# Learning to Adapt the Trotting Gait of the Solo Quadruped

Michel Aractingi, Pierre-Alexandre Leziart, Thomas Flayols, Julien Perez, Tomi Silander, Philippe Souères

# Learning to Adapt the Trotting Gait of the Solo Quadruped

Michel Aractingi[12*], Pierre-Alexandre Leziart[1], Thomas Flayols[1], Julien Perez[2], Tomi Silander[2] and Philippe Souères[1]

***Abstract*— Predefined gait patterns for quadruped locomotion can hardly be optimal in all situations with regard to stability, cost of transport and velocity tracking error. Hence, in this work, we tackle the challenge of adapting a predefined trotting gait, implemented in the model-based controller of Solo, to optimize both energy consumption and velocity tracking. To this end, we propose a model-free reinforcement learning method for adapting the timings of the contact/swing phases for each foot. The learned agent augments a control pipeline that was previously developed for the Solo robot. We also propose to use a self-attention mechanism over the history of states in order to extract useful information for adapting the gait. Through a comprehensive set of experiments, we demonstrate how, compared to the nominal gait, our method significantly reduces energy consumption, better tracks the desired velocity, and makes it possible to reach higher speeds. A video of the method is found at https://youtu.be/ykbDUyASXs4.**

## I. Introduction

Quadrupedal locomotion has become an increasingly popular topic of robotics and artificial intelligence research [1], [2], [3], [4]. Unlike wheeled robots, legged robots can traverse challenging terrains [5]. Recently, several robotic platforms have been developed for mastering quadruped locomotion [6], [7], [8]. Robots like Spot, Mini Cheetah [8], HyQ [9], ANYmal [6] or Laikago [10] provide reference test benches for designing control approaches. Solo [7] is a more recent alternative which provides a reliable, low-cost, open-access quadruped within the Open Dynamic Robot Initiative. Previous work on these platforms has managed to develop control schemes that feature dynamic and robust locomotion [11], [12], [13].

Many classical methods propose complex control architectures composed of sequences of blocks with hand-tuned parameters. Each block outputs a solution to one part of the problem based on dynamic models and estimated states. Among them, the control scheme developed in [12] for Mini Cheetah constitutes a reference in the domain. It combines Model Predictive Control (MPC) and Whole-Body Control (WBC). The MPC is tasked with long horizon planning based on the under-actuated part of the dynamics while the WBC handles low-level control at finer timesteps. This hybrid architecture proved to be stable while achieving a record running speed for this robot. A similar control pipeline, including simplifying solutions for the computation of the WBC, was used for Solo in [14].

Model-free reinforcement learning (RL) makes it possible to learn controllers without hand-tuning such control blocks. RL methods have shown some success in learning locomotion tasks [2], [15], [16], particularly in adapting to new situations where classic controllers fail to work satisfactorily [17]. A hierarchical RL approach was used in [16] to train policies at two levels of control: planning and execution. Then, in [15] a multiple actor-critic approach was introduced for training several policies that specialize in different maneuvers such as running, skipping and jumping. However these learning methods require the availability of privileged information about the terrain, which is only feasible when running experiments in simulation. Moreover, RL methods usually require lots of data, which limits training to simulated environments, hence creating a problem of transferring learned skills from simulation to hardware [1]. Learning locomotion policies for real platforms has also been studied. For example, reference trajectories collected from motion capture recordings of a real dog were used to train a policy to mimic animal behaviours in [2]. In [1], a model of the actuation dynamics of the real ANYmal was used in simulation to produce policies that adapt to actuator noise. A meta learning approach was used in [17] to adapt to the noise in the dynamics when deployed on the real system. In all cases, more work and data were required to bridge the sim2real gap.

Combining model-based control and RL has been proposed before. Such methods often replace or augment certain control blocks with function approximators such as neural networks that can be trained to optimize decisions that are difficult to model. Recently, a policy that learns to choose the next contact sequence from a set of predefined contacts was proposed in [3]. In [5], the authors proposed learning a swing phase delta-variable, which decides the contact phase of the foot, along with the displacement of the step position. A method to learn gait transitions by learning the gait schedule as a function of the reference velocity was introduced in [18]. Modification of the guided policy search, where learning is guided by an MPC with the objective of optimizing the control Hamiltonian, was proposed in [19]. Learning of high-level decision variables for a low-level MPC was also proposed to adapt to difficult situations [20]. Model-based motion planning was used in [4] to train a gait planner and gait executioner policies when moving on non-flat terrains. With learning, one can obtain context-dependent control parameters that are automatically adapted to better generalize the decisions. Moreover, limiting the use of RL to specific sub-problems of the control naturally requires less data than learning the full control from scratch.

In this paper, we study the problem of how to adapt the

[1]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[2]NAVER LABS Europe, 6 chemin de Maupertuis, Meylan, 38240, France
*Corresponding author: michel.aractingi@naverlabs.com

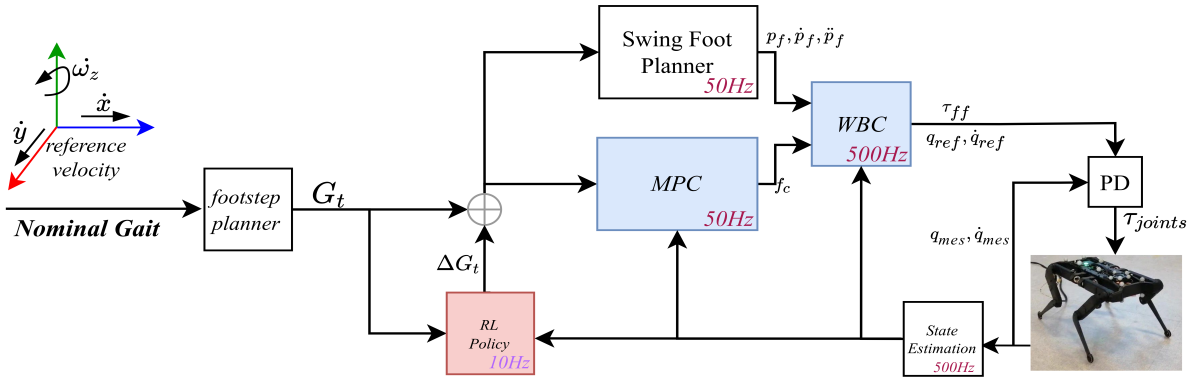



Fig. 1: Description of the controller. A user commands the reference velocity and gait. The planner outputs a gait matrix $G_t$ which is updated by the RL policy. Considering this updated value, the MPC plans contact forces for the ground feet while the swing feet trajectories are determined by a planner. The information from both blocks are sent to the whole-body control that outputs reference torques and reference joint positions and velocities. An impedance controller finally computes the joint torques based on encoders measurements.

gait of a quadruped in order to improve its spent energy and velocity tracking. Our motivation comes from the fact that humans and animals can lower their cost of transport by adapting their gait depending on their velocity [21]. Our main contribution is a method that uses a model-free deep RL policy to adapt the timings of the contact/swing phases of each foot independently. We use this policy to augment a model-based controller that was previously developed to endow the Solo quadruped with trotting, walking and static gait capabilities [14]. We show that the proposed method can adapt the nominal trotting in different situations between a walking trot pattern to a rapid trot. This improves the energy consumption and reference velocity tracking of the controller. Another contribution is the policy architecture. We argue that the prediction of events that would impact future decisions can be improved by considering a history of states. To take advantage of this history, we propose using self-attention mechanisms [22] that are commonly used in language tasks to handle sequences of inputs. The experiments demonstrate improvements in the convergence and performance of the learning.

The paper is organized as follows. Description of background information about the controller and RL is summarized in section II. The method and experimental results are outlined in section III and section IV respectively.

## II. Preliminaries

The robotic platform used in this paper is the Solo12 quadruped, a 12 degrees of freedom (DoFs) version of the Solo8 quadruped, introduced in [7], with three actuators per leg. The task is to follow a user-defined reference velocity. The RL approach in this paper is complementary to the model-based controller proposed by [14], with minor modifications to the foot trajectory generator, so that the planned trajectory of the swing feet can be modified and adapted by the RL agent on the fly. In the following section we briefly describe the controller used and the RL formalism.

### A. Nominal Controller Architecture

The nominal controller is centered around two main control blocks: a MPC and a WBC. The MPC computes a sequence of ground contact forces based on the centroidal dynamics model and the location of current and future footholds. Its objective is to have the body track the reference velocity. The WBC takes as inputs the desired contact forces, for the feet in stance phase, and the desired motion of feet in swing phase and outputs desired torques, positions and velocities for the 12 actuators. The final torque values sent to the actuators are the WBC torques values to which are added the feedback torques of a PD controller. While the WBC solves an instantaneous problem and provides low-level commands at high frequency (500 Hz), the MPC plans over a prediction horizon knowing the future footholds but at lower frequency (50 Hz) due to computational requirements.

The trajectory of a foot during swing phase is planned using polynomial functions to link its current position to a target position on the ground. The foot trajectory generator outputs a reference position, velocity and acceleration at each time step of the swing phase. These values are used by the WBC as references for the inverse kinematics and torques computation. A footstep planner outputs the target locations of footsteps using heuristics that rely on the gait, the current and desired body velocities to be tracked. The controller has shown successful trials on the real Solo12 platform. The controller works well for gaits where two or more feet are in contact with the ground, i.e., it supports various trotting, walking and static gaits. We refer to the controller's main paper for more details [14].

In general, a gait is defined as a periodic pattern of limb movements made during locomotion, both for robots and animals. Animals often have the ability to adapt their gait depending on the environmental conditions and desired characteristics such as speed, stability, manoeuvrability or energy efficiency. Formally, Solo's trotting sequence is determined at time $t$ by a binary gait matrix $G_t \in \{0,1\}^{M\times 4}$ that

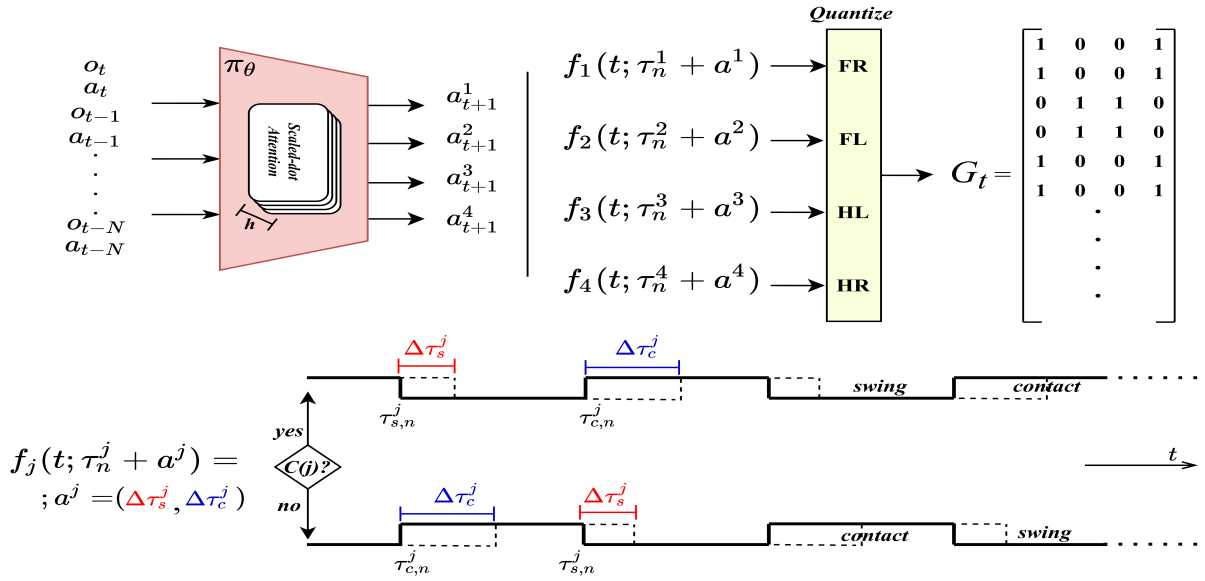



Fig. 2: Multi-headed self-attention layers, with $h = 8$ heads, are used as the base of the policy. The actions modify the contact sequence given by the oscillations of each leg. The bottom plot shows an example of the nominal sequence of contact/swing in bold. Depending on the current contact state of foot $j$, the dotted line draws the new oscillation after the shifts in timings given by the policy are taken into account.

describes the planned feet contacts for the incoming $M$ time steps. Column $j$ of the matrix describes the future ground contact states for foot $j$. The $i^{\text{th}}$ row of $G_t$ describes the ground contact states of the four feet that the controller should consider for the $i$-th time step of the MPC prediction horizon, i.e, at time $t + i \times \Delta t_{\text{mpc}}$, where $\Delta t_{\text{mpc}}$ is the length of one MPC time step.

A trotting gait is defined by having two diagonally opposite feet in contact with the ground while the other two are in a swing phase. The policy can adapt the gait period and the duration of both stance and swing phases within one period, and therefore the potential overlap of those phases between the feet. These modifications can lead to variations of the hard-coded trot that have different properties in terms of speed, energy consumption, reactivity and robustness. The controller's gait matrix is pre-defined to trotting with a period of 0.32s. Our work offers a method to adapt the value of the period by modifying the timings of the contact/stance phases.

### B. Reinforcement learning formalization

We model the RL learning environment as a Markov decision process (MDP) [23]. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, P_0)$, where $\mathcal{S}$ is a set of states and $\mathcal{A}$ is a set of actions. Taking an action $a$ in a state $s$ yields a stochastic reward, expectation of which is defined by a function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The environment dynamics is described by a conditional transition probability distribution $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}^+$, with the interpretation that $\mathcal{T}(s, a, s') = p(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability (density) that the next state is $s'$ given that the current state is $s$ and that action taken is $a$. $P_0$ is the initial state probability distribution. In RL setting, only spaces $\mathcal{S}$ and $\mathcal{A}$ of the MDP are known by the learning agent. By acting in the environment, the learning agent receives samples from the reward and state transition distributions.

We define a policy $\pi_\theta(h, a) = p_\theta(a_t = a \mid h_t = h)$ parameterized by $\theta$ that gives the probability of taking an action $a$ given a state-action history $h = (s_0, a_0, s_1, a_1, ..., s_t)$. The learning objective is to find the parameters $\theta$ of the policy that maximize the value of the expected discounted sum of rewards $J(\theta) := \mathbb{E}_{\pi_\theta, P_0, \mathcal{T}}[\sum_{t=0}^{H} \gamma^t R(s_t, a_t)]$, where $H$ is the horizon of the episode and $\gamma \in [0, 1]$ is the discount factor. Section III-B defines the core components of the MDP, i.e., state space, action space and reward function, that are used throughout this paper.

## III. Method

We propose a method to learn how to adapt the contact patterns for each foot in order to improve the control performance. We choose to formalize the task as a sequential decision problem and solve it with RL techniques. One can write a trajectory optimization program, based on an optimal control solution, to adapt the contact phases of the gait [24]. However, such a method would require intensive computations at each MPC cycle, whereas with a policy learned with deep RL the decision is made with a single forward pass through the network.

### A. Controlling the gait timings

While we could in theory directly control the binary indicators in the gait matrices $G_t$, this would create an intractable action space with $2^{4M}$ actions, most of which would not correspond to any relevant locomotion. Since the nature of quadrupedal locomotion is periodic, we propose to

view the creation of the gait matrices through parameterized oscillation functions. We define a base oscillation $\bar{f}(t;\tau_0,\tau_1)$ where $\tau_0 < \tau_1$ are the timings for which a change in value occurs:

$$\bar{f}(t;\tau_0,\tau_1,T) = \begin{cases} 0, & \text{if } \tau_0 < t \bmod T < \tau_1, \text{ and} \\ 1, & \text{otherwise.} \end{cases}$$

Given $C(j)$, a binary indicator of the current contact state of foot $j \in \{1\dots4\}$, the oscillation function $f_j(t) : \mathbb{R}_+ \to \{0,1\}$ describes the future contact states of foot $j$ as a function of time $t$. The oscillation is parameterized by two switch timing parameters $\tau_s^j$ and $\tau_c^j$, which indicate the beginning of the swing and stance phases of foot $j$ respectively. $f_j$ is then defined as[1]:

$$\begin{aligned} f_j(t;\tau_s^j,\tau_c^j) = \quad & \delta_{1,C(j)} * \bar{f}(t;\tau_s^j,\tau_c^j,T^j) \\ + \quad & \delta_{0,C(j)} * (1 - \bar{f}(t;\tau_c^j,\tau_s^j,T^j)), \end{aligned}$$

where the period length is defined as $T^j = \max(\tau_s^j,\tau_c^j)$.

In order to control the four oscillation functions we introduce a $4\times2$-dimensional continuous action space, $\mathcal{A} = \{(a^1,a^2,a^3,a^4)\}$ with $a^j = (\Delta\tau_s^j, \Delta\tau_c^j) \in \mathbb{R}^2$. For each foot $j$ there are two actions that define the displacement with respect to the timings of the nominal trotting gait, $\tau_n^j = (\tau_{s,n}^j, \tau_{c,n}^j)$, that are hardcoded in the controller. Controlling the deltas of the timing values rather than the values directly is the key for the method to work as it reduces the exploration space.

The gait matrix can be created by assigning $G_t[i,j] = f_j(i * \Delta t_{\text{mpc}}; \tau_n^j + a_t^j)$, where $i \in \{1,\dots,M\}$ and $j \in \{1,2,3,4\}$. We will next describe the nature of the state space, reward function and policy architecture that make the RL agent learn to effectively adapt contact timings.

### *B. MDP definition*

The MDP is defined over fixed discrete timesteps. The RL policy runs at a frequency of 10Hz. We found that this frequency gives the policy enough time for executing an action and receiving a useful learning signal, while keeping its reactivity in adapting gait sequences quickly enough.

**State definition:** We define the observation $O_t \in \mathbb{R}^{d=65}$ to contain proprioceptive information $o_t \in \mathbb{R}^{57}$ about the robot at time $t$ along with the last eight-dimensional command $a_{t-1}$. The elements composing the observation are the base height and orientation $q_{base} \in \mathbb{R}^4$ , base velocity $\dot{q}_{base} \in \mathbb{R}^6$, joint angles $q \in \mathbb{R}^{12}$ and velocities $\dot{q} \in \mathbb{R}^{12}$, feet positions relative to the body frame $p_{feet} \in \mathbb{R}^{12}$ and the current and past gait contact sequences, both four-dimensional binary vectors. The velocity reference command $\dot{q}_{ref} \in \mathbb{R}^3$ is added to the observation so that the policy is aware of the command. The observation is thus constructed $O_t = (o_t, a_{t-1})$.

The history of the last $N = 16$ observations are concatenated to construct the state $s_t \in \mathbb{R}^{N\times d}$. We argue that having a history of observations, especially when the period between each RL action is very short, is necessary for decision making in order to detect changes in the environment dynamics.

**Reward definition:** We design a basic reward function based on three terms: (1) a positive constant $c$ per timestep to encourage the policy not to commit any actions that would end the episode early, (2) the squared distance between the commanded velocity and the velocity of the robot and (3) an energy penalty term to encourage the policy to learn actions that save energy.

Our reward function is similar to the ones proposed by [3], [18]. However, we propose using the energy instead of the torques magnitudes. As the energy is a function of the torques and joint angles, optimizing the torques magnitudes, while important from a control perspective, does not equate to minimizing the energy under certain joint angles. The energy for joint $l$ at $t$ is the integration of power $P_l$ spent over the last RL timestep. The total energy at time $t$ is summed over all joints:

$$E_t = \sum_{l=1}^{12} \int_{t'=t-T_{RL}}^{t} P_{l,t'} dt',$$

where $T_{RL}$ is the time period between each RL step. The reward at time $t$ is then defined to be:

$$R_t = c - \int_{t'=t-T_{RL}}^{t} \|\dot{q}_{ref,t'} - \dot{q}_{base,t'}\|^2 dt' - \lambda E_t,$$

where $\lambda$ is the coefficient that balances the importance of energy conservation in the reward function. Throughout the experiments we used $\lambda = 10$ and $c = 1.0$.

**Policy design:** In order to take advantage of the sequence of observation-action pairs, we utilize self-attention layers in our policy model [22]. Self-attention has been very successfully used in language tasks where the input is sequential. We argue that with self-attention the model can focus on parts of the state that indicate changes in the dynamics. Moreover, it also facilitates coordination of the different legs by contextualizing inputs from one another. To our knowledge, this is the first time self-attention is used for learning locomotion. In Section IV-D, we show that using a self-attention model yields greater rewards with less samples than a standard stack of fully-connected layers.

## IV. EXPERIMENTS

In this section, we present the experimental results of training a policy to adapt the trotting gait of the Solo12 quadruped with the proposed action space. The main questions we answer are: (1) *can the policy learn to manipulate the trot so that the robot tracks the reference velocity while optimizing the energy consumption?* and (2) *What is the effect of using a self-attention mechanism in the policy network?*

**Environment:** The training process takes place in a synthetic environment. The simulation is based on PyBullet [25] that uses the Bullet physics engine for simulating rigid body dynamics and detecting collisions. The Pinocchio library [26] is used for low-level dynamics and kinematics, e.g., to get the positions of the feet in the body frame of the robot.

[1]The `mod` refers to the modulus operation to indicate that after period $T^j$ the time resets to zero.
$\delta_{i,j}$ refers to the Kronecker delta, i.e., $\delta_{i,j} = 1$ if $i = j$ else 0.

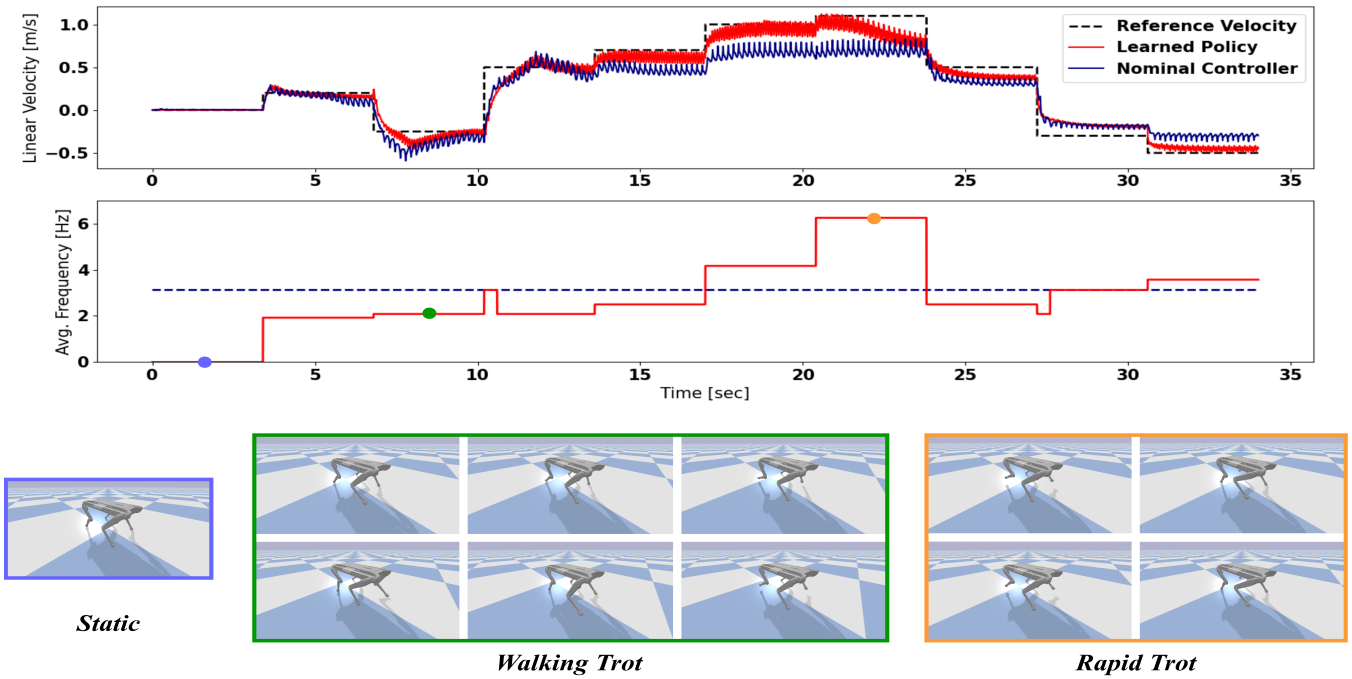



Fig. 3: Top: achieved velocity of the robot when following a predefined velocity plan (dashed line). The policy is able to get the robot closer to the desired reference in most cases. Middle: the blue dashed line represents the nominal trotting frequency. The red line indicates the average frequency of all legs as adapted by the policy. The frequency of stepping is slowed down and sped-up to accommodate the reference velocity. Bottom: snapshots of the achieved gaits at different levels of the run.

**Implementation details:** As mentioned before, a self-attention mechanism is used in the architecture of our policy-network. We use the encoder layer of the Transformer architecture as our base model [22]. The output of the encoder is propagated through a multi-layer perceptron that outputs the actions. The actions are quantized into a `Multi-discrete` space. As the contact patterns are quantized over the timestep of the MPC, the shift in the timings are multiples of the MPC timestep. Therefore, the action space is implicitly discrete. We use the Proximal Policy Optimization algorithm (PPO) [27] for learning the optimal policy. PPO is robust for a wide range of RL tasks [27]. We found that pretraining the policy representation on a torque prediction task improves the overall performance of the RL (more details in Section IV-D).

### A. Adapted gaits and velocity tracking

Our experiments demonstrate that we can learn a policy that adapts the nominal trotting gait for different reference velocities. At zero velocity, we obtain an optimal energy saving policy with a static gait where all feet are in contact with the ground. As the commanded velocity increases the gait evolves into a walking trot at low velocities. At high velocities the policy adapts the timings to output a fast rapid trotting which is more costly in terms of energy, but necessary to follow the velocity command with low error. Examples of the resulting gaits, due to the policy adaption of the nominal trotting, are shown in Figure 3[2].

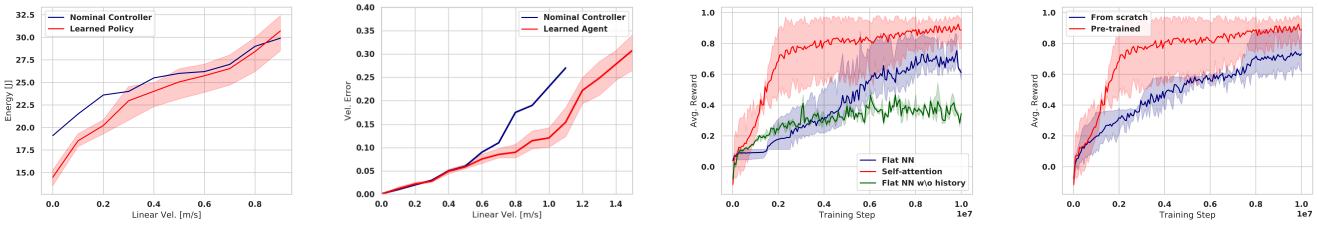
The learned policy shows higher fidelity in tracking the commanded velocity at each moment. In the middle figure, We observe a clear decrease in the frequency, making the stepping slower, for lower velocities. For high velocities the frequency is increased, thereby making the stepping faster which helps stabilize the base and follow the reference velocity.

### B. Energy efficiency

We run the learned policy five times with different random seeds in a setup where the reference velocity is gradually increased starting from 0 m/s. Figures (4a, 4b) show the trade-off between the average episodic energy consumption and velocity error for the learned policy vs. the nominal controller with fixed gait. We observe the improvement in energy consumption particularly at low velocities where the dynamic movement of the nominal trot is not necessary. At higher velocities, the velocity error is lower for the learned agent while still being comparable in energy efficiency. The lower velocity error is due to the policy learning a rapid trot that tracks higher velocities better.

At high velocities the energy consumption of the nominal controller and of the learned policy appear to be almost equal. However, it is important to note that Figure 4a is plotted as a function of the reference commanded velocity

[2]These views are snapshots of one of the policy test runs that are presented in the attached video in the supplementary material.




(a) Energy consumption. (b) Velocity error (c) Policy architecture. (d) Pre-training.

Fig. 4: Figures (a, b) are comparisons between the learned policy and the nominal controller at test time over an increasing forward velocity reference. (a) Plot of the average energy consumption per episode. (b) Plot of velocity error. The nominal trotting fails after 1.1 m/s. The Training curves in (c,d) are averaged over five random seeds. (c) Average reward for different policy architectures. (d) Effect of pre-training on a torque prediction task.

and not the actual velocity of the robot. As shown in Figure 4b, the model-based controller exhibits higher velocity error when the energy consumption is similar to the policy. Figure 3 (top) confirms that at high velocities the robot controlled by the model-based controller is slower than when controlled by the learned policy. Therefore, while the energy consumption is comparable for high reference velocities, for actual realized velocities the robot controlled by the learned policy consumes less energy than the one controlled by the nominal controller.

### C. Maintaining stability at higher speeds

We found that the controller with the nominal gait can reach the forward velocity up to 1.1 m/s before failing and falling (see Figure 4b). The proposed learned policy was able to break that limit and achieve velocities up to 2.5m/s. The experiments were conducted by gradually increasing the reference velocity over 2.0m/s. This implements a simple linear curriculum over the difficulty of the task. The result is a very rapid trot with period around 0.12s-0.16s. The attached video shows the difference between the nominal gait that fails and the gait adapted by the policy that is successful in stabilizing the robot at high speeds.

### D. Ablation studies

In this section, we present ablations of the policy architecture and of the method of pretraining the representation using torque prediction. We illustrate the advantages of using our proposed method over standard approaches. Throughout this section, we use a task where the agent is simply expected to learn how to stay still and balanced. The agent is rewarded for completing the task successfully without falling and it is penalized for its energy consumption and velocity error.

**Policy architecture:** We train a policy using three different setups: (1) The proposed self-attention based policy, (2) a feed-forward neural network policy where the input is a sequence of $N+1$ last states, and (3) a feed-forward neural network policy without history of the last $N$ states. Figure 4c shows the expected cumulative reward during the training process for each setup. By using self-attention we attain higher rewards with less training steps than using standard neural network. For the zero reference velocity, the proposed policy yields a static gait with all feet staying in contact with the ground thus conserving energy. The other policies are not able to reach such a solution in the given number of training iterations, but they settle for a dynamic gait. This explains their overall lower cumulative reward.

**Pretraining for torque predictions:** Pretraining neural networks to improve the overall sample efficiency and performance of the model is a common practice in computer vision and NLP. Therefore, we propose to pretrain our model to predict the torques using a fixed gait from the standard controller. We then use the pretrained model to initialize the policy network for learning the gait timing control task. Figure 4d illustrates the improvement in sample efficiency when using pretrained initialization. The overall asymptotic performance, in terms of average reward, is also higher. Without the pretrained representation many experiments with different seeds failed to converge. The pretrained representation stabilizes the training process and decreases the number of failed training runs.

## V. Conclusion

We proposed a model-free RL method that adapts the nominal trot pattern of a model-based controller designed for the locomotion of the Solo quadruped. The proposed action space consists of displacements of the contact/swing timings for each leg independently. We propose using a self-attention mechanism and pretraining to improve overall performance and sample efficiency. Our experiments demonstrate that the learned policy is able to adapt the gait according to the reference body velocity. The resulting behaviour conserves energy better than the nominal controller at low velocities and tracks the commanded speed better at high velocities. With the learned policy our robot is able to reach higher speeds than with the original controller. In the future we plan to investigate controllers that implement other gaits like galloping and bounding. After these promising results with a model-based controller that is already proven to work on the real Solo robot, we are currently working on testing the method on the real platform.

## REFERENCES


[1] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

[2] X. B. Peng, E. Coumans, T. Zhang, T. W. E. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," 2020.

[3] X. Da, Z. Xie, D. Hoeller, B. Boots, A. Anandkumar, Y. Zhu, B. Babich, and A. Garg, "Learning a contact-adaptive controller for robust, efficient legged locomotion," 2020.

[4] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "DeepGait: Planning and Control of Quadrupedal Gaits Using Deep Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, 2020.

[5] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020.

[6] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger, "Anymal - a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 38–44.

[7] F. Grimminger, A. Meduri, M. Khadiv, J. Viereck, M. Wüthrich, M. Naveau, V. Berenz, S. Heim, F. Widmaier, T. Flayols, J. Fiene, A. Badri-Spröwitz, and L. Righetti, "An open torque-controlled modular robot architecture for legged locomotion research," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3650–3657, 2020.

[8] B. Katz, J. D. Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6295–6301.

[9] 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*, vol. 225, no. 6, pp. 831–849, 2011. [Online]. Available: https://doi.org/10.1177/0959651811402275

[10] X. Wang, "Laikago Pro, Unitree Robotics," 2018. [Online]. Available: http://www.unitree.cc/e/action/ShowInfo.php?classid=6&id=355

[11] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control," in *IEEE International Conference on Intelligent Robots and Systems*, 2018.

[12] D. Kim, J. D. Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," 2019.

[13] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic Locomotion Through Online Nonlinear Motion Optimization for Quadrupedal Robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2261–2268, jul 2018.

[14] P.-A. Léziart, T. Flayols, F. Grimminger, N. Mansard, and P. Souères, "Implementation of a Reactive Walking Controller for the New Open-Hardware Quadruped Solo-12," Dec. 2020, working paper or preprint. [Online]. Available: https://hal.laas.fr/hal-03052451

[15] X. B. Peng, G. Berseth, and M. Van De Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," in *ACM Transactions on Graphics*, vol. 35, no. 4, 2016.

[16] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning," in *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[17] X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, and J. Tan, "Rapidly adaptable legged robots via evolutionary meta-learning," 2020.

[18] Y. Yang, T. Zhang, E. Coumans, J. Tan, and B. Boots, "Fast and efficient locomotion via learned gait transitions," *CoRR*, vol. abs/2104.04644, 2021. [Online]. Available: https://arxiv.org/abs/2104.04644

[19] J. Carius, F. Farshidian, and M. Hutter, "MPC-Net: A First Principles Guided Policy Search," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, 2020.

[20] Y. Song and D. Scaramuzza, "Learning High-Level Policies for Model Predictive Control," 2020.

[21] D. F. Hoyt and C. R. Taylor, "Gait and the energetics of locomotion in horses," *Nature*, vol. 292, no. 5820, pp. 239–240, jul 1981. [Online]. Available: https://www.nature.com/articles/292239a0

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 2017-December. Neural information processing systems foundation, jun 2017, pp. 5999–6009. [Online]. Available: https://arxiv.org/abs/1706.03762v5

[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[24] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, 2018.

[25] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[26] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integrations (SII)*, 2019.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347