Evolving Dynamic Locomotion Policies in Minutes

Konstantinos I. Chatzilygeroudis[1], Constantinos G. Tsakonas[1], and Michael N. Vrahatis[1]

***Abstract*— Many effective evolutionary methods have been proposed that allow robots to learn how to walk. Most of the proposed methods have one or more of the following drawbacks: (a) utilization of hand designed open loop policies that cannot scale to different robots, and/or (b) requiring big wall time due to sample inefficiency and simulation costs, a fact that limits the practical usage of those algorithms. In the paper at hand, we propose combination of (a) a simplified model for locomotion dynamics, and (b) the effectiveness of quality-diversity algorithms, and propose a novel algorithm that is able to evolve, in less than an hour on a standard computer, generic (*e.g.* neural network), and reactive locomotion policies. Our approach makes it possible to generate in a few minutes reactive policies for locomotion that can perform dynamic motions like jumps. We also present preliminary results of transferring the behaviors to realistic simulators using a whole body inverse kinematics solver and a joint impedance controller.**

I. Introduction & Related Work

Over the last few years, we are witnessing more and more legged robots that leave the safe lab conditions and transition to real world and less constrained environments. Most of this rapid progress has been enabled by a combination of advances in control and robot learning, while optimization-based control methods have been the core ingredient in many of the milestone results [1], [2]. We are now starting to observe bipeds, quadrupeds and humanoids that can walk reliably in flat ground, climb stairs, and even operate in uneven terrain under uncertainty [3]. Through these advancements, we are also witnessing the first practical and real-world deployments of legged robots (*e.g.* ANYmal [4] and Spot [5], [6] quadrupeds).

However, manually designing controllers for legged robots is a challenging process, mostly because the dynamics of the systems are non-trivial and the robots have many degrees of freedom. Optimization-based control is one of the most promising directions and it is the one that has produced the most effective controllers [1]. Robot learning [2] is an orthogonal approach to optimization-based control, but few works have effectively merged the two paradigms [1], [7].

Many *Evolutionary Algorithms* (EAs) have been proposed to automatically design locomotion controllers with impressive results especially for robots with many feet [8]. For example, MAP-Elites has been used to pre-compute a big diverse set of controllers to allow fast adaptation despite mechanical damages in a hexapod robot [9], while similar ideas have been explored in many recent works [10]–[12]. Another successful example is EvoRBC [13] that proposes an hierarchical setup where the first level used *Quality-Diversity* (QD) methods to generate diverse behaviors and the second level learns a controller to "select" among them to solve a specific task. Contrary to model-based methods, EAs produce behaviors that require much less tuning to work and are less computationally demanding (usually the final policies are simple sinusoidal signals). Overall, the path from zero to a walking robot is much shorter with EAs than the same path with model-based optimization methods [2], [14].

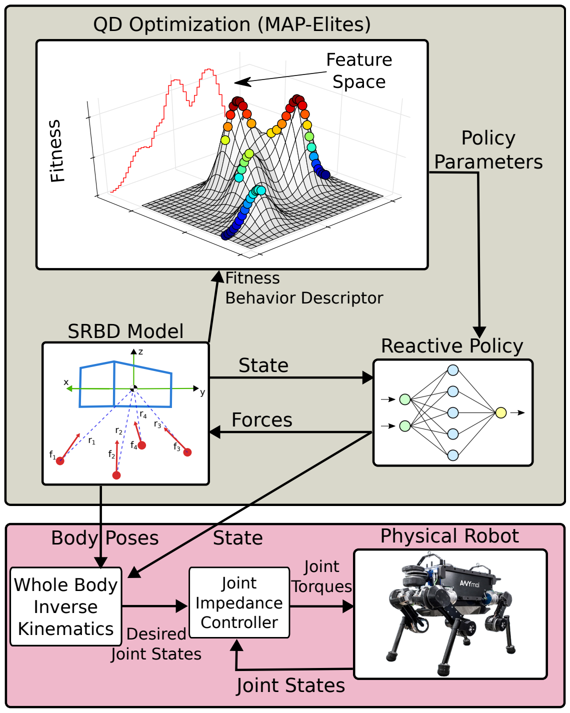

Fig. 1. Overview of EvoLoco approach

EA approaches, however, have some limitations that prohibits their practical usage in real-world settings. First, because by construction we are not in a position to understand what the controller chooses and why, it is difficult to use them on physical expensive platform like humanoid robots. Secondly, since the optimization almost exclusively happens in simulation, the resulting policies tend to overfit the simulator they have been trained on, a fact which makes the real-world transferring challenging [14]–[16]. Lastly, in an attempt to bridge the *reality gap* the usage of "expensive" simulators is common and thus most EAs for generating locomotion policies require very big wall times.

In the paper at hand, we present a novel method, called

[1]Computational Intelligence Laboratory (CILab), Department of Mathematics, University of Patras, GR-26110 Patras, Greece, `costashatz@upatras.gr`, `tsakonas_k@upnet.gr`, `vrahatis@math.upatras.gr`

EvoLoco (see Fig. 1 for an overview of EvoLoco approach), that takes inspiration from the optimization-based control literature [1], [17] and recent methods on combining learning and model-based control [7], [18]–[20], and effectively combines (a) a simplified model for locomotion dynamics [21], and (b) the effectiveness of Quality-Diversity algorithms [22], [23]. EvoLoco is one of the first methods that allows the generation of multiple diverse and different behaviors in a few minutes for any legged robot with minimal to zero hyper-parameter tuning. Overall, our method is split in three parts:

1) First, we create *a fast and lightweight but realistic simulator* using a simplified model for locomotion dynamics;
2) We, then, use this simulator to *generate many diverse reactive closed-loop behaviors* using a state-of-the-art Quality-Diversity algorithm;
3) Finally, we showcase how the generated behaviors can be transferred to a realistic simulator using a *whole body inverse kinematics solver and a joint impedance controller*.

To the best of our knowledge, EvoLoco is one of the first methods that can automatically generate reactive closed-loop locomotion policies in a few minutes on a standard desktop.

The rest of the paper is organized as follows. In Section II our approach is presented. In Section III experimental results are given and analyzed. The paper ends in Section IV with a synopsis and concluding remarks.

II. Approach

Our approach relies on the usage of a simplified but strong modeling of walking robots, the *Single Rigid Body Dynamics* (SRBD) model (Sec. II-A) [1]. Using this model, we define a simulation procedure that can produce realistic walking behaviors while being much faster than off-the-shelf rigid body simulators. We then combine this model with Quality-Diversity algorithms and propose a pipeline for efficiently (*i.e.* in a few minutes) evolving hundreds of reactive control policies that can achieve a wide range of behaviors (Sec. II-B). Finally, we devise a whole body inverse kinematics solver together with a joint impedance controller to effectively follow the control policies on a real or realistically simulated robot (Sec. II-C). *The overall goal of our method is to automatically identify and learn controllers that can make a robot (of any number of feet), walk in any direction effectively*.

A. Single Rigid Body Dynamics Model

We model walking robots with any number of legs using a single rigid body dynamics model with contacts (Fig. 2). In this model, we assume that the robot can be modeled as a single rigid body with constant moment of inertia and mass-less limbs/legs that can generate contact forces.

The rigid body has a mass $m \in \mathbb{R}^+$ and moment of inertia $\boldsymbol{I} \in \mathbb{R}^{3\times 3}$. The body can be described by its linear position $\boldsymbol{p}_b \in \mathbb{R}^3$, linear velocity[1] $\dot{\boldsymbol{p}}_b \in \mathbb{R}^3$, body orientation $\boldsymbol{R} \in SO(3)$ and angular velocity[2] $\boldsymbol{\omega} \in \mathbb{R}^3$. Each leg i has a position $\boldsymbol{p}_i \in \mathbb{R}^3$, it can generate contact forces $\boldsymbol{f}_i \in \mathbb{R}^3$, and we also assume a phase variable $\varphi_i \in \mathbb{Z}^+$ per leg that increases linearly in time. We define $T \in \mathbb{Z}^+$ the cycle time and $T_{\text{swing}} \in \mathbb{Z}^+$ the duration of the flight time for each leg ($T_{\text{swing}} < T$). If $(\varphi_i \bmod T) < T_{\text{swing}}$, then the foot i is in the swing phase; otherwise it is in contact with the ground/environment (stance phase). When the leg i is in the swing phase, then it does not exert any contact force, $\boldsymbol{f}_i = \boldsymbol{0}$. Given all the above, at each time step the root body is "getting" the following forces and torques:

$$\boldsymbol{f}_{\text{total}} = \sum_i \boldsymbol{f}_i + m\,\boldsymbol{g},$$
$$\boldsymbol{\tau}_{\text{total}} = \sum_i \boldsymbol{r}_i \times \boldsymbol{f}_i, \qquad (1)$$

where $\boldsymbol{g}$ is the gravity vector, $\boldsymbol{r}_i = \boldsymbol{p}_i - \boldsymbol{p}_b$, and $\times$ defines the cross product of two 3D vectors. Given the forces and torques in the above equations, we can compute the linear and angular accelerations of the body as follows:

$$\ddot{\boldsymbol{p}} = \frac{\boldsymbol{f}_{\text{total}}}{m},$$
$$\dot{\boldsymbol{\omega}} = \boldsymbol{I}^{-1}(\boldsymbol{R}^\top \boldsymbol{\tau}_{\text{total}} - \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega}). \qquad (2)$$

These equations are usually referred to as the *Newton–Euler equations* for rigid body motion in 3D space.

1) Forward Simulation: Once we have the accelerations we can use any integration scheme to perform forward simulation. We use *a semi-implicit Euler integration scheme*[3]:

$$\dot{\boldsymbol{p}}_{t+1} = \dot{\boldsymbol{p}}_t + \ddot{\boldsymbol{p}}dt,$$
$$\boldsymbol{p}_{t+1} = \boldsymbol{p}_t + \dot{\boldsymbol{p}}_{t+1}dt,$$
$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \dot{\boldsymbol{\omega}}dt,$$
$$\boldsymbol{R}_{t+1} = \boldsymbol{R}_t \exp(\boldsymbol{\omega}_{t+1}dt), \qquad (3)$$

where exp is the exponential mapping of $SO(3)$ [24].

Using the above toolkit, we can define a forward simulation where at each time-step the user defines the desired foot

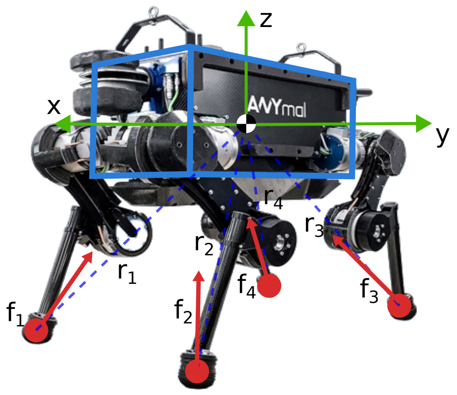

Fig. 2. Single rigid body dynamics model with contacts

[1]We denote time derivatives with an upper dot: *e.g.* $\dot{x}$.

[2]We express the angular velocity in the body coordinate frame, while the rest variables are in the world/inertial coordinate frame.

[3]We could use more advanced integration schemes, *e.g.* corresponding high-order Runge-Kutta methods, but we chose the semi-implicit Euler due its simplicity while being satisfactorily accurate and stable for the task.

forces, $\boldsymbol{f}_i$, to follow. In order to have realistic forces (*i.e.* legs that do not slip when in contact), we need to adhere to the following constraints:

$$\begin{aligned}
&\boldsymbol{f}_i \cdot \boldsymbol{n}_i = 0, \text{ if leg } i \text{ in swing phase},\\
&\boldsymbol{f}_i \cdot \boldsymbol{n}_i \geqslant 0, \text{ if leg } i \text{ in stance phase},\\
&-\mu \boldsymbol{f}_i \cdot \boldsymbol{n}_i \leqslant \boldsymbol{f}_i \cdot \boldsymbol{t}_i \leqslant \mu \boldsymbol{f}_i \cdot \boldsymbol{n}_i,\\
&-\mu \boldsymbol{f}_i \cdot \boldsymbol{n}_i \leqslant \boldsymbol{f}_i \cdot \boldsymbol{b}_i \leqslant \mu \boldsymbol{f}_i \cdot \boldsymbol{n}_i,
\end{aligned} \tag{4}$$

where $\boldsymbol{n}_i, \boldsymbol{t}_i, \boldsymbol{b}_i$ are the normal and tangential directions of the i-th contact, μ is the friction coefficient between the leg and the environment. We then do one step with the semi-implicit Euler integration.

In order to be able to fully simulate a walking robot, we need the feet of the robot to be moving. Since we assume that during the swing phase of leg i the force $\boldsymbol{f}_i = \mathbf{0}$, this means that the movement of the leg during the swing phase does not affect the motion of the root body. For this reason, we do not model explicitly the movement of the legs in swing phase, but implictly (see Sec. II-C). This modeling produces the following realization: we only need to keep track of the feet positions $\boldsymbol{p}_i$ for each stance phase. Moreover, we assume that a foot that is in stance phase should not move at all. This yields the fact that when one foot changes from stance to swing phase $\big($aka when $(\varphi_i \bmod T) = 0\big)$, we can compute the next target impact position of this foot. We do this using a Raibert-like heuristic [25]:

$$\begin{aligned}
\boldsymbol{p}_i^{\text{next}} &= \boldsymbol{p}_b + \boldsymbol{R}(\boldsymbol{r}_{\text{ref}} + k_{\text{foot}} \boldsymbol{R}^\top \dot{\boldsymbol{p}}_b),\\
\boldsymbol{p}_i^{\text{next}/z} &= \mathrm{h}(\boldsymbol{p}_i^{\text{next}/(x,y)}),
\end{aligned} \tag{5}$$

where $\boldsymbol{r}_{\text{ref}}$ is the reference position of the foot in the body frame, h is a terrain function that gives the height of the terrain at a given (x, y) position (in our case $\mathrm{h} = 0$ for all positions; *i.e.* we use a flat terrain), and k_{foot} is a user defined gain (we use $k_{\text{foot}} = 40dt$).

B. Quality-Diversity for Locomotion Generation

Getting inspiration from [9], [10], [13], we leverage *Quality-Diversity* (QD) algorithms [22], [23], [26] in order to generate a diverse set of locomotion behaviors. While QD algorithms are effective in many domains [27]–[29], they have been shown to be particularly effective in fast robot adaptation [9], [10], [30], and in designing locomotion controllers in general [9], [13], [31].

In this work, we use the MAP-Elites algorithm [27], [32]: an evolutionary algorithm that keeps diversity in a *behavior space* (this is also referred to as *feature space*). In short, at each iteration (Alg. 1), MAP-Elites alters copies of solutions, $\boldsymbol{\theta}$, that are already in an archive to form new solutions. The alterations are done with mutation and cross-over operators like in traditional evolutionary algorithms. The new solutions are evaluated and then potentially added to the cell corresponding to their behavior descriptor, $\mathbf{b}$. If the cell is empty, the solution is added to the archive. Otherwise, only the best solution is kept in the archive. The archive has a maximum capacity of k and can have many forms. In the original, implementation the archive is a grid [9], [27], while in recent extensions [22], [23], [32] more generic container and initialization schemes are adopted.

Algorithm 1 MAP-Elites algorithm

```
1:  procedure MAP-ELITES($k$)
2:      $\mathcal{A} \longleftarrow \emptyset$                    ▷ Empty Archive with capacity $k$
3:      for $i = 1 \to G$ do          ▷ Initialization: $G$ random $\boldsymbol{\theta}$
4:          $\boldsymbol{\theta}$ = random_solution()
5:          ADD_TO_ARCHIVE($\boldsymbol{\theta}, \mathcal{A}$)
6:      for $i = 1 \to I$ do              ▷ Main loop, $I$ iterations
7:          $\boldsymbol{\theta}$ = selection($\mathcal{A}$)
8:          $\boldsymbol{\theta}'$ = variation($\boldsymbol{\theta}$)
9:          ADD_TO_ARCHIVE($\boldsymbol{\theta}', \mathcal{A}$)
10:     return $\mathcal{A}$
11: procedure ADD_TO_ARCHIVE($\boldsymbol{\theta}, \mathcal{A}$)
12:     $(p, \mathbf{b}) \longleftarrow$ evaluate($\boldsymbol{\theta}$)
13:     $c \longleftarrow$ get_cell_index($\mathbf{b}$)
14:     if $\mathcal{A}(c) = null$ or $\mathcal{A}(c).p < p$ then
15:         $\mathcal{A}(c) \longleftarrow p, \boldsymbol{\theta}$
```

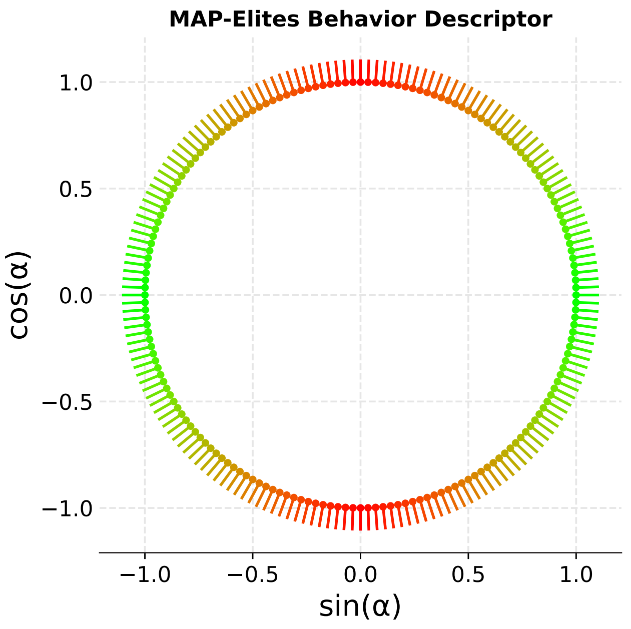

Fig. 3. Visualization of MAP-Elites behavior descriptor. Each dot represents one center of the archive, and the lines showcase the direction of the robot that corresponds to that point. The colors visualize the magnitude of the $\cos(\alpha)$ (the more red the bigger) and of the $\sin(\alpha)$ (the more green the bigger).

1) Behavior Descriptor & Performance Measure: Since we are interested in generating controllers that can walk in any direction, we need to define the behavior descriptor in such a way that allows for MAP-Elites to keep this type of diversity. In most relevant works, the behavior descriptor of choice is the final $\mathbf{b} = \{x, y\}$ position of the robot at the end of each episode/evaluation. Although, this behavior descriptor enables MAP-Elites to produce many diverse controllers, it also makes MAP-Elites produce many uninteresting behaviors. For example, in a hexapod locomotion task with this descriptor, MAP-Elites will most likely find many controllers that have traveled less than $0.1\,m$ in a $10\,s$ episode length. Although some of these behaviors might be important for fast online adaptation, most of them will not be of any actual interest.

For this reason, we define a behavior descriptor as the heading/direction that the final position of the robot "dictates", that is $\mathbf{b} = \{\cos(\alpha), \sin(\alpha)\}$, where $\alpha = \text{atan2}(y, x)$ if $x \geqslant 0$, and $\alpha = \text{atan2}(y, x) - \pi$ otherwise[4]. The tuple (x, y) defines the 2D position of the robot at the end of each episode. We also define $k = 180$ and discretize the behavior space using intervals of $\delta\alpha = 2\pi/k$. In essence, we encode the direction of the movement of the robot and discretize it at $\delta\alpha$ intervals. Fig. 3 depicts the behavior descriptor visually.

Additionally, for MAP-Elites to work effectively, we define an objective function (to compute the fitness values p) that penalizes deviations from the target α over the whole trajectory of the behavior, and rewards faster gaits (i.e., behaviors that travel bigger distances). We also include some regularization terms to prevent divergence of the algorithm: in particular, we add some terms regarding keeping the orientation close to identity and the height of the base close to the nominal one.

2) Closed-Loop Controller: As a last step for MAP-Elites, we define the controller and its parameters $\boldsymbol{\theta}$. Contrary to many relevant works that use open-loop and hand designed policies, we define a reactive policy $\pi(\boldsymbol{s}|\boldsymbol{\theta})$ as a neural network that accepts as input the SRBD state and outputs the forces to exert per foot, $\boldsymbol{f}_i$. In more detail, we use the state $\boldsymbol{s} = \{z, \dot{\boldsymbol{p}}, a_x, a_y, a_z, \boldsymbol{\omega}, \boldsymbol{r}_i, \varphi_i\} \in \mathbb{R}^{10+4N_{\text{feet}}}$, where a_x, a_y, a_z is the Euler angles (XYZ) representation of the orientation of the root body; we do not include the x, y displacement in order to not overfit specific regions of the space and have a controller that can work in any initial condition.

C. Whole-Body Inverse Kinematics and Joint Impedance Control for Behavior Tracking

The SRBD model, as we will also see in the results sections, is an effective model that can produce realistic behaviors that can be tracked on actual hardware. Nevertheless, executing the output of the model, specifically the feet forces and body accelerations, on an actual robot is not an easy task. Designing a closed-loop controller to effectively track a reference signal, even from a static plan, poses a non-trivial challenge that involves numerous design choices [33], [34]. EvoLoco does not only give a static behavior/reference signal, but outputs a closed-loop policy that adapts the signal based on the current state of the system. Tracking this closed-loop policy on a real robot is an even more challenging task that requires separate consideration.

In this work, we are interested in showcasing that the produced behaviors can indeed be realized on realistic systems and that the SRBD model is capable of producing such behaviors. For this reason, we take the following two-phase approach:

1) **Offline Phase:**
   a) We take each policy from MAP-Elites and run them inside the SRBD simulator;
   b) We record the body poses and feet positions[5];
   c) We use a whole body inverse kinematics solver to compute the joint positions that correspond to the behavior.
2) **Online Phase:**
   a) While executing the behavior on the robot, we get the joint positions that correspond to the current time and we follow them using a joint impedance controller.

In essence, we create an open-loop controller that attempts to follow the behaviors produced by MAP-Elites. We leave the realization of a real-time closed-loop controller for future work. Below we explain the key parts of each module.

1) Whole Body Inverse Kinematics: The goal of a *Whole Body Inverse Kinematics* (WBIK) solver is to find the joint positions that achieve the desired end-effector poses of multiple end-effectors at the same time. The main difference between WBIK and traditional inverse kinematics is that we formulate the problems as a non-linear program that we solve using efficient numerical methods. This is needed because when we have multiple end-effectors, there can be different chains of joints that contribute to the same end-effector and thus we cannot solve for each end-effector independently.

In order to find a solution to the problem, we solve a sequence of *Quadratic Programming* (QP) problems until we have the desired accuracy. Before going deeper into the QP formulation of the subproblems and in order to improve clarity, we define the following:

a) $\boldsymbol{q} \in \mathbb{R}^j$ is the full state of the system (with j being the number of DoFs of the robot, including the 6-DoFs of the floating base, if present);
b) $\boldsymbol{x}_e \in \mathbb{R}^l$ is the the pose (containing position and orientation) in Cartesian space of the end-effector e;
c) $\boldsymbol{J}_e \in \mathbb{R}^{l\times j}$ is the world-space Jacobian of the the end-effector e.

Having the above in mind, the QP problems are formulated as follows [35]:

$$\begin{aligned} \min_{\mathcal{X}} \; & -\frac{1}{2}\mathcal{X}^\top \boldsymbol{G}\mathcal{X} + \boldsymbol{g}^\top\mathcal{X}, \\ \text{s.t. } & \boldsymbol{H}_E\mathcal{X} = \boldsymbol{b}_E, \\ & \boldsymbol{H}_I\mathcal{X} \geqslant \boldsymbol{b}_I, \end{aligned} \tag{6}$$

where $\mathcal{X} = \dot{\boldsymbol{q}}$ are the optimization variables, and we are optimizing tasks of the form $\frac{1}{2}\left\|\boldsymbol{H}\mathcal{X} - \boldsymbol{b}\right\|^2$, and where $\boldsymbol{G} = \boldsymbol{H}^\top\boldsymbol{H}$ and $\boldsymbol{g} = -\boldsymbol{H}^\top\boldsymbol{b}$.

[4]We always normalize angles inside the interval $[-\pi, \pi]$.

[5]As we have discussed, the model does not give us targets for the feet when in swing phase. This is an easier task and we define a spline curve parameterized by time to connect two sequential stance foot positions for a user specified terrain/ground clearance (we use $0.15\,m$ for ANYmal).

In order to have a more concrete example, let us assume that we have a robot with two end-effectors and a $j = 18$ degrees of freedom. We have one desired pose per end-effector, namely $\boldsymbol{x}_1^d$ and $\boldsymbol{x}_2^d$. We also have the Jacobians, $\boldsymbol{J}_1$ and $\boldsymbol{J}_2$. Assuming that the optimization starts at $\boldsymbol{q}_k$, we can define two tasks as following:

$$\begin{aligned}\boldsymbol{H}_1 &= \boldsymbol{J}_1(\boldsymbol{q}_k),\\ \boldsymbol{b}_1 &= \dot{\boldsymbol{x}}_1^d,\\ \boldsymbol{H}_2 &= \boldsymbol{J}_2(\boldsymbol{q}_k),\\ \boldsymbol{b}_2 &= \dot{\boldsymbol{x}}_2^d,\end{aligned} \tag{7}$$

where[6]

$$\begin{aligned}\dot{\boldsymbol{x}}_1^d &= K_p(\boldsymbol{x}_1^d \ominus \boldsymbol{x}_1) - K_d\dot{\boldsymbol{x}}_1,\\ \dot{\boldsymbol{x}}_2^d &= K_p(\boldsymbol{x}_2^d \ominus \boldsymbol{x}_2) - K_d\dot{\boldsymbol{x}}_2,\\ \boldsymbol{H} &= \begin{bmatrix}\boldsymbol{H}_1\\ \boldsymbol{H}_2\end{bmatrix},\\ \boldsymbol{b} &= \begin{bmatrix}\boldsymbol{b}_1\\ \boldsymbol{b}_2\end{bmatrix}.\end{aligned} \tag{8}$$

The result of each QP problem is a joint velocity vector $\dot{\boldsymbol{q}}$ that we use to integrate and find a new joint configuration vector $\boldsymbol{q}_{k+1}$. We repeat the process for K iterations and stop once the errors $\boldsymbol{x}_i^d \ominus \boldsymbol{x}_i$ are smaller than some threshold. We perform this iterative solve for each timestep and we start the optimization of the next step from the solution of the previous timestep. We use the `whc` library for our implementation: `https://github.com/costashatz/whc`.

2) Joint Impedance Controller: In order to effectively track the produced joint positions on the real robot, we need to be compliant to ground reaction forces. Otherwise, *we risk pushing too hard on the floor and tipping the robot over*. For this purpose, we define a joint space impedance controller as [36]:

$$\boldsymbol{\tau} = -K(\boldsymbol{q} - \boldsymbol{q}_d) - D(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) + \boldsymbol{M}(\boldsymbol{q}, \dot{\boldsymbol{q}})(\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_d), \tag{9}$$

where $\boldsymbol{q}$ and $\boldsymbol{q}_d$ are the measured and desired joint positions respectively. $\boldsymbol{q}_d$ comes from the inverse kinematics solver. K and D indicate the damping and the stiffness of the controller, and $\boldsymbol{M}(\cdot,\cdot)$ is the generalized mass matrix of the robot. We integrate the desired behavior profile to get the desired joint velocities and accelerations. Since the base of the robot cannot be explicitly controlled, the first 6 dofs are ignored by the simulator.

This controller acts as a mass-spring-damper system and thus when configured with appropriate gains we get a controller that can accurately follow the desired joint positions but also be compliant to ground reaction forces.

III. Experiments

With the experiments in this section, we aim at answering the following questions:

[6] $\ominus$ denotes a meaningful difference in the task space.

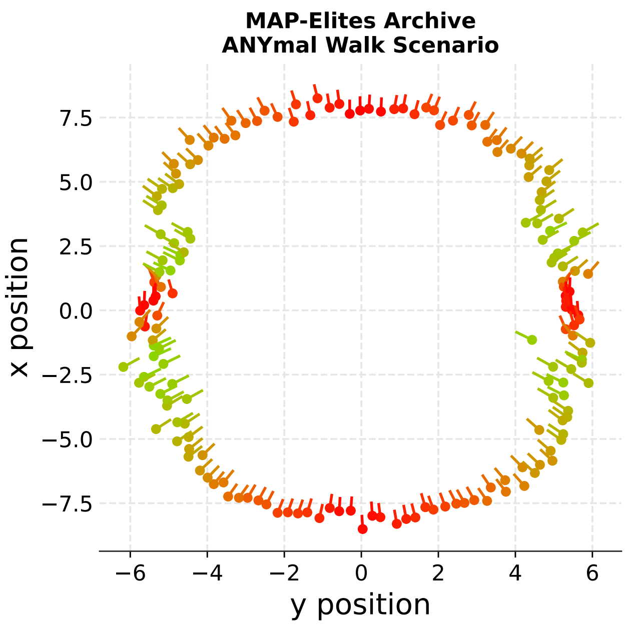

Fig. 4. The "median archive" of the MAP-Elites procedure for the ANYmal walking scenario. There are 180 dots (one for each center of the archive), and each dot represents the median (x,y) position over 10 replicates of the final position of running the controller that corresponds to this center. The lines represent the median heading over 10 replicates. The color is following the same convention as in Fig. 3.

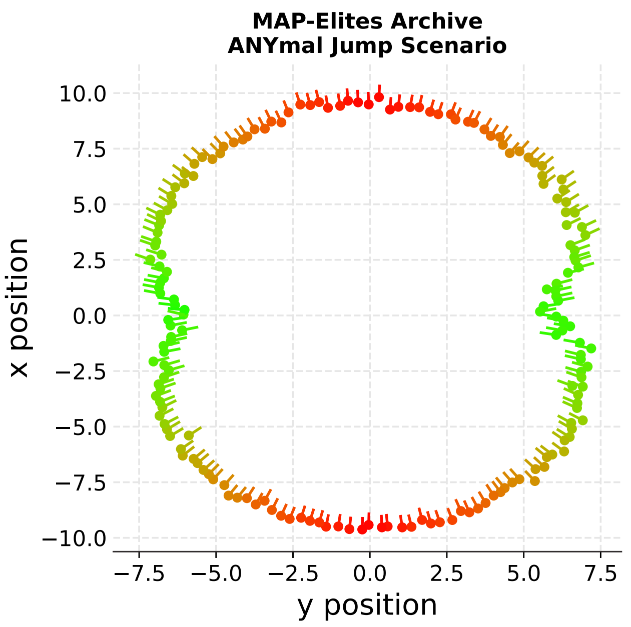

Fig. 5. The "median archive" of the MAP-Elites procedure for the ANYmal jump scenario. There are 180 dots (one for each center of the archive), and each dot represents the median (x,y) position over 10 replicates of the final position of running the controller that corresponds to this center. The lines represent the median heading over 10 replicates. The color is following the same convention as in Fig. 3.

1) Can EvoLoco produce diverse and interesting locomotion behaviors?
2) How much faster is the SRBD model (and consequently EvoLoco) compared to full dynamics simulators?
3) Can the behaviors produced by the first part of EvoLoco

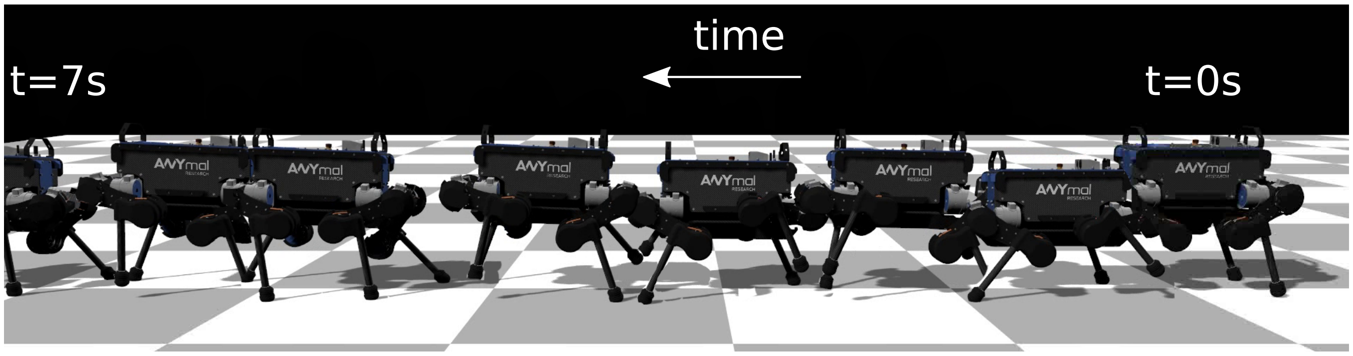

Fig. 6. One example behavior generated by the MAP-Elites procedure of EvoLoco for the ANYmal walking scenario.

be tracked by robots on realistic simulators?

In order to answer the above questions and showcase the ability of EvoLoco to operate with any robot, we devise the following scenarios:

a) A scenario where an ANYmal quadruped robot [4] learns how to **walk** in every direction;
b) A scenario where an ANYmal quadruped robot [4] learns how to **jump** in every direction.

We will use both scenarios to test *Question #1 & #3*, while we will focus on the first scenario for *Questions #2*.

A. Diverse Behavior Generation (Questions 1 & 2)

We first approximate the full ANYmal model (Fig. 2) with the single rigid body equivalent (mass m and moment of inertia $\boldsymbol{I}$). We set the parameters of the SRBD as follows:

i) dt: 0.01;
ii) T: 60;
iii) For **trotting** we set T_{swing} = 25 and the initial $\varphi_1 = 0$, $\varphi_2 = T_{\text{swing}}$, $\varphi_3 = 0$, $\varphi_4 = T_{\text{swing}}$;
iv) For **jumping/hopping** we set T_{swing} = 20 and the initial $\varphi_1 = T_{\text{swing}}$, $\varphi_2 = T_{\text{swing}}$, $\varphi_3 = T_{\text{swing}}$, $\varphi_4 = T_{\text{swing}}$.

We then run one MAP-Elites procedure for trotting and one for jumping as explained in Sec. II-B using a batch/population size of 256. We execute 10 replicates per scenario. All the hyper-parameters are the same in both cases: for the reactive policy, we use a neural network with 1 hidden layer of 5 neurons. The neural network outputs forces for all feet for every timestep; we ignore the forces for the feet that are in swing phase. Each episode has a length of $10\,s$.

The results showcase that EvoLoco is able to produce behaviors that span all the $k = 180$ cells of the behavior space for both scenarios (Fig. 4, Fig. 5). Overall, we observe that EvoLoco is able to find very effective gaits that can reach up to $8.5\,m$ when walking and up to $9.82\,m$ when jumping[7]. Qualitatively, the motions generated by the SRBD model look natural and produce dynamic behaviors (Fig. 6). A video showcasing the generated behaviors can be found at `https://youtu.be/VdyUlAAWMzQ`.

In order to showcase the speedup that we get from the SRBD model, we run the same experiment using a full

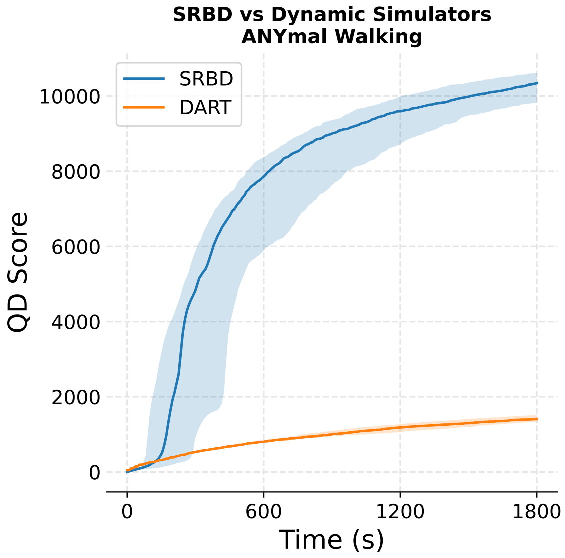

Fig. 7. Comparison between the SRBD model and full dynamic simulators. Because of the SRBD model, EvoLoco is able to produce effective gaits in less than 30 minutes. On the contrary, MAP-Elites with a fully dynamic simulator (based on the DART simulator) struggles to produce meaningful gaits in the given time budget. The QD score is the sum of the fitness values of all valid individuals inside the MAP-Elites archive [22], [26]. Solid lines are the median over 10 replicates and the shaded regions are the regions between the 25-th and 75-th percentiles.

dynamics simulator. Here it is worth mentioning that it is difficult to run exactly the same experiments, as it is not obvious how to dictate the type of gait when using a dynamic simulator. In other words, we cannot just change some step sequence parameters as in the SRBD model and get different type of gaits; one would need to find the appropriate reward function. It is the case that with the SRBD model we can more easily "dictate" specific type of gaits.

The results showcase that EvoLoco is able to run a lot more simulated interaction time than the baseline (that uses the DART simulator [37][8]) and EvoLoco generates much more effective gaits (Fig. 7). It is worth noting as well, that the baseline is trained using a parameterized open-loop periodic signal (similar to [9], [10]) and position control. In preliminary experiments with a closed-loop neural network policy and torque control, we were not able to produce any meaningful behavior in the time budget (max 30 minutes)

[7]Median values over 10 runs of the MAP-Elites procedure.

[8]We use the robot_dart wrapper.

that we allocated for each run. EvoLoco is able to run around 81906432 episodes in the time budget (≈ 45504 episodes$/s$), while when using a full dynamic simulator we were able to run around 20352 episodes (≈ 11.3 episodes$/s$, median values over 10 replicates).

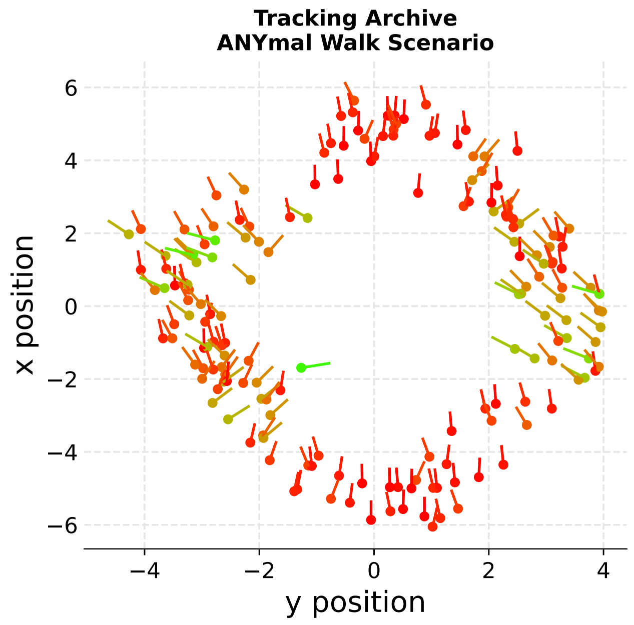

Fig. 8. The "median archive" of the MAP-Elites procedure for the ANYmal walking scenario when executing the behaviors in a realistic simulator with the WBIK solver and the joint impedance controller. There are 180 dots (one for each center of the archive), and each dot represents the median (x,y) position over 10 replicates of the final position of running the controller that corresponds to this center. The lines represent the median heading over 10 replicates. The color is following the same convention as in Fig. 3.

B. *Tracking of Generated Behaviors (Question 3)*

In this section, we perform experiments that validate the fact that the SRBD model produces realistic behaviors which can be tracked on realistic simulations. In more detail, in this section we run the full EvoLoco approach, take the policies produced by the MAP-Elites procedure, and test them on a realistic full model simulator (the same as in the previous section) using the WBIK solver and joint impedance controller as described in Sec. II-C.

We continue the ANYmal experiments from the previous sections, and the results showcase that the proposed control procedure is able to effectively follow the controllers generated using the SRBD model. In Fig. 8, we see that the final archive for the ANYmal walking scenario is similar to the one produced by the SRBD model. Qualitatively, we see that EvoLoco can even follow highly dynamic movements like repeated jumps (Fig. 9). Videos of the produced and tracked behaviors are available in the following url: `https://youtu.be/VdyUlAAWMzQ`.

IV. Synopsis and Concluding Remarks

We have proposed a novel pipeline and control structure that enables automatic generation of a diverse set of locomotion behaviors and real-time tracking of those behaviors on realistic simulated robots. The main innovation of the method lies in the effective combination of simplified dynamics with the power of quality-diversity algorithms. Our method is fast and produces reactive controllers that can run on actual robots. To the best of our knowledge, our method is one of the first ones to be able to generate reactive generic policies for locomotion in a few minutes, and can easily be incorporated in other pipelines. For example, QDax [38] is a recent pipeline for fast QD evolution through massive GPU parallelization. Our method is orthogonal to QDax since our simplified dynamics model can be fully vectorized and run on the GPU. Overall, we can get even bigger speed-ups by combining our method with QDax.

The presented methodology presents many paths for future improvements. Firstly, we can use more sophisticated simplified models (like the Centroidal Dynamics with Kinematics model [1], [17], [39]) that are more expressive than our SRBD but still much faster than off-the-shelf simulators. Another direction for future work, is to define an hierarchical learning structure, where we fastly learn a diverse set of behaviors with our proposed method, while we fine-tune them on realistic simulators (or even the real robots) [9], [40]. Lastly, the most obvious next step is to implement a state-of-the-art whole body inverse dynamics controller or model predictive control method [41], [42] in order to create a real-time closed loop controller that can follow the generated behaviors better than the proposed open-loop controller.

Finally, the main limitation of our method is the way we update the stance feet target locations. At the moment, we are using a Raibert-like heuristic (see Sec. II-A) which is effective for periodic behaviors, but can be problematic in other cases or challenging terrain. To by-pass this issue, we can use the power of QD algorithms to learn generic functions that dictate how the feet change between stance and swing phases, and where to place the feet.

Acknowledgments

This work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "3rd Call for H.F.R.I. Research Projects to support Post-Doctoral Researchers" (Project Acronym: NOSALRO, Project Number: 7541). The authors would like to thank Dionis Totsila for the constructive feedback and help with the creation of some figures.

References

[1] P. M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard, and A. Del Prete, "Optimization-based control for dynamic legged robots," *arXiv:2211.11644*, 2022.
[2] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, "A survey on policy search algorithms for learning robot controllers in a handful of trials," *IEEE Transactions on Robotics*, vol. 36, pp. 328–347, 2020.
[3] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020.
[4] M. Hutter *et al.*, "ANYmal - a highly mobile and dynamic quadrupedal robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.
[5] "Spot - The Agile Mobile Robot — Boston Dynamics," Accessed on June 17, 2023. [Online]. Available: https://www.bostondynamics.com/products/spot

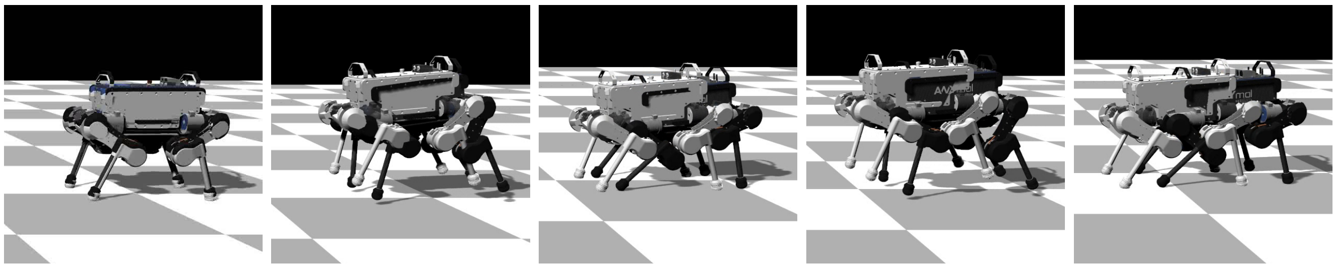

Fig. 9. From left to right, screenshots of a controller learned by EvoLoco in the ANYmal jumping scenario. This is the tracking behavior using the whole pipeline including the WBIK solver and joint impedance controller. We visualize the tracking of two consecutive jumps. The gray robot is the output of the SRBD model and the full textured robot is the dynamically simulated one.

[6] "Nebula-SPOT," Accessed on June 17, 2023. [Online]. Available: https://www.jpl.nasa.gov/robotics-at-jpl/nebula-spot

[7] F. Khadivar, K. Chatzilygeroudis, and A. Billard, "Self-correcting quadratic programming-based robot control," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.

[8] D. Gong, J. Yan, G. Zuo, *et al.*, "A review of gait optimization based on evolutionary computation," *Applied Computational Intelligence and Soft Computing*, vol. 2010, 2010.

[9] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[10] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free Trial-and-Error Learning for Robot Damage Recovery," *Robotics and Autonomous Systems*, vol. 100, pp. 236–250, 2018.

[11] M. Allard, S. C. Smith, K. Chatzilygeroudis, B. Lim, and A. Cully, "Online Damage Recovery for Physical Robots with Hierarchical Quality-Diversity," *ACM Transactions on Evolutionary Learning and Optimization*, 2023.

[12] M. Allard, S. C. Smith, K. Chatzilygeroudis, and A. Cully, "Hierarchical Quality-Diversity for Online Damage Recovery," in *Genetic and Evolutionary Computation Conference*, 2022.

[13] M. Duarte, J. Gomes, S. M. Oliveira, and A. L. Christensen, "Evolution of Repertoire-based Control for Robots with Complex Locomotor Systems," *IEEE Transactions on Evolutionary Computation*, 2017.

[14] J. Lehman, J. Clune, D. Misevic, C. Adami, L. Altenberg, J. Beaulieu, P. J. Bentley, S. Bernard, G. Beslon, D. M. Bryson, *et al.*, "The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities," *Artificial life*, vol. 26, no. 2, pp. 274–306, 2020.

[15] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Transactions on Evolutionary Computation*, vol. 17, pp. 122–145, 2013.

[16] J.-B. Mouret and K. Chatzilygeroudis, "20 years of reality gap: a few thoughts about simulators in evolutionary robotics," in *Genetic and Evolutionary Computation Conference Companion*, 2017.

[17] J. Carpentier and N. Mansard, "Multicontact locomotion of legged robots," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1441–1460, 2018.

[18] Y. Song and D. Scaramuzza, "Policy search for model predictive control with application to agile drone flight," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2114–2130, 2022.

[19] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural MPC: Deep learning model predictive control for quadrotors and agile robotic platforms," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2397–2404, 2023.

[20] K. Tsinganos, K. Chatzilygeroudis, D. Hadjivelichkov, T. Komninos, E. Dermatas, and D. Kanoulas, "Behavior policy learning: Learning multi-stage tasks via solution sketches and model-based controllers," *Frontiers in Robotics and AI*, vol. 9, 2022.

[21] Z. Xie, X. Da, B. Babich, A. Garg, and M. v. de Panne, "Glide: Generalizable quadrupedal locomotion in diverse environments with a centroidal model," in *Algorithmic Foundations of Robotics XV: Fifteenth Workshop on the Algorithmic Foundations of Robotics*. Springer, 2022.

[22] K. Chatzilygeroudis, A. Cully, V. Vassiliades, and J.-B. Mouret, "Quality-diversity optimization: a novel branch of stochastic optimization," in *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*. Springer, 2021, pp. 109–135.

[23] A. Cully and Y. Demiris, "Quality and Diversity Optimization: A Unifying Modular Framework," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 245–259, 4 2018.

[24] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge University Press, 2017.

[25] M. H. Raibert, *Legged robots that balance*. MIT press, 1986.

[26] K. O. Stanley, J. K. Pugh, and L. B. Soros, "Quality Diversity: a new Frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, 2016.

[27] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[28] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *IEEE conference on Computer Vision and Pattern Recognition*, 2015.

[29] A. Gaier, A. Asteroth, and J.-B. Mouret, "Data-efficient design exploration through surrogate-assisted illumination," *Evolutionary computation*, vol. 26, no. 3, pp. 381–410, 2018.

[30] R. Kaushik, P. Desreumaux, and J.-B. Mouret, "Adaptive prior selection for repertoire-based online adaptation in robotics," *Frontiers in Robotics and AI*, vol. 6, 2020.

[31] A. Cully and J.-B. Mouret, "Behavioral repertoire learning in robotics," in *Genetic and Evolutionary Computation Conference*, 2013.

[32] V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret, "Using centroidal Voronoi tessellations to scale up the multi-dimensional archive of phenotypic elites algorithm," *IEEE Transactions on Evolutionary Computation*, 2017.

[33] C. D. Bellicoso, F. Jenelten, P. Fankhauser, C. Gehring, J. Hwangbo, and M. Hutter, "Dynamic locomotion and whole-body control for quadrupedal robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[34] 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.

[35] S. Feng *et al.*, "Optimization based full body control for the atlas robot," in *International Conference on Humanoid Robots*, 2014.

[36] N. Hogan, "On the stability of manipulators performing contact tasks," *IEEE Journal on Robotics and Automation*, vol. 4, no. 6, pp. 677–686, 1988.

[37] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[38] B. Lim, M. Allard, L. Grillotti, and A. Cully, "Accelerated quality-diversity through massive parallelism," *Transactions on Machine Learning Research*, 2023.

[39] A. Macchietto, V. Zordan, and C. R. Shelton, "Momentum control for balance," in *ACM SIGGRAPH 2009 papers*, 2009, pp. 1–8.

[40] K. Chatzilygeroudis and J.-B. Mouret, "Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[41] C. Mastalli, W. Merkt, G. Xin, J. Shim, M. Mistry, I. Havoutis, and S. Vijayakumar, "Agile maneuvers in legged robots: a predictive control approach," *arXiv preprint arXiv:2203.07554*, 2022.

[42] A. W. Winkler, C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini, "Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.