ECE276C Final Project Report

1st Yuchen Zhang
University of California San Diego
La Jolla, USA
yuz008@ucsd.edu

***Abstract*—The selection of lower-level controllers are non-trivial and task dependent. this paper proposed a network architecture that learns to combine multiple low-level controllers. Results shown better performance than traditional hybrid controller but worse than the MPC implementation on the velocity tracking task.**

***Index Terms*—hierarchical control, quadruped, locomotion**

I. Introduction

Currently, high level policy learned via reinforcement learning interface with robot hardware via commands for low level controllers. The low level controllers can either be predetermined, model-based procedures as [3], trained low-level policy as [4] and various hierarchical learning algorithms [6] , or an high level policy that switches between traditional controllers and learned policies. [5]

The additional level of controllers between hardware and high level policy in the above structures can be seen as modification of hardware interfaces for the high level policy. For example, inverse kinematics controller can transform the highly non-linear relationship between joint position and foot position to a affine transform. This modification may decouple the hardware interface for high-level controllers, reducing the complexity of training. This selection of appropriate low-level controllers, however, is non-trivial and depends on the specific task. As a center velocity controller may be well suited for navigation through a maze, it leaves no space for optimizing gaits in a energy-constraint task.

I propose a network architecture that learns to combine various low-level interfaces. Different from hard switching as [5], the network I proposed combines controller outputs of the same kind with weights from the shared network. This soft switching allows controllers to function as residue actions.

II. Methods

My architecture requires controllers to be able to output torque command, or their command can be converted to torque command (for example, a target position command can be converted to torque command via a PID controller).

The shared network will compute a action $a_1, ..., a_N$ for different controllers. the dimension of the shared network output depends on the controllers' input. Next, the output of controllers of the same physical quantity is added with weights from the network. Then, this physical quantity is converted to torque command via a converter as required for controllers. Finally, the torques are weighted and summed.

See figure 1 for a diagram of the proposed architecture.

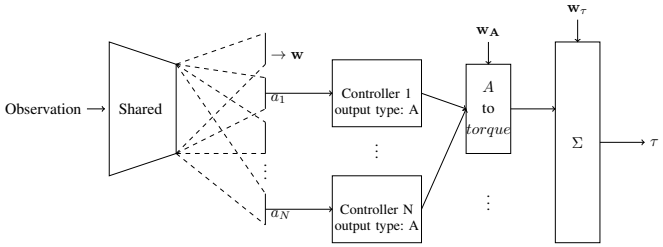

Fig. 1. Diagram of proposed architecture

III. Experiment setting

A. Architecture

For the experiment I have 3 controllers - joint torque controller, joint position controller, and a inverse kinematics controller. Both joint position and IK controller outputs $\mathbb{R}^{12}$ joint position command, they are first summed together with weights. Then, this weighted average position command is converted to torque via a PD controller. Finally, the torque output from the PD controller is summed with the output of joint torque controller according to pos-torque weight from the network to produce the final torque command.

The network is a MLP with inner layer size $[256, 128, 64]$ and elu as activation function. The input to the shared network are:

TABLE I
Network input

parameter	Dimension
base linear velocity	3
base angular velocity	3
projected gravity	3
velocity commands	3
joint position history	12×4
joint velocity	12
foot contact force	4×3

B. Joint Position Controller

The joint position controller is a PD controller with $K_p = 55\text{N} \cdot \text{m/rad}$ and $K_d = 2.0\text{N} \cdot \text{m}/(\text{rad/s})$.

C. Inverse Kinematics Controller

I created a inverse kinematics controller that takes in $\mathbb{R}^{12}$ input target foot positions and output $\mathbb{R}^{12}$ target joint angles. I modified inverse kinematics formula according to [1] to

make it work for the simulated A1 robot, which have different coordinate system at the left half of the robot as the paper's.

Let the robot leg length be l_0, l_1, l_2, from body to foot. Let the joint angles be $\theta_0, \theta_1, \theta_2$. Let the target foot position be $\vec{p} = [x, y, z]^T$. The target joint angle is computed by:

$$D = \frac{x^2 + y^2 + z^2 - l_0^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$

$$\theta_2 = -\text{atan2}(-\sqrt{1 - D^2}, D)$$

$$\theta_1 = \text{atan2}(z, \sqrt{x^2 + y^2 - l_0^2}) - \text{atan2}(l_2 \sin(\theta_2), l_1 + l_2 \cos(\theta_2))$$

$$\theta_0 = \begin{cases} -\text{atan2}(-y, x) - \text{atan2}(\sqrt{x^2 + y^2 - l_0^2}, -l_1) & left \\ -\text{atan2}(-y, x) - \text{atan2}(\sqrt{x^2 + y^2 - l_0^2}, l_1) & right \end{cases}$$

The above inverse kinematics algorithm have solution only if the square roots produce real results. The two square roots operations require the following:

$$\begin{cases} \|\vec{l}\|^2 - 2 l_1 l_2 & \leq \|\vec{p}\|^2 \leq \|\vec{l}\|^2 + 2 l_1 l_2 \\ x^2 + y^2 & \geq l_0^2 \end{cases}$$

In which $\|\vec{l}\|^2 = l_0^2 + l_1^2 + l_2^2$. I added the following remapping of the target position to guarantee a valid solution for any input position:

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \end{bmatrix} = \begin{bmatrix} x \\ \max(y, l_0) \\ z \end{bmatrix}$$

$$\vec{x}_{norm} = \begin{cases} \vec{x}_{clip} & \|\vec{x}_{clip}\| \leq \|\vec{l}\| \\ \vec{x}_{clip} \cdot \frac{\sqrt{\|\vec{l}\|^2 + 2 l_1 l_2}}{\|\vec{x}_{clip}\|} & \|\vec{x}_{clip}\|^2 > \|\vec{l}\|^2 + 2 l_1 l_2 \end{cases}$$

Finally, a affine mapping is used to transform normalized action output from network ($[-1, 1]$) to target position specified below.

TABLE II
A1 INVERSE KINEMATICS PARAMETERS

parameter	symbol	value (m)
hip length	l_0	0.08505
thigh length	l_1	0.2
calf length	l_2	0.2
x_range	$[x_{min}, x_{max}]$	$[-0.3, 0.3]$
y_range	$[y_{min}, y_{max}]$	$[0.15, 0.35]$
z_range	$[z_{min}, z_{max}]$	$[-0.3, 0.3]$

D. Environment

I used the IssacGym simulator [7] for efficient parallelization.

1) velocity command tracking: The reward function is computed as follows:

$$r = \alpha_{xy} \exp\left(-\frac{(v_x - v_{x,cmd})^2 + (v_y - v_{y,cmd})^2}{\text{exp\_coeff}}\right) + \alpha_\omega \exp\left(-\frac{(\omega_z - \omega_{z,cmd})^2}{\text{exp\_coeff}}\right) - \alpha_z v_z^2 - \alpha_\tau \sum_{i=0}^{12} \tau_i^2$$

In which the velocities are transformed to body frame via rotation and the coefficients are: exp_coeff $= 0.4$, $\alpha_{xy} = 2.0$, $\alpha_\omega = 1.0$, $\alpha_z = 0.02$, and $\alpha_\tau = 0.000025$.

The target velocity command is sampled from a uniform distribution of:

$$v_{x,cmd} \sim \mathcal{U}(0.3, 1.6)$$
$$v_{y,cmd} \sim \mathcal{U}(-0.5, 0.5)$$
$$\omega_{z,cmd} \sim \mathcal{U}(-0.5, 0.5)$$

The following terrains are constructed with the same observation and reward functions as described above.

a) Flat Plane Terrain: The ground is a flat plane with static friction coefficient and dynamic friction coefficient of 10.0 and restitution of 0.0.

b) Uneven Plane Terrain: The ground is a mesh that have uniformly distributed height $z \in [-0.05, 0.05]$ for grid points separated by $0.2m$ and a plane at $z = 0$. The mesh is sufficiently large to contain the robots during episodes. Figure 2 shows a screenshot of the terrain. Collisions between agents are disabled.

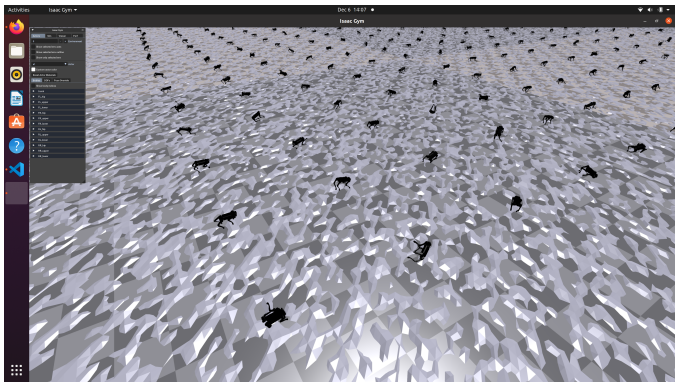

Fig. 2. uneven plane terrain

2) Dash: The reward function is computed as follows:

$$r' = \alpha_{xy} \max(0, v_x) - \alpha_\omega \omega_z^2 - \alpha_z v_z^2 - \alpha_\tau \sum_{i=0}^{12} \tau_i^2$$

$$r = \max(0, r')$$

The clip added to the reward is used to guide the policy not to intentionally terminate during early stages of training.

The dash task have only one terrain:

a) Flat Plane Terrain: The ground is a flat plane with static friction coefficient and dynamic friction coefficient of 10.0 and restitution of 0.0.

E. Training

Policies are trained using PPO provided by the IssacGymEnvs framework [9]. The training is parallelized with 256 agents and trained for 2000 epochs.

IV. Results

A. Weights Analysis

The weights between joint-position / inverse kinematics and combined position / torque is collected during training. The weight is the mean of weights of 256 actors at every 100 time steps.

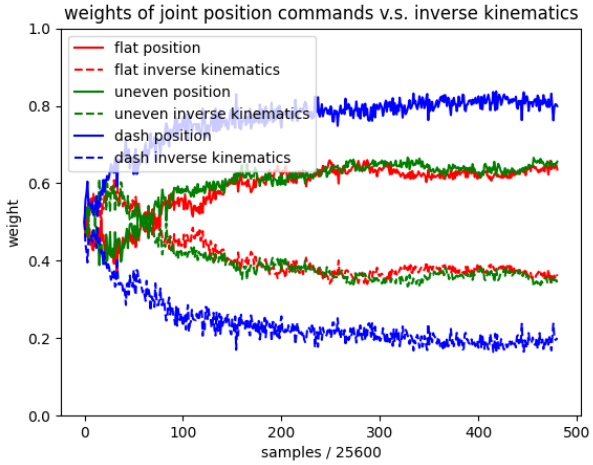

Fig. 3. weights between position commands and inverse kinematics

According to figure 3, policies that are trained for velocity tracking command have similar weights for joint position and inverse kinematics. Policy for dash task have more weights on joint position command, potentially due to the position command can actuate the robot limbs to a wider range.

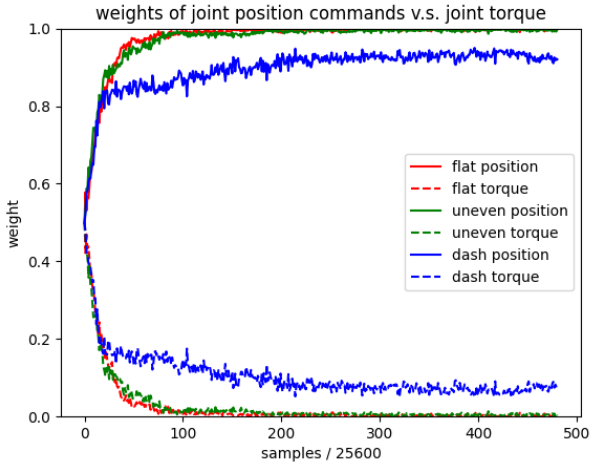

Fig. 4. weights between position commands and torque command

Figure 4 show that policies trained for target velocity tracking have a torque command weight close to 0 relying almost entirely on position commands, while the policy trained for dash task maintained a non-trivial torque weight. This fits the intuition that a dash gait need to access to lower level controllers to fine tune its details.

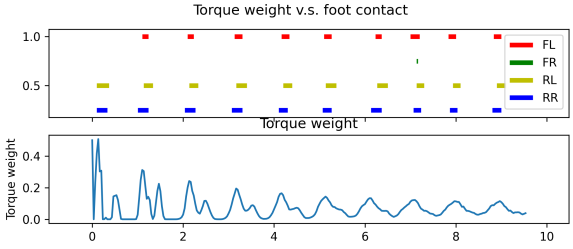

Fig. 5. weights of torque command concentrates at ground contact

A closer analysis of the dash policy shows the weight of the torque command is concentrated when the robot's foot contacts the ground. Figure 5 shows the torque weight and ground contact information for a single agent during testing.

Ablation study of the torque command weight is conducted. For comparison, the torque output is set to zero and the weight for combined position command is set to 1.0.

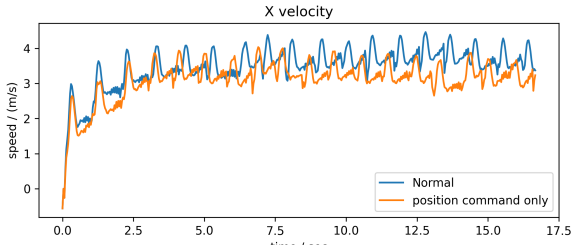

Fig. 6. removal of torque command

Removal of the torque command yields lower velocity in the dash environment and shortened air time. This shows the contribution of torque output is non-trivial.

B. Comparison to baseline

The baseline I choose is the MPC from google [10] and hybrid control - network only output joint position and joint torque actions, summing them at full weight.

I retrieved the MPC implementation from the motion imitation repository from Google research [11] and implemented the reward function and terrain at their codebase.

TABLE III
VELOCITY TRACKING TASK

Terrain	proposed	hybrid	MPC
Flat	55.53	39.58	**64.36**
Uneven	44.51	24.13	**62.63**

TABLE IV
DASH TASK

Terrain	proposed	hybrid
Flat	**80.9**	35.65

V. Discussions

According to results section, the proposed architecture performs worse compared to the MPC baseline, and performs better compared to traditional hybrid controller that does not using weights and inverse kinematics.

According to weights analysis, the proposed architecture is able to select controllers based on task requirements and performed better compared to hybrid controllers (joint position and torque). Future research on this technique is needed.

There are limitations of this architecture. Firstly, adding more potentially helpful low level controller will enlarge the action space, which will cause the network harder to train. The balance between more controllers and network size should be investigated. Secondly, all controllers is connected to the same depth of the high-level network in this work. However, different controllers represent different levels of complexity handled by it for the network, and should require different depth of network.

Acknowledgment

The codebase is based from work of my peer Minghao Zhang, a group member at professor Xiaolong Wang's lab.

References

[1] Şen, Muhammed Arif & Bakırcıoğlu, Veli & Kalyoncu, Mete. (2017). Inverse Kinematic Analysis Of A Quadruped Robot. International Journal of Scientific & Technology Research. 6.

[2] V. Makoviychuk et al., "Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning," CoRR, vol. abs/2108.10470, 2021, [Online]. Available: https://arxiv.org/abs/2108.10470

[3] X. Zhang, X. Guo, Y. Fang and W. Zhu, "Reinforcement Learning-based Hierarchical Control for Path Following of a Salamander-like Robot," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020, pp. 6077-6083, doi: 10.1109/IROS45743.2020.9341656.

[4] Jain, D., Iscen, A., & Caluwaerts, K. (2019). Hierarchical Reinforcement Learning for Quadruped Locomotion. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). doi:10.1109/iros40897.2019.8967913

[5] S. Gillen, M. Molnar, and K. Byl, Combining Deep Reinforcement Learning And Local Control For The Acrobot Swing-up And Balance Task. 2020.

[6] J. Gehring, G. Synnaeve, A. Krause, and N. Usunier, Hierarchical Skills for Efficient Exploration. 2021.

[7] V. Makoviychuk et al., "Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning," CoRR, vol. abs/2108.10470, 2021, [Online]. Available: https://arxiv.org/abs/2108.10470

[8] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning. 2021.

[9] NVIDIA-Omniverse, "Nvidia-Omniverse/Isaacgymenvs: Isaac gym reinforcement learning environments," GitHub. [Online]. Available: https://github.com/NVIDIA-Omniverse/IsaacGymEnvs. [Accessed: 07-Dec-2021].

[10] 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," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 1-9, doi: 10.1109/IROS.2018.8594448.

[11] X. B. Peng, E. Coumans, T. Zhang, T.-W. E. Lee, J. Tan, and S. Levine, "Learning Agile Robotic Locomotion Skills by Imitating Animals," Jul. 2020. doi: 10.15607/RSS.2020.XVI.064.