Neural-Network-Controlled SLIP Model 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 then can be mapped to the whole-body model through an optimization-based inverse dynamics controller.**

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]. Due to its simplicity and platform-independence property, it has been widely used in the design and control of legged robots [3]–[5].

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 [6]. A large body of research in the SLIP literature has been directed towards more accurate and realistic controls, most of which can be categorized into two schemes: the methods which implement dead-beat like controllers through solving the running return maps [7], [8]; 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 [9]–[11]. 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.

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.

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

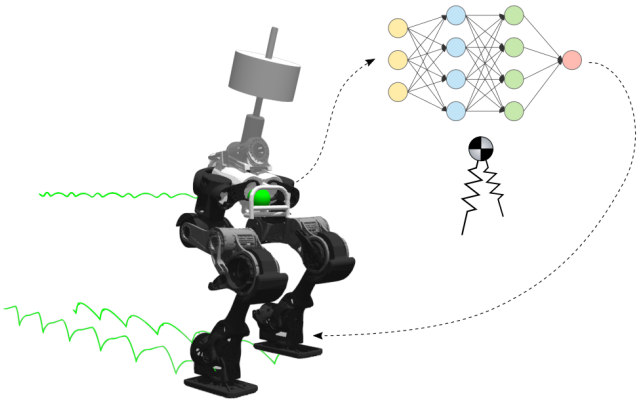

Fig. 1. The Center of Mass dynamics of the robot is controlled to match that of a SLIP model. A neural network trained offline with a large amount of SLIP simulation data is used to encode the foot placement behaviour.

II. Deep Neural Network for SLIP-like Motion Embedding

A. Spring Loaded Inverted Pendulum Model

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. 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 $\dot{x}$ due to the total energy conservation throughout the whole step. 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:

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

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

$$\theta^*_{TD,n} = f^{-1}(\dot{x}_n, \dot{x}^*_{n+1}) \tag{2}$$

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 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:

$$\begin{aligned} \theta^*_{TD,n} &= \underset{\theta}{\operatorname{argmin}} |\dot{x}^*_{n+1} - f(\dot{x}_n, \theta)| \\ s.t. \quad & \theta_{min} < \theta < \theta_{max} \end{aligned} \tag{3}$$

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 (2).

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 (1). 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:

$$\dot{x}^{(i)}_1 = f(\dot{x}^{(i)}_0, \theta^{(i)}_0) \tag{4}$$

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

$$\begin{cases} \boldsymbol{x}^{(i)} = [\dot{x}^{(i)}_0, \dot{x}^{(i)}_1]^T \\ \boldsymbol{y}^{(i)} = [\theta^{(i)}_0] \end{cases} \tag{5}$$

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

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 [9] 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. For a given input $\boldsymbol{x}^{(i)}$, we can define the FNN in a recursive way as follows:

$$\begin{aligned} \boldsymbol{h}_l &= f_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l), \quad \forall l \in \{1, \cdots, L\} \\ \text{with} \quad & \boldsymbol{h}_0 = \boldsymbol{x}^{(i)} \quad \text{and} \quad \boldsymbol{h}_L = \hat{\boldsymbol{y}}^{(i)}, \end{aligned} \tag{6}$$

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:

$$\mathcal{L}_{MSE} = \sum_{i=1}^{N} ||\boldsymbol{y}^{(i)} - f_{NN}(\boldsymbol{x}^{(i)}; \boldsymbol{W})||^2, \tag{7}$$

along with the Adam optimizer.

III. 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 non-linear 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 be embedded into real-time control framework. The encoding of two-dimensional SLIP model long-term dynamics (return map) has been introduced in this paper. The approach is general and can be well extended to three-dimensional SLIP model.

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. 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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