mathbf{S})$ and $\mathcal{F}^o(\mathbf{S})$ for each hip joint, and $\mathcal{F}^r(\mathbf{S})$, and $\mathcal{F}^o(\mathbf{S})$ for each knee joint and abduction joint. The robot having 4 hip joints, 4 abduction joints and 4 knee joints, the total number of outputs of the neural network is 28. The CPG takes these functions as sensory feedback and outputs joint positions for each joint. The weights of the neural network W^D are tuned by an unsupervised learning process, using Particle Swarm Optimization (PSO). Other non-convex optimization algorithms like Genetic Algorithms or Simulated Annealing could of course also be used but we chose PSO for its good convergence properties in rough fitness landscapes as considered here. Note that we cannot use supervised learning methods such as back-propagation here because we do not know the function to be learned, and thus have no training data. Together with the weights of the neural network, convergence parameters of the CPG and the slope of the sigmoid of the neurons are tuned since they also determine the influence of the feedback on the output trajectories. The total parameter vector for the optimization is: $[W^D, \gamma, w_{ij}, \kappa_r, \kappa_\phi, \kappa_o, \lambda]$, λ being the slope of the sigmoid $\frac{1}{1+e^{-\lambda x}}$

A typical optimization scenario is then to initialize a first random population of these parameters, run the simulation with the type of terrain considered, and record a measure of fitness for the optimization process. Here we use the following fitness function:

$$F = \rho \times \Theta_P \times \Theta_R \quad (19)$$

$$\Theta_P = \left(\frac{1}{1 + \frac{1}{\tau} \int_{t=0}^{\tau} |\theta_P(t)| dt} \right)^{\beta_P} \quad (20)$$

$$\Theta_R = \left(\frac{1}{1 + \frac{1}{\tau} \int_{t=0}^{\tau} |\theta_R(t)| dt} \right)^{\beta_R} \quad (21)$$

where $\theta_P(t)$ and $\theta_R(t)$ are the absolute pitch and roll angle of the robot at time t, ρ is the traveled distance and τ is the total simulation time. β_P and β_R are gains used to give less or more importance to the minimization of the angles with respect to the maximization of the traveled distance. We typically use $\beta_P = \beta_R$ in this paper.

This fitness is expressed so as to maximize the traveled distance but also minimize the integrated pitch and roll angles of the robot during the simulation.

This fitness is used by the optimization process to generate the next populations by selecting the best individuals. This process is quite heavy computationally (typically more than 100 iterations to converge, with 100 particles per iteration). However, once learned, the feedback controller only requires a simple feedforward neural network computation, and is thus much cheaper computationally than methods based on inverse models.

III. EXPERIMENTS

In this section, we present experiments we performed on two different terrains: a descending slope and a height map simulating a random rough terrain. Figure 5 shows the kinds of terrains we are considering. For all the experiments described next, the open-loop gait is the same. We use a frequency of 2.5Hz, amplitudes for the hip protraction/retraction of 15 degrees, amplitude of the knees of 0.5 and 0.05 (unitless, an amplitude of 1 meaning full flexion) for the swing and stance respectively, and a duty factor of 0.6. This gait is used both in simulation and on the real robot, with similar performance. It is a very dynamic gait, where the springs are used extensively and which reaches about 40 cm/s (1.7 body-lengths/s) both in simulation and on the real Oncilla robot. We first show the results when learning in each terrain, and analyze the strategies found out by optimization. Then we will investigate if the feedback functions learned on one terrain improve the performance of the robot on the other. All simulations are performed using a model of the Oncilla robot in Webots ([9]), a robotics simulation software based on the Open Dynamics Engine (ODE). A video showing the results of this work is available with this paper and a better quality version can be watched in [1].

A. Learning Procedure

The learning procedure is the same for each terrain. A first population of parameters are generated where all the weights of the neural network are close to 0, and thus the performed gait is very similar to the open-loop gait. This gives a first good guess to the optimization, where the robot

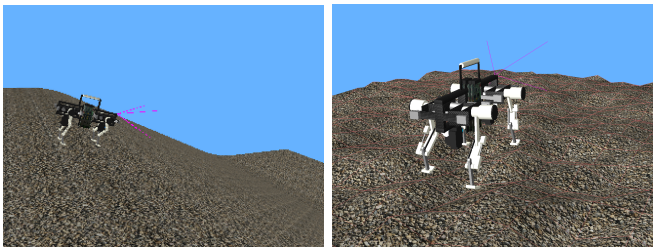


Fig. 5: The two worlds used for his work. Left: a slope. Right: height map

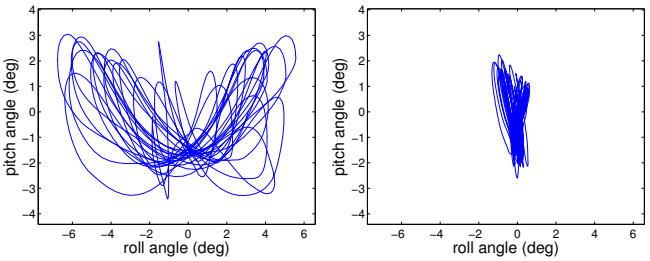


Fig. 6: Pitch vs Roll angle (in deg) over 10 seconds of simulation with the open-loop (left) and closed-loop (right) controllers on flat ground.

at least moves forward in a stable way on the flat section before the changing terrain. For the next generations, the particles explore the search space as specified by the PSO algorithm. We run the optimization for 300 iterations with 100 particles in each iteration. For each terrain we ran 2 times this optimization procedure: first using only the gyroscope as sensory input and then using only the camera. The camera sampling rate was set to 50Hz, while the gyro sampling rate was set to the control frequency: 167Hz (6ms timestep). Here want to investigate the performance using each sensor separately, but our idea for future works is to fuse the different sensory inputs, by simply adding them as input to the neural network. We repeated each optimization 3 times for each terrain and each sensor to check that our learning procedure is independent of initial conditions of the Particle Swarm Optimization. Only one of these repetitions is shown here, but qualitatively similar results (with slight differences in the strategies found) were achieved in each case.

B. Flat ground

As a first proof of concept, we want to check if our controller can decrease the pitch and roll angles of the robot walking on flat ground. We run the learning procedure described before with the fitness presented in Equation 19.

After about 150 iterations the optimization has converged and the resulting controller is able to reduce the average pitching and rolling angles of the robot by 32% and 89% respectively. Figure 6 compares the pitch and roll angles of the robot using the open-loop controller and the closed-loop controller.

C. Slope

The robot is placed in front of a slope (Figure 5, left). For each particle of the optimization, we repeat the simulation with 3 different slopes of angles 0.3, 0.4 and 0.5 radians (17, 23 and 29 degrees) . The fitness of each particle is then computed as the average of the traveled distance on

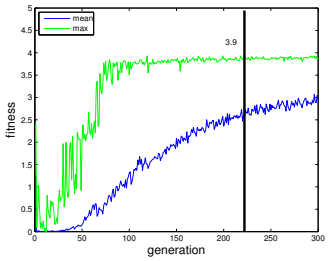


Fig. 7: Evolution of mean, maximum and minimum fitness (the traveled distance) of each iteration throughout the optimization, using gyro as sensory input.

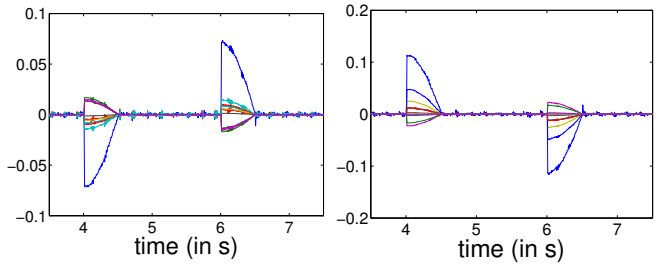
these 3 slopes, including the flat sections before and after the slope. This corresponds as setting β_P and β_R to 0 in Equation 19. We do not want to minimize the pithing and rolling angles here, as walking on a slope with a flat trunk is very unnatural and may not lead to the best performance. The idea is that, after learning, the controller should be able to generalize to any slope between these values.

The evolution of the fitness during the optimization using the gyroscope as only sensory input is given in Figure 7. The best individual reached a traveled distance of about 3.9 meters which is the maximal achievable fitness (it corresponds to all 3 slopes being crossed). This is qualitatively similar when using the camera, instead of gyro, although the optimization takes longer to converge (about 220 iterations), most likely because of the higher number of parameters (we use 8 values for the optical flow, and only 3 values for the gyro). The similar performance of the controller using optical flow and gyro is an interesting result, since the gyro provides information more than 3 times more often than the camera. Moreover, the information provided by the gyro (rotational speeds of the trunk) is much more explicit than the optical flow provided by the camera, which only gives the linear speed of the pixels. However, the optimized neural network seems to interpret the optical flow as well as the gyro rotational speeds.

Despite the relatively high number of parameters, (in this case 378), the optimization converges nicely in about 100 iterations. The average fitness keeps getting better trough the rest of the optimization and our best controller emerges after 221 iterations.

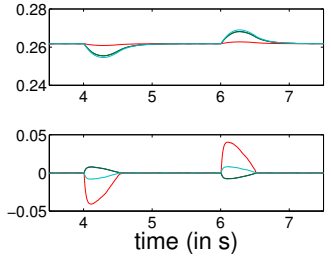
It is hard to see what kind of feedback is learned when looking at the data from the robot walking, since the feedback is basically always active. To analyze this feedback, we actuate the robot in the air. To simulate the transitions from flat ground to slope and back, at $t = 4s$ we rotate the robot around its pitch axis for $0.5s$, until it reaches an angle of 0.4 radians (23 degrees). At $t = 6$ we rotate it back to its initial position for $0.5s$. Figure 8 shows the feedback on the radius and offset when using gyro as sensory information, as well as the evolution of the radius and offset and the commands of each oscillator. The feedback learned did not make use of the feedback on the phase.

The first thing worth noting is that even though the feedback is not very smooth (due to noise, low sampling frequency of the sensor etc. - Fig. 8a and 8b), the commands

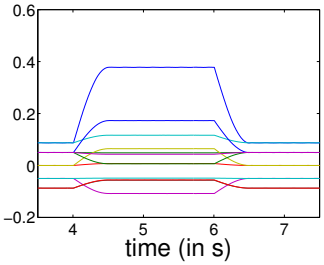


(a) feedback on the radius

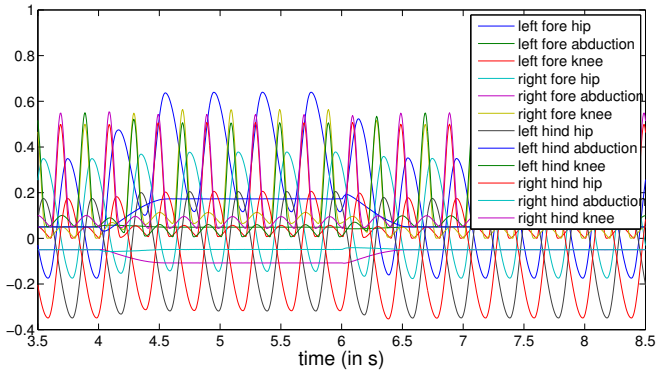
(b) feedback on the offset



(c) evolution of the radius (top: hip protraction, bottom: hip abduction)



(d) evolution of the offsets



(e) commands in radians for hips protraction and abduction, unit-less for the knee (0: knee fully extended, 1: knee fully flexed)

Fig. 8: Data when the robot is actuated in the air, using gyro as sensory input. At $t = 4s$ the robot is rotated by 0.4 radians. At $t = 6$ it is rotated back by -0.4 radians.

applied to the joints are (Fig. 8e). This shows the smoothing ability of the CPGs in the presence of perturbations (here sensory feedback). Then we notice that the robot makes good use of both the radius and offset feedback. The robot uses the feedback on the radius to temporarily adapt its leg positions and lengths to compensate the forward inertia during the transitions (Fig. 8c). Between the two transitions, a new stable gait is reached where only the offsets are different from the open-loop gait. Finally, let us observe that the new stable gait found when the robot is rotated is asymmetric (see Figure 8d). The left fore leg is placed much more forward than the other one to prevent tipping over. This is a strategy for the robot to keep balance that is different to what is usually done with model-based control. Yet, it proves effective since it allows the robot to cross a nearly 55% slope. Figure 9 shows snapshots of the robot going down a slope, where on can see this strategy in action.

To test whether our best controller can generalize to other slopes, we run 100 times the simulation with random slopes

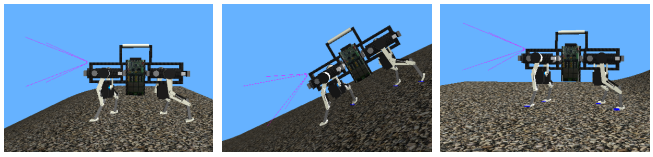


Fig. 9: Snapshots of the robot walking before the slop (left), on the slope (middle) and after the slope (right). When the robot is on the slope it puts its front left foot forward to compensate its forward inertia. After the slope, it goes back to the normal gait.

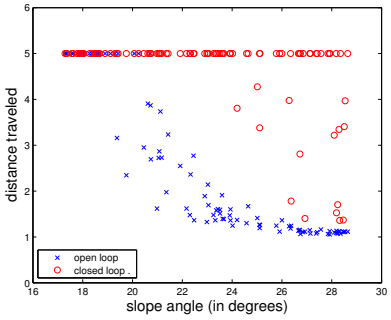


Fig. 10: Test of generalization of the best learned feedback controller. The maximum achievable distance in this world is 5m, which means any lesser traveled distance implies that the robot has fallen.

varying from 0.3 to 0.5 radians (about 17 to 29 degrees). To check that our feedback control is independent of the timing at which the robot arrives on the slope, we also set the initial position of the robot to a random value 10 cm around the position used for the training. We also use longer slopes than in the optimization phase (5 meters instead of 4). For each run, we compare the results to those of the open-loop controller. The results are shown in Figure 10. In all cases, the closed-loop controller performed equally or better than the open-loop one. The closed-loop controller was successful in crossing the whole slope in 85% of the cases, while the open-loop was successful only in 26%. Note that the open-loop controller was able to cross slopes up to 19 degrees (about 34%) in all cases, and up to 21 degrees (about 38%) in specific cases. This performance is very good for an open-loop controller, which shows that compliance and finely tuned gait parameters can lead to good open-loop performance on rough terrain. However, the closed-loop controller was able to cross significantly steeper slopes, up to 28 degrees (55%).

D. Height Map

We used the same optimization procedure as for the flat and slope grounds on a random height map, with the fitness described in Equation 19. We repeated the simulation for each particles with 3 different randomized height maps, of respective maximum height of 4, 5, and 6 cm, in a grid of 10cm steps. This corresponds respectively to about 22%, 28% and 33% of the leg length of the robot, and respectively 40%, 50% and 60% maximum local slope.

Figure 11 (left) shows the evolution of the fitness throughout the optimization. As for the slope, the optimization converges in about 100 iterations. At iteration 80, the optimization has already found a controller able to cross all three worlds. Even after the distance traveled has been maximized,

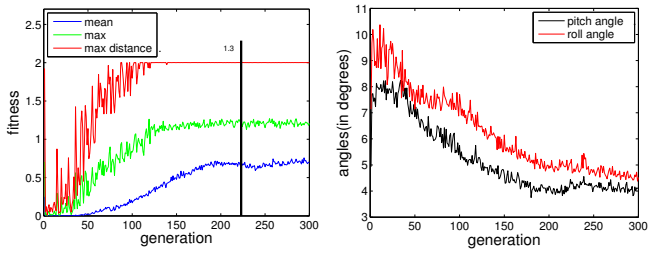


Fig. 11: Evolution of mean and maximum fitness (left) and the mean integrated pitch and roll angles (right) of each iteration throughout the optimization .

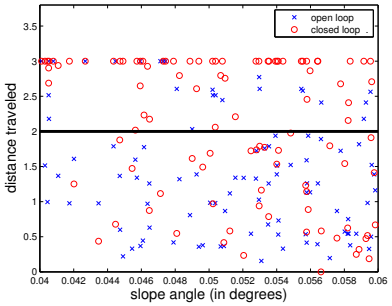


Fig. 12: Test of generalization of the best learned feedback controller. The maximum success distance is set to 2 m.

the fitness is further improved by decreasing the pitch and roll angles of the robot, as shown in Figure 11 (right).

We ran 100 generalization tests with random height maps of maximum heights ranging from 0.04 to 0.06. Figure 12 show the results. When setting the success distance to 2 meters, as in the learning, the success rate of the closed-loop controller was 57% against 25% for the open-loop controller. The closed-loop controller was better in 74% of the cases with an average traveled distance increase of 41%. When increasing the success distance, the absolute success rates of both the open-loop and the closed-loop controller decrease. However, the performance of the open-loop controller drops faster than the closed-loop one. With a success distance of 2.6m, the performance of the closed-loop and open-loop controllers are 49% and 13% respectively, and 40% and 9% for 2.9m.

Thus our controller improves significantly the performance of the open-loop controller on height-maps, but is less efficient than on slopes. This can be explained by the fact that the randomized height map contains much more stochasticity than the slope. The large difference between the training and testing set performances implies that our learning process has overfitted to the three training worlds. Thus, repeating the learning in only 3 different worlds might not be enough and inscreasing the number of training worlds should increase the generalization performance.

E. Hardware implementation

We started implementing this work on the real Oncilla robot. For now only the open-loop gait has been implemented. The robot reaches a speed of about 40 cm/s (1.7 body-lengths/s), which is comparable to what is obtained in simulation. Faster gaits have also been tested reaching speeds

of up to 55 cm/s (2.3 body-lengths/s) at 3.5Hz (See movie [1]).

IV. CONCLUSION

In this paper we have presented a framework to learn gait stability using a neural network as sensory feedback for central pattern generators. We have shown that by carefully integrating feedback in the CPG, we are able to make full use of its stability properties to generate smooth gait modifications to achieve the considered task. By learning the weights of the neural network, together with the convergence factors of the CPG in an unsupervised matter, we showed that the designed sensory feedback can significantly improve the locomotion performance (i.e. better balance, fewer falls) of the robot on rough terrain. We showed that our feedback controller can be used seamlessly with different sensory information, such as the rotational speeds provided by the gyro, but also less explicit information like the optical flow. Our controller does not need any model of the robot, whether kinematic of dynamic, but directly learns the mapping between sensors and gait corrections. Thus it can be in theory applied to any robot, even robots where obtaining a model is very hard or even impossible like deformable or tensegrity robots. We showed good performance of the controller with a simulated compliant quadruped robot on a slope, both on the training test and the testing set. To keep balance on the slope, the robot developed an original strategy consisting in putting one leg more forward than the others, very much unlike what is done in traditional control. The closed-loop controller always performed better than the open-loop gait and reached 85% success on random slopes up to 55%, against 26% for the open-loop gait. Our controller was not as efficient on the randomized height map than the slope, reaching 57% success against 25% for the open-loop controller. It still performed better in 74% on the cases than the open-loop controller. This relatively lower performance can be explained by the fact that the randomized height map contains significantly more stochasticity than the slope. Our learning process, even when repeating it in three different worlds, may have overfitted. One solution to improve the performance in the height map scenario could be to repeat the learning in more different worlds. One of the strength of our approach is the simplicity of fusing information from different sensors and thus we are currently investigating how fusing more sensors can further improve the performance. For instance, including information from force sensors on the feet of the robot could greatly help the stabilization in the randomized height map scenario, since the controller would be able to detect missing contacts and learn leg extension and stumbling reflexes. Having phase-dependent feedback could also improve the controller performance, and can be seamlessly implemented by including the phase of one or more oscillators of the CPG as input to the neural network. Finally we are starting to implement our controller on the real Oncilla robot, and comparing it to the model-based feedback controller presented in [3].

ACKNOWLEDGMENT

This work was funded by the Portuguese Fundation for Science and Technology (FCT) through the IST-EPFL joint initiative, EPFL's Biorobotics Laboratory, and the European Commission grant AMARSI FP7-ICT-248311. The authors would like to thank Mostafa Ajallooeian and Alexandre Tuleu for their help in conducting this work.

REFERENCES

[1] Video: Learning robot gait stability. . `http://www.youtube.com/watch?v=ef_Z6RUc_6g`.

[2] M. Ajallooeian, S. Pouya, A. Sproewitz, and A.J. Ijspeert. Central pattern generators augmented with virtual model control for quadruped rough terrain locomotion. In *IEEE International Conference on Robotics and Automation (ICRA 2013)*, 2013.

[3] Mostafa Ajallooeian, Sébastien Gay, Alexandre Tuleu, Alexander Spr, and Auke J Ijspeert. Modular Control of Limit Cycle Locomotion over Unperceived Rough Terrain. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[4] V. Barasuol, J. Buchli, C. Semini, M. Frigerio, E.R. De Pieri, and D.G. Caldwell. A Reactive Controller Framework for Quadrupedal Locomotion on Challenging Terrain. *Accepted for IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[5] Jonas Buchli, Mrinal Kalakrishnan, Michael Mistry, Peter Pastor, and Stefan Schaal. Compliant quadruped locomotion over rough terrain. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 814–820, October 2009.

[6] Avis H Cohen. Walking of a Quadruped Robot on Natural Ground Based on Biological Concepts. *The International Journal of Robotics Research*, 2010.

[7] Sarah Degallier, Ludovic Righetti, Sebastien Gay, and Auke Ijspeert. Toward simple control for complex, autonomous robotic applications: combining discrete and rhythmic motor primitives. *Autonomous Robots*, 31:155–181, 2011.

[8] Christophe Maufroy, Hiroshi Kimura, and Kunikatsu Takase. Towards a general neural controller for quadrupedal locomotion. *Neural networks : the official journal of the International Neural Network Society*, 21(4):667–81, May 2008.

[9] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.

[10] M. P. Murphy, a. Saunders, C. Moreira, a. a. Rizzi, and M. Raibert. The LittleDog robot. *The International Journal of Robotics Research*, 30(2):145–149, December 2010.

[11] Dai Owaki, Takeshi Kano, Ko Nagasawa, Atsushi Tero, and Akio Ishiguro. Simple robot suggests physical interlimb communication is essential for quadruped walking. *Journal of the Royal Society, Interface / the Royal Society*, (October), October 2012.

[12] M. Raibert, M. Chepponis, and H. Brown. Running on four legs as though they were one. *Robotics and Automation, IEEE Journal of*, 2(2):70–82, 1986.

[13] Marc Raibert and Kevin Blankespoor. Bigdog, the roughterrain quadruped robot. *Proceedings of the 17th World Congress, The International Federation of Automatic Control*, pages 6–9, 2008.

[14] M.H. Raibert and J.K. Hodgins. Legged robots. In R.D. Beer, R.E. Ritzmann, and T.M. McKenna, editors, *Biological Neural Networks in Invertebrate Neuroethology and Robotics*, pages 319–354. Academic Press, 1993.

[15] C. Semini, N.G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D.G. Caldwell. Design of hyq a hydraulically and electrically actuated quadruped robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 225(6):831–849, 2011.

[16] A. Sproewitz, A. Tuleu, M. Vespignani, M. Ajallooeian, E. Badri, and A.J. Ijspeert. Towards dynamic trot gait locomotion – design, control, and experiments with a compliant quadruped robot. *Interational Journal of Robotics Research (IJRR)*, 2012.

[17] Alexander Sproewitz, Lorenz Kuechler, Alexandre Tuleu, Mostafa Ajallooeian, Michiel D'Haene, Rico Moeckel, and Auke Jan Ijspeert. Oncilla robot: a light-weight bio-inspired quadruped robot for fast locomotion in rough terrain. In *Adaptive Motion in Animals and Machines, 5th International symposium, Abstracts*, 2011.