Proceedings Paper:
Xin, S, Delhaisse, B, You, Y et al. (2018) Neural-Network-Controlled Spring Mass Template for Humanoid Running. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2018 IEEE/RSJ (IROS), 01-05 Oct 2018, Madrid, Spain. IEEE, pp. 1725-1731. ISBN: 978-1-5386-8094-0. ISSN: 2153-0866. EISSN: 2153-0866.

https://doi.org/10.1109/IROS.2018.8593403

Neural-Network-Controlled Spring Mass Template for Humanoid Running

Songyan Xin, Brian Delhaisse, Yangwei You, Chengxu Zhou,
Mohammad Shahbazi, Nikos Tsagarakis

***Abstract*— To generate dynamic motions such as hopping and running on legged robots, model-based approaches are usually used to embed the well studied spring-loaded inverted pendulum (SLIP) model into the whole-body robot. In producing controlled SLIP-like behaviors, existing methods either suffer from online incompatibility or resort to classical interpolations based on lookup tables. Alternatively, this paper presents the application of a data-driven approach which obviates the need for solving the inverse of the running return map online. Specifically, a deep neural network is trained offline with a large amount of simulation data based on the SLIP model to learn its dynamics. The trained network is applied online to generate reference foot placements for the humanoid robot. The references are then mapped to the whole-body model through a QP-based inverse dynamics controller. Simulation experiments on the WALK-MAN robot are conducted to evaluate the effectiveness of the proposed approach in generating bio-inspired and robust running motions.**

I. Introduction

The Spring-Loaded Inverted Pendulum (SLIP) model is a well recognized template model [1] for hopping and running based on the biomechanical studies [2], [3]. Despite its simplicity, it accurately describes the CoM dynamics, ground reaction force profiles, and transitioning between different phases of motion observed in humans [4]. Thanks to its reductive and platform-independent model, it has been widely used in the design and control of legged robots [5]–[8]. In particular, the controlled SLIP is used as a planner in the high-level control structures of robot to provide it with references that are implicitly consistent with the natural dynamics of running.

The SLIP running is a dynamic gait rendering cyclic stability, which requires a sufficiently large prediction horizon for control. Early studies in this regard are largely influenced by the simple intuitive control implemented on Raibert's hoppers [9]. The machines were able to exhibit dynamic behaviors while it was assumed that the control of hopping height, speed and posture are decomposed. Although inspiring, all those robots share similar design of light prismatic legs, i.e., a SLIP-like morphology. When it comes to controlling legged robots with non SLIP-like morphologies like a humanoid, such an intuitive approach inspired by biology alone shows limited success. Moreover, the aforementioned decoupling leads to long convergence time due to the simplistic model used for control.

A large body of research in the SLIP literature has been directed towards more accurate and realistic controls, most of which may be categorized into two schemes: the methods which implement dead-beat like controllers through solving the running return maps [10]–[13]; and tabular control methods relying on look-up tables constructed upon the data generated by comprehensive forward-in-time simulations covering a wide range of SLIP states and parameters [14]–[16]. Application of the former to online control is not preferred, due to the non-linear optimization inevitably involved in the computations. The latter is fast enough for online implementation since a look-up table can be constructed offline. However, it is practical only for the range of parameters using which the look-up table is constructed. Moreover, the size of the table grows exponentially with the number of the input variables, which challenges the generality of the approach.

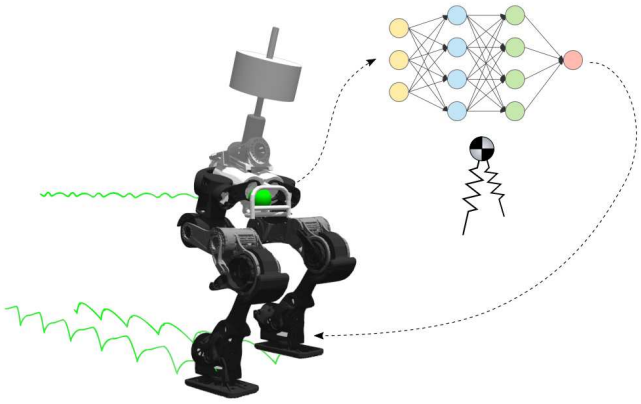

Fig. 1. WALK-MAN Lower body running in simulation. The Center of Mass dynamics of the robot is controlled to match that of a SLIP model. A neural network trained offline with a large amount of SLIP simulation data is used to encode the foot placement behaviour.

The present work strives to fill the gap between the non-linear optimization method and the classical look-up table method by using a deep neural network. The network is trained offline with large amount of simulation data based on the SLIP model to learn its dynamics. Once this most time-consuming part has been done, the trained network could be easily deployed online for real-time querying. The knowledge learned from simulation data are encoded in a limited number of weight parameters and this parametric representation does not enlarge with inputs and outputs. Comparing to the look-up table approach, the interpolation between data are naturally embedded inside the network.

Having developed an effective data-driven controller for the SLIP planner, we consider the embedding of the template behaviors into the whole-body robot. At this stage we need to focus also on the abstracted out dynamics such as the

Department of Advanced Robotics, Istituto Italiano di Tecnologia, via Morego, 30, 16163 Genova, Italy.
Email: name.surname@iit.it

touchdown impact and torso stabilization. The former is known to be a real challenge in control of legged robots in general and dynamic gaits in particular, as the impact phenomenon is a fuzzy phase of motion. In order to make sure that the planned template behaviors, which are intentionally constrained to be energy conservative, remain feasible for the real robot, we adopt an energy regulation technique that determines the takeoff moment as the moment at which the energy lost due to the touchdown impact has been restored. Application of this takeoff event condition on top of the previously proposed leg length modulation [17] remarkably improves the robustness against uncertainties introduced by the touchdown impact and other unmodeled dynamics in the planning phase. This takeoff event modulation together with the CoM reference trajectory and the touchdown angle tracking, all planned by the controlled SLIP, are mapped into the whole-body robot through a state-of-the-art QP-based inverse dynamics controller.

The paper is arranged as follows. In Section II-A, the SLIP model and its dynamics in stance and flight phase are described. Section II-B gives details about how the training data are generated and how it is used to train the neural network. Section III-A deals with the mapping issue from template model to whole-body model. Section III-B presents the formulation of whole-body controller. Simulation results are presented in Section IV and conclusions are given in Section V.

II. Deep Neural Network for SLIP-like Motion Embedding

A. Spring Loaded Inverted Pendulum Model

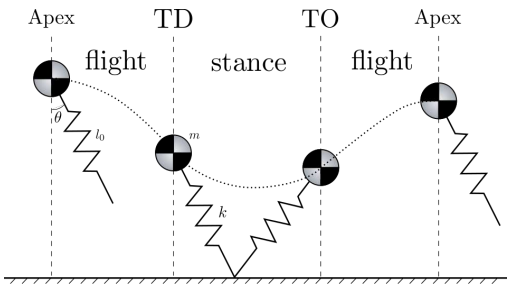

Fig. 2. The spring-loaded inverted pendulum (SLIP) model. The figure shows the sequence of events (Apex-TD-TO-Apex) and phases (flight-stance-flight) involved in one step of running. The leg angle at TD moment (θ_{TD}) decides the evolution of stance and ascending flight phases.

The spring-loaded inverted pendulum (SLIP) model consists of a point mass m and a massless spring with stiffness k and rest length l_0 as shown in Figure 2. Three phases (flight-stance-flight) are involved in one step of the running motion and they are separated by touchdown (TD) and takeoff (TO) events. During flight phase, the mass follows a ballistic projectile trajectory with the dynamics:

$$\begin{cases} \ddot{x} = 0 \\ \ddot{z} = -g \end{cases} \tag{1}$$

where (x, z) are the coordinates of the point mass in sagittal plane, and g is the gravitational acceleration. The massless leg can be arbitrarily positioned during flight phase in preparation for the touchdown. At the touchdown moment, the system switches to stance phase, the point mass follows the dynamics:

$$\begin{cases} m\ddot{x} = k[l_0(x^2+z^2)^{-1/2} - 1]x \\ m\ddot{z} = k[l_0(x^2+z^2)^{-1/2} - 1]z - mg \end{cases} \tag{2}$$

Once the leg length $l = \sqrt{(x^2+z^2)}$ reaches its rest length l_0 during spring extension, the system takes off and enters the flight phase again.

The apex point ($\dot{z} = 0$) during flight phase is usually chosen to study the periodic motion of system. At the apex point, the system state can be described by one variable $\dot{x}$ (or z) due to the total energy conservation:

$$\frac{1}{2}m\dot{x}^2 + mgz \equiv E \tag{3}$$

where E is the total energy of the system which is conserved throughout the whole process. Here the velocity $\dot{x}$ at apex point is chosen since we are more interested in regulating the running speed. Given the speed at one apex point, the system behavior in the ensuing stance and flight phases is fully determined by the touchdown angle θ_{TD}. The next apex state is a function of current apex state and the touchdown angle:

$$\dot{x}_{n+1} = f(\dot{x}_n, \theta_{TD,n}) \tag{4}$$

where n denotes the current running step. A one step deadbeat controller emerges by inverting this apex return map:

$$\theta^*_{TD,n} = f^{-1}(\dot{x}_n, \dot{x}^*_{n+1}) \tag{5}$$

where $\theta^*_{TD,n}$ is the touchdown angle that ensures reaching the desired velocity $\dot{x}^*_{n+1}$ at the next apex. However, the hybrid nature of the return map and nonlinearity of stance phase dynamics (2) exclude the possibility of finding a closed form solution for this inverse relationship. As such, the problem of finding $\theta^*_{TD,n}$ is inevitably transformed into a nonlinear optimization problem:

$$\begin{aligned} \theta^*_{TD,n} &= \underset{\theta}{\operatorname{argmin}} \, |\dot{x}^*_{n+1} - f(\dot{x}_n, \theta)| \\ s.t. \quad & \theta_{min} < \theta < \theta_{max} \end{aligned} \tag{6}$$

where θ is the touchdown angle to be optimized to bring the system state at next apex $f(\dot{x}_n, \theta)$ as close as possible to the desired one $\dot{x}^*_{n+1}$ respecting the angle limits. Usually, this time-consuming optimization process can only be conducted offline, while convergence cannot be guaranteed. These limitations motivate us to explore a different possibility which better suits the online implementation requirement, that is a neural-network-based representation for the inverse mapping (5).

B. Deep Neural Network Controller

The proposed neural network takes the inputs $[\dot{x}_n, \dot{x}^*_{n+1}]$ and outputs the touchdown angle $\theta^*_{TD,n}$. A deep learning techniques is adopted to train the network offline. The trained network is then applied online which produces an output for

every possible inputs. Below, we first describe how valid datasets are generated for training the network and then present the structure of the neural network and the training process in detail.

1) Data Generation: The neural network under consideration learns from datasets that comes from apex-to-apex simulations of the SLIP model as given in (4). Each simulation produces one dataset. For simplicity, we choose a constant energy level and all simulations are performed with this energy level E_{cons}. Given a fixed energy level, the initial state can be completely determined by an initial horizontal velocity $\dot{x}_0$. Together with a touchdown angle θ_0, we can simulate forward the SLIP model to get the next apex velocity $\dot{x}_1 = f(\dot{x}_0, \theta_0)$. At this point, a training example has been generated. A general representation of this process is:

$$\dot{x}_1^{(i)} = f(\dot{x}_0^{(i)}, \theta_0^{(i)}) \tag{7}$$

from which a training example $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ is collected as:

$$\begin{cases} \boldsymbol{x}^{(i)} = [\dot{x}_0^{(i)}, \dot{x}_1^{(i)}]^T \\ \boldsymbol{y}^{(i)} = [\theta_0^{(i)}] \end{cases} \tag{8}$$

where $i = 1, 2, ..., n$. Repeating this process with a different initial velocity and touchdown angle, the whole data set can be collected:

$$\boldsymbol{X} = \begin{bmatrix} -(\boldsymbol{x}^{(1)})^T- \\ -(\boldsymbol{x}^{(2)})^T- \\ \cdots \\ -(\boldsymbol{x}^{(n)})^T- \end{bmatrix} \qquad \boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}^{(1)} \\ \boldsymbol{y}^{(2)} \\ \cdots \\ \boldsymbol{y}^{(n)} \end{bmatrix} \tag{9}$$

One thing worth mentioning is that the initial velocity $\dot{x}_0^{(i)}$ and touchdown angle $\theta_0^{(i)}$ can be chosen randomly but should be within reasonable limits. Specifically, touchdown angle limits are defined as: $\theta \in [0, \tan^{-1}(\mu)]$ where μ is the static friction coefficient. The velocity is limited in the range $[0, \sqrt{2(E_{cons} - mgl_0)/m}]$ where the upper bound is defined with respect to the minimum height (rest length l_0), the slip model can take. Below is the pseudocode used to generate the training data sets. To be concise, we have not presented the guard functions inside the forward simulation we used to eliminate those bad data examples such as certain combination of $\dot{x}_0^{(i)}$ and $\theta_0^{(i)}$ which leads to negative vertical velocity at TO moment. In this paper, the parameters used for training is $E_{cons} = 750, m = 85, l_0 = 0.8, k = 42500$. A training set of size 30000 is collected and 5% of it has been used as test set.

Algorithm 1: Generating the training data sets

$\boldsymbol{X}, \boldsymbol{Y}$ = [], [];
for *simulation (i=0, i<n, i++)* **do**
  $\dot{x}_0^{(i)} = rand(0, \sqrt{2(E_{cons} - mgl_0)/m})$;
  $\theta_0^{(i)} = rand(0, \tan^{-1}(\mu))$;
  $\dot{x}_1^{(i)} = f(\dot{x}_0^{(i)}, \theta_0^{(i)})$;
  $\boldsymbol{x}^{(i)} = [\dot{x}_0^{(i)}, \dot{x}_1^{(i)}]^T$;
  $\boldsymbol{y}^{(i)} = [\theta_0^{(i)}]$;
  $\boldsymbol{X}$.insert($\boldsymbol{x}^{(i)}$) ;
  $\boldsymbol{Y}$.insert($\boldsymbol{y}^{(i)}$) ;
end

2) Neural Network Structure and Training: To learn the generated data, a fully-connected feed-forward network (FNN) has been used. Compared to tabular approaches [14] which check the entry closest to a given input and produce the associated output, FNN allows to generalize to different inputs. In addition, it can model non-linear functions by using non-linear activation functions, improving over the previous linear methods [18]. For a given input $\boldsymbol{x}^{(i)}$, we can define the FNN in a recursive way as follows:

$$\begin{gathered} \boldsymbol{h}_l = f_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l), \quad \forall l \in \{1, \cdots, L\} \\ \text{with} \quad \boldsymbol{h}_0 = \boldsymbol{x}^{(i)} \quad \text{and} \quad \boldsymbol{h}_L = \hat{\boldsymbol{y}}^{(i)}, \end{gathered} \tag{10}$$

where L is the total number of layers, f_l is the activation function applied on the corresponding layer l, $\boldsymbol{W}_l$ are the weight matrices, $\boldsymbol{b}_l$ are the bias terms, and $\hat{\boldsymbol{y}}^{(i)}$ is the predicted output. We can summarize the above equation by $\hat{\boldsymbol{y}}^{(i)} = f_{NN}(\boldsymbol{x}^{(i)}; \boldsymbol{W})$ where $\boldsymbol{W} = \{\boldsymbol{W}_1, \boldsymbol{b}_1, \cdots, \boldsymbol{W}_L, \boldsymbol{b}_L\}$ are the weights that need to be optimized. In our experiments, our network has 3 hidden layers with 20, 50, and 20 units respectively. We used 'relu' as the non-linear activation function for each hidden layer. To avoid overfitting, we regularize our network using dropout. The training was carried out using the mean-squared loss:

$$\mathcal{L}_{MSE} = \sum_{i=1}^{N} ||\boldsymbol{y}^{(i)} - f_{NN}(\boldsymbol{x}^{(i)}; \boldsymbol{W})||^2, \tag{11}$$

along with the Adam optimizer. We trained the network for 100 epochs using a batch size of 128, and a learning rate of 0.0001.

III. Mapping SLIP-like Motions to Whole-body Robot

We now describe how the planned template behaviors are encoded into the whole-body robot. The humanoid robot used for the simulations is the WALK-MAN [19]. In simulation we only use the lower-body of the robot for simplicity but with the idea in mind that the upper body could improve the performance by fully utilizing the swing motion of arms [12].

The Equations of Motion of the robot in the standard form are given as:

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{S}_\tau^T \boldsymbol{\tau} + \boldsymbol{J}_{\text{c}}^T(\boldsymbol{q})\boldsymbol{\lambda} \tag{12}$$

where $\boldsymbol{H}(\boldsymbol{q})$ is the mass matrix, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ is the velocity terms and $\boldsymbol{G}(\boldsymbol{q})$ is the gravitational forces. $\boldsymbol{\lambda}$ symbolizes the ground reaction forces (GRFs) and $\boldsymbol{J}_{\text{c}}$ is the corresponding contact Jacobian. $\boldsymbol{q} = [\boldsymbol{q}_f^T, \boldsymbol{q}_a^T]^T$ represents the generalized coordinates which include the 6 DoF floating-base coordinates $\boldsymbol{q}_f$ and actuated body joint coordinates $\boldsymbol{q}_a$, $\boldsymbol{\tau}$ is joint torques and $\boldsymbol{S}_\tau = [\boldsymbol{0}_{n_a \times 6}, \boldsymbol{I}_{n_a}]$ is a selection matrix for the actuated joints.

A. SLIP model to Whole-body Model

The fundamental difference between the template and real robot models is the inertia distribution. Different from the SLIP model which can arbitrarily position its mass-less leg during the flight phase, the swinging movement of heavy legs of our humanoid robot would cause noticeable change in the torso orientation. One needs to carefully consider this in designing the swing leg movement, otherwise the robot can shortly lose the balance. We avoid this problem by limiting the movement speed of swing leg at a cost of velocity regulation speed.

Apart from that, the effect of touchdown impact for the robot with heavy legs is severe, and it results in considerable energy loss in each step. Since our SLIP model simulation is conducted at a certain energy level (3), it is critical to maintain the same energy level for the robot to ensure relevant mapping. A number of strategies are proposed to compensate the energy loss. In [17], an energy correction law is proposed to inject energy with pre-compressed spring leg before touchdown based on a pre-computed value of the energy loss assuming an ideal inelastic impact.

$$\begin{aligned}\Delta E_{loss} &= \frac{1}{2}|\dot{\boldsymbol{q}}^{+T}\boldsymbol{H}\dot{\boldsymbol{q}}^{+} - \dot{\boldsymbol{q}}^{-T}\boldsymbol{H}\dot{\boldsymbol{q}}^{-}| \\ &= \frac{1}{2}\dot{\boldsymbol{q}}^{-T}\boldsymbol{J}_c^T(\boldsymbol{J}_c\boldsymbol{H}^{-1}\boldsymbol{J}^{T_c})^{-1}\boldsymbol{J}_c\dot{\boldsymbol{q}}^{-}\end{aligned} \tag{13}$$

The resting length of the spring at touchdown is changed to:

$$l_0^{+} = l_0^{-} + \Delta l \tag{14}$$

where $\Delta l = \sqrt{2\Delta E_{loss}/k}$. In practice, we find it difficult to compute an accurate estimation of the energy loss due to the touchdown timing inaccuracy and impact model mismatch. The impact is usually modelled as an elastic contact lasting for a period grater than an instant. As such, techniques relying merely on feed-forward calculations such as the one presented above may not guarantee an appropriate energy regulation. To tackle this issue, we propose a touchdown energy boost up and takeoff energy cut-off action pair. At TD instant, extra boost up energy ΔE_{boost} has been added so as to make sure that the system will achieve higher energy level when the leg extends to reach

$$l_0^{+} = l_0^{-} + \Delta l + \Delta l_{boost} \tag{15}$$

where $\Delta l_{boost} = \sqrt{2\Delta E_{boost}/k}$ is the extra extension due to boost energy. However, for the TO detection, instead of checking leg length, energy checking will be used to decide the TO moment:

$$TO : E = \frac{1}{2}mv^2 + mgz > E_{cons} \tag{16}$$

where E_{cons} is the desired energy level, beyond this level, the robot switches to flight phase immediately. Applying this method on the humanoid robot, we are able to bring the system energy to desired level in one step.

B. Whole-body torque controller

Whole-body dynamics controller receives CoM position, torso orientation, feet poses and contact state (no support, left foot support, right foot support, double support) as inputs and generate joint torques as outputs. Inside the controller, a quadratic programming (QP) problem is formulated [20] [21] [22] [23] [24]. The optimization variable $\boldsymbol{x} = [\ddot{\boldsymbol{q}}^T, \boldsymbol{\lambda}^T]^T$ combines generalized acceleration and ground reaction forces. Joint torques $\boldsymbol{\tau}$ can be easily calculated with (12). The cost function is a weighted combination of multiple tasks:

$$\min_{\boldsymbol{x}} \sum_{i=1}^{n} \omega_i ||\boldsymbol{Q}_i\boldsymbol{x} - \boldsymbol{c}_i||^2 \tag{17}$$

Each task is defined by the corresponding $\boldsymbol{Q}_i$ matrix and $\boldsymbol{c}_i$ vector and corresponding weights ω_i. During the optimization process, several constraints are considered:

$$\boldsymbol{H}_b(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}_b(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}_b(\boldsymbol{q}) = \boldsymbol{J}_{cb}^T(\boldsymbol{q})\boldsymbol{\lambda}, \tag{18}$$

$$\boldsymbol{J}_c\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}} = 0, \tag{19}$$

$$\boldsymbol{\tau} \in [\boldsymbol{\tau}_{\min}, \boldsymbol{\tau}_{\max}], \tag{20}$$

where subscript b in (18) stands for the 6 DoF of floating base and this constraints ensure the dynamic feasibility. (19) makes sure there is no slip in contact points, (20) reinforces joint torque limits. For each contact point, the contact wrench is defined as $\boldsymbol{\lambda}_i = [f_{ix}, f_{iy}, f_{iz}, m_{ix}, m_{iy}, m_{iz}]^T$, where f and m denote force and torque. The nonlinear friction cone is approximated as a linear polyhedral cone which limits the contact force in feasible range with respect to the friction coefficient μ: $|f_{ix}/f_{iz}| \leq \mu$, $|f_{iy}/f_{iz}| \leq \mu$. The unilateral constraints $f_{iz} > 0$ make sure the robot stays in contact with the ground. Finally, $d_x^- \leq m_{iy}/f_{iz} \leq d_x^+$, $d_y^- \leq -m_{ix}/f_{iz} \leq d_y^+$ restricts the ZMP inside support polygon which is a rectangle defined within the limits $[d_x^-, d_x^+]$ and $[d_y^-, d_y^+]$ based on the foot geometry.

Tracking of main tasks such as centroidal dynamics [25] (CoM tracking and angular momentum dissipation) and body position and orientation tracking are explained below. The whole-body linear momentum $\boldsymbol{l}_G$ and angular momentum $\boldsymbol{k}_G$ are commanded with PD control laws:

$$\begin{cases} \dot{\boldsymbol{l}}_G^{\text{ref}} = m(\ddot{\boldsymbol{p}}_G^{\text{des}} + \boldsymbol{K}_p^l(\boldsymbol{p}_G^{\text{des}} - \boldsymbol{p}_G) + \boldsymbol{K}_d^l(\dot{\boldsymbol{p}}_G^{\text{des}} - \dot{\boldsymbol{p}}_G)) \\ \dot{\boldsymbol{k}}_G^{\text{ref}} = -\boldsymbol{K}_d^k\boldsymbol{k}_G. \end{cases} \tag{21}$$

where $\boldsymbol{p}_G^{\text{des}}$, $\dot{\boldsymbol{p}}_G^{\text{des}}$ and $\ddot{\boldsymbol{p}}_G^{\text{des}}$ are desired CoM position, velocity and acceleration which come from the template model trajectory. $\boldsymbol{p}_G$ and $\dot{\boldsymbol{p}}_G$ denote the current state of the CoM. $\boldsymbol{K}_p^l$ and $\boldsymbol{K}_d^l$ are the feedback gains. For the angular momentum part $\boldsymbol{k}_G$, we only dissipate it and $\boldsymbol{K}_d^k$ is the damping coefficient and $\boldsymbol{k}_G$ is the current angular momentum of the robot.

For any interested part of the robot, its position and orientation can be also tracked through PD control laws:

$$\begin{cases} \ddot{\boldsymbol{p}}^{\text{ref}} = \ddot{\boldsymbol{p}}^{\text{des}} + \boldsymbol{K}_p^p(\boldsymbol{p}^{\text{des}} - \boldsymbol{p}) + \boldsymbol{K}_d^p(\dot{\boldsymbol{p}}^{\text{des}} - \dot{\boldsymbol{p}}) \\ \dot{\boldsymbol{\omega}}^{\text{ref}} = \dot{\boldsymbol{\omega}}^{\text{des}} + \boldsymbol{K}_p^\omega log(\boldsymbol{R}^{\text{des}}\boldsymbol{R}^T) + \boldsymbol{K}_d^\omega(\boldsymbol{\omega}^{\text{des}} - \boldsymbol{\omega}) \end{cases} \tag{22}$$

where $\boldsymbol{p}^{\text{des}}$, $\dot{\boldsymbol{p}}^{\text{des}}$ and $\ddot{\boldsymbol{p}}^{\text{des}}$ are respectively the desired position, velocity and acceleration which come from the template model. $\boldsymbol{R}^{\text{des}}$, $\boldsymbol{\omega}^{\text{des}}$ and $\dot{\boldsymbol{\omega}}^{\text{des}}$ are respectively the desired orientation, angular velocity and angular acceleration. $\boldsymbol{p}$ and $\dot{\boldsymbol{p}}$ denote the current position and velocity. $\boldsymbol{R}$ and $\boldsymbol{\omega}$ are current orientation and current angular velocity. $log(\boldsymbol{R}_F^{\text{des}}\boldsymbol{R}_F^T)$ is the logarithm of a orientation matrix which gives the error between the desired and current orientation [26].

IV. Simulation

The simulation environment used in this paper is Gazebo + ROS (Robot Operating System). The default physics engine ODE (Open Dynamics Engine) has been chosen to simulate the whole-body robot. The control loop runs at 1 kHz and control commands are sent to Gazebo through ROS. Two dimensional sagittal plane hopping has been simulated first to verify the proposed method. After that, running in three dimensional space has been conducted by composing two template models [27].

A. Hopping in Sagittal Plane

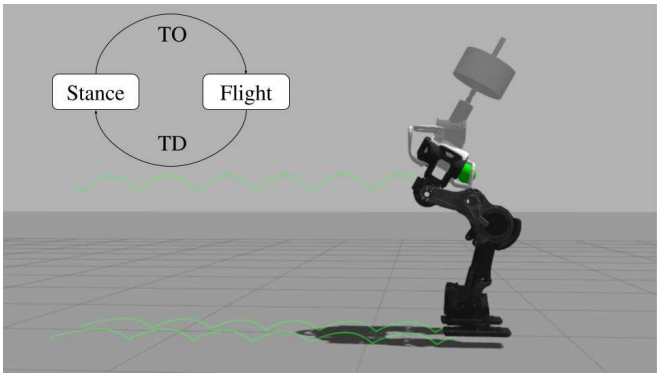

Fig. 3. Snapshot of sagittal plane hopping. The green sphere represents the CoM of the whole-body robot. The three green traces show the history of CoM and center points of the feet. Two states (Stance and Flight) and two events (TD and TO) are involved.

The hopping motion in sagittal plane can be well approximated by a two dimensional SLIP model. Two phases involved in the motion: stance phase and flight phase. A state machine has been employed to monitor the phase transitions. It is constantly checking TO or TD event to trigger the corresponding transition. TD happens when the feet touch the ground. Different checking methods (feet height, force-torque sensor) may trigger the transition at varying moment. Thanks to the energy boost and cut-off (15) (16) correction, the robot could end up with the same TO energy level.

Tasks involved in each phase are different. The whole-body controller will switch between the various tasks based on the state machine.

During the stance phase, tasks being closely tracked are: centroidal dynamics (21), torso orientation regulation (22) and ground reaction force distribution (equal distribution between the feet). For the feet, no specified goals are given, they adapt to the ground due to the non-slip constraints (19). At the TD moment, the high level controller will forward simulate once with the SLIP model (rest length modified for energy injection purpose) to get the whole CoM trajectory during the following stance phase. This CoM trajectory is then been used as desired tracking trajectory for the whole-body controller.

During the flight phase, the robot is under-actuated. The system CoM follows a ballistic trajectory. The only task to be controlled is the foot placement. In this work, no trajectories are specifically designed for the feet. The foot placement targets with respect to CoM are calculated from the touchdown angle provided by the trained neural network:

$$\begin{cases} {}^{G}x_f = l_0 \sin(\theta_{TD}) \\ {}^{G}z_f = l_0 \cos(\theta_{TD}) \end{cases} \tag{23}$$

For accurate velocity tracking, on top of the touchdown angle provided by the trained neural network, an simple PID controller has been added:

$$\begin{aligned} \theta_{TD} &= \theta_{ff} + \theta_{fb} \\ &= f_{NN}([\dot{x}, \dot{x}^*]) + \text{PID}(\dot{x} - \dot{x}^*) \end{aligned} \tag{24}$$

where θ_{TD} is the reference touchdown angle sent to the whole-body controller which is composed of a feed-forward term θ_{ff} and a feedback term θ_{fb}. $\dot{x}$ is the current CoM velocity and $\dot{x}^*$ is the desired one during the next flight phase.

For this sagittal plane hopping case, the goal is to regulate the forward speed to 1.0 m/s. The robot is released from a 0.85 m height in the air and with an initial velocity of 0.3 m/s in x direction. It directly enters flight phase after releasing. These initial states are chosen rather randomly without special calculation. The CoM velocity recorded from the simulation is plotted in Figure 4. As a comparison, we also plot the data recorded from another simulation in which the Raibert foot placement controller has been used [28]. Theoretically, the SLIP model should be able to regulate to any achievable velocity in one step. However, this ability is limited by the touchdown angle range and also by the kinematic limits and actuation limits presented in the whole-body robot. In spite of that, it can be seen that the neural network controller took fewer steps to reach the desired velocity and also with less regulation error. In both cases, the energy correction law successfully regulate the energy to the desired level within one step. The results also prove its generality.

B. Running

Running is a three dimensional movement. In the previous section, the two-dimensional case has demonstrated the effectiveness of the neural network controller. To extend it to three dimensional space, two possibilities are: 1) composing two two-dimensional SLIP model to generate three dimensional running. 2) considering the 3D-SLIP model. The later one requires generating new simulation data and train a new neural network with expanded input and output dimensions. In this paper, we adopt the former idea. Without any modification of the trained neural network, we directly apply it to the lateral plane motion. Actually, observing the

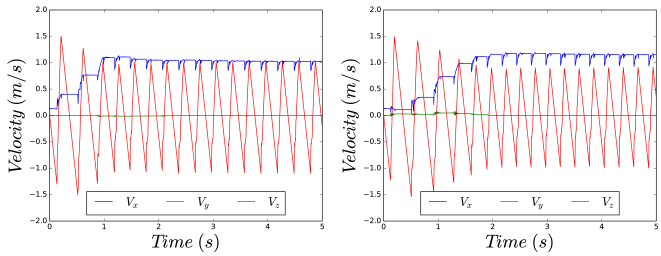

Fig. 4. Sagittal plane hopping CoM velocity. On the left side is the results from the neural network controller, and the right one comes from the Raibert controller. In both cases, the robot are released from the same height of 0.85 m with a forward velocity of 0.3 m/s. The target velocity is 1 m/s.

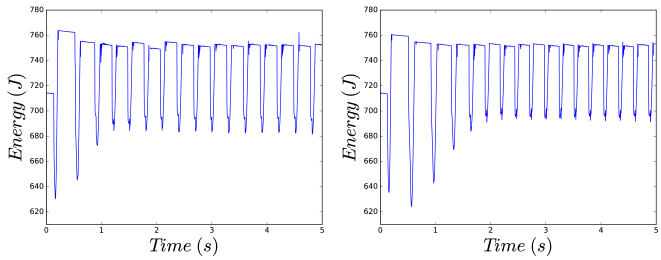

Fig. 5. Sagittal Plane Hopping Energy Level. Results come from different controllers but the same energy regulation law. The left one is our neural network controller and the right one is the Raibert controller

training data, a large amount of simulation ends up with reversed TO velocity comparing to the initial velocity which is exactly the case of lateral hopping motion. Therefore, running is treated as a hopping motion composed of sagittal hopping and lateral hopping, each component is governed by a two-dimensional SLIP model. These two template models are synchronized by a state machine as shown in Figure 6.

Tasks controlled in single stance phase are: centroidal dynamics, torso orientation and swing foot placement tracking. For the stance foot, no specified goals are given, it adapts to the ground due to the non-slip constraints (19). No ground reaction force distribution is needed since all forces comes from the single stance foot. In flight phase, only the foot preparing for landing is paid more attention to and it is controlled carefully to track the touchdown angle provided by neural network. The other foot stays in idle mode in horizontal direction and only keeps a clearance between itself and the ground. Additionally, any foot in the air is always controlled to be parallel to the ground.

Again, the CoM velocity results are compared between the neural network controller and the Raibert controller are shown in Figure 7. The neural network controller shows better regulation speed and less steady state error.

V. Conclusion

In this paper we proposed to use a deep neural network to encode the dynamics of a simple template model and then map to the whole-body robot. Different from the nonlinear optimization based approach or the classical tabular method, it transfers most of the computations offline. Once trained, the query of learned knowledge is very fast and can

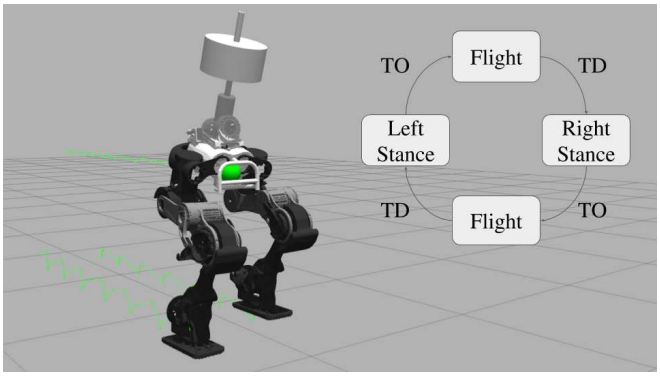

Fig. 6. Snapshot of running. The green sphere stands for the CoM position of the robot. The three green traces represent the history of CoM and center points of the feet. State machine includes three states: Flight, Left Stance, Right Stance. Two events (TO and TD) trigger the transition between these states.

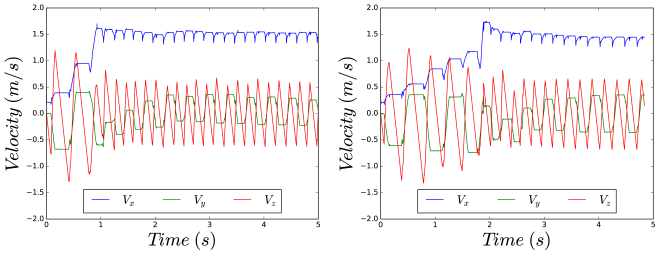

Fig. 7. Running CoM velocity plot. Results from composed neural network controller (left) and composed Raibert controller (right). The robot is released from the height of 0.85 m with a forward velocity of 0.2 m/s. After three steps it regulates to desired velocity 1.5 m/s.

be embedded into real-time control framework. The two-dimensional SLIP model long-term dynamics (return map) has been successfully learned by the neural network. The approach itself is general and not limited to this 2D case with low-dimensional inputs and outputs. In future work, we plan to add more freedoms to this model, for example energy level changes, varying leg stiffness. More interesting extension would be the 3D-SLIP model.

Acknowledgment

This work is supported by the European Horizon 2020 robotics program CogIMon (ICT-23-2014 under grant agreement 644727).

References

[1] R. J. Full and D. E. Koditschek, "Templates and anchors: neuromechanical hypotheses of legged locomotion on land," *Journal of Experimental Biology*, vol. 202, no. 23, pp. 3325–3332, 1999.

[2] R. Alexander and A. Jayes, "Vertical movements in walking and running," *Journal of Zoology*, vol. 185, no. 1, pp. 27–40, 1978.

[3] R. Blickhan and R. Full, "Similarity in multilegged locomotion: bouncing like a monopode," *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, vol. 173, no. 5, pp. 509–517, 1993.

[4] H. Geyer, A. Seyfarth, and R. Blickhan, "Compliant leg behaviour explains basic dynamics of walking and running," *Proceedings of the Royal Society B: Biological Sciences*, vol. 273, no. 1603, pp. 2861–2867, 2006.

[5] M. Ahmadi and M. Buehler, "Controlled passive dynamic running experiments with the ARL-monopod II," *IEEE Transactions on Robotics*, vol. 22, no. 5, pp. 974–986, 2006.

[6] U. Saranli, M. Buehler, and D. E. Koditschek, "RHex: A simple and highly mobile hexapod robot," *The International Journal of Robotics Research*, vol. 20, no. 7, pp. 616–631, 2001.

[7] R. Playter, M. Buehler, and M. Raibert, "BigDog," in *Pro. of SPIE 6230, Unmanned Systems Technology VIII*, pp. 62302O–62302O, 2006.

[8] J. A. Grimes and J. W. Hurst, "The design of ATRIAS 1.0 a unique monoped, hopping robot," in *Proceedings of the 2012 International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, pp. 548–554, 2012.

[9] M. H. Raibert *et al.*, *Legged robots that balance*, vol. 3. MIT press Cambridge, MA, 1986.

[10] W. J. Schwind, *Spring loaded inverted pendulum running: a plant model*. PhD thesis, Electrical Enginnering: Systems, University of Michigan, 1998.

[11] A. Wu and H. Geyer, "The 3-d spring–mass model reveals a time-based deadbeat control for highly robust running and steering in uncertain environments," *IEEE Transactions on Robotics*, vol. 29, no. 5, pp. 1114–1124, 2013.

[12] P. M. Wensing and D. E. Orin, "High-speed humanoid running through control with a 3d-slip model," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 5134–5140, IEEE, 2013.

[13] S. G. Carver, N. J. Cowan, and J. M. Guckenheimer, "Lateral stability of the spring-mass hopper suggests a two-step control strategy for running," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 19, no. 2, p. 026106, 2009.

[14] M. H. Raibert and F. C. Wimberly, "Tabular control of balance in a dynamic legged system," *IEEE Transactions on systems, man, and Cybernetics*, no. 2, pp. 334–339, 1984.

[15] D. Koepl and J. Hurst, "Force control for planar spring-mass running," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 3758–3763, IEEE, 2011.

[16] H. Herr, A. Seyfarth, and H. Geyer, "Speed-adaptive control scheme for legged running robots," Nov. 13 2007. US Patent 7,295,892.

[17] M. Hutter, C. D. Remy, M. A. Höpflinger, and R. Siegwart, "Slip running with an articulated robotic leg," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 4934–4939, IEEE, 2010.

[18] Y. You, Z. Li, D. G. Caldwell, and N. G. Tsagarakis, "From one-legged hopping to bipedal running and walking: A unified foot placement control based on regression analysis," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4492–4497, IEEE, 2015.

[19] N. G. Tsagarakis, D. G. Caldwell, F. Negrello, W. Choi, L. Baccelliere, V. Loc, J. Noorden, L. Muratore, A. Margan, A. Cardellino, *et al.*, "Walk-man: A high-performance humanoid platform for realistic environments," *Journal of Field Robotics*, 2016.

[20] B. J. Stephens and C. G. Atkeson, "Dynamic balance force control for compliant humanoid robots," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 1248–1255, IEEE, 2010.

[21] P. M. Wensing and D. E. Orin, "Generation of dynamic humanoid behaviors through task-space control with conic optimization," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 3103–3109, IEEE, 2013.

[22] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, "Optimization based full body control for the atlas robot," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pp. 120–127, IEEE, 2014.

[23] C. G. A. Salman Faraji, Soha Pouya and A. J. Ijspeert, "Versatile and robust 3d walking with a simulated humanoid robot (atlas) a model predictive control approach," *IEEE International Conference on Robotics and Automation*, 2014.

[24] A. Herzog, L. Righetti, F. Grimminger, P. Pastor, and S. Schaal, "Balancing experiments on a torque-controlled humanoid with hierarchical inverse dynamics," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 981–988, IEEE, 2014.

[25] D. E. Orin, A. Goswami, and S.-H. Lee, "Centroidal dynamics of a humanoid robot," *Autonomous Robots*, vol. 35, no. 2-3, pp. 161–176, 2013.

[26] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[27] A. De and D. E. Koditschek, "Averaged anchoring of decoupled templates in a tail-energized monoped," in *Robotics Research*, pp. 269–285, Springer, 2018.

[28] M. H. Raibert, H. B. Brown Jr, and M. Chepponis, "Experiments in balance with a 3d one-legged hopping machine," *The International Journal of Robotics Research*, vol. 3, no. 2, pp. 75–92, 1984.