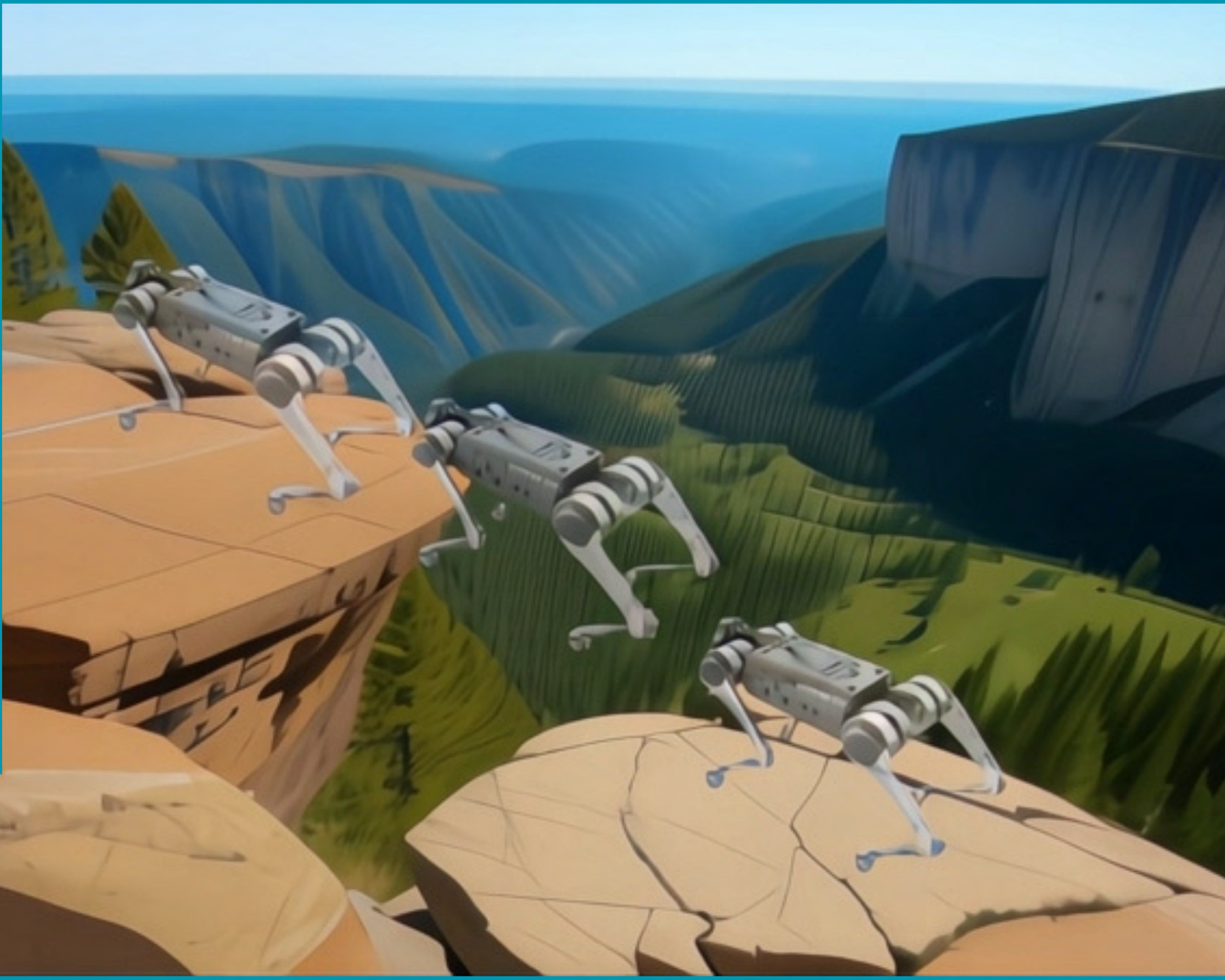
Terrain Adaptive Quadrupedal Jumping for Rigid and Articulated Soft Robots using Example Guided Reinforcement Learning

Georgios Apostolides

Master Thesis Report



Supervisors: Prof. C. Della Santina, Department of Cognitive Robotics, TU Delft
Dr. J. Ding, Department of Cognitive Robotics, TU Delft

Terrain Adaptive Quadrupedal Jumping for Rigid and Articulated Soft Robots using Example Guided Reinforcement Learning

Master Thesis

For the degree of Master of Science in Robotics at
Delft University of Technology

Georgios Apostolides

Student Number:	5377498
Supervisors:	Prof. Cosimo Della Santina Dr. Jiatao Ding
Thesis Committee:	Prof. Cosimo Della Santina Prof. Jens Kober Dr. Jiatao Ding
Submission Date:	August 5th 2024

Department of Cognitive Robotics,
Faculty of Mechanical Engineering,
Delft University of Technology

Acknowledgements

I would like to express my heartfelt gratitude to my supervisors, Dr. Cosimo Della Santina and Dr. Jiatao Ding. Their guidance, insightful feedback, and patience have been invaluable throughout this journey. Their expertise not only helped me shape the direction of my thesis but also inspired me to think more critically and creatively.

I am also deeply thankful to my friends who have been a constant source of support and laughter. Whether it was listening to my endless musings about my thesis over countless cups of coffee or offering words of encouragement, your presence made this journey much more enjoyable.

Lastly, I owe a special debt of gratitude to my beloved family and girlfriend. Your unwavering belief in me, along with your emotional support, has been a cornerstone throughout this process. Your confidence in my abilities has kept me motivated and focused, and for that, I am truly grateful.

Georgios Apostolides
Delft, August 5th, 2024

Terrain Adaptive Quadrupedal Jumping for Rigid and Articulated Soft Robots using Example Guided Reinforcement Learning

Georgios Apostolides

***Abstract*—The challenge of navigating uneven terrain is a critical obstacle in the advancement of robotic locomotion. Traditional quadrupedal locomotion methods, such as walking, are often insufficient for dynamic and complex environments. Agile skills like jumping are necessary and must be adaptable over uneven terrain. This study addresses this issue by developing a policy for executing jumps over uneven terrain using a single demonstration. Initially, the system learns to imitate a forward jump based on a single demonstration from a SLIP trajectory planner. It then generalises its jumping abilities to various distances in both forward and lateral directions. The study compares the performance of systems with and without parallel elasticity, demonstrating the energetic benefits of using elastic actuation for quadrupedal jumping. Results show that the system with parallel elastic actuation is 15.20% more energy-efficient and experiences a 15.79% reduction in peak power compared to the system without parallel elasticity. A policy trained using the proposed methodology successfully performs jumps of variable distances over uneven terrain with height perturbations of +/-4 cm using only proprioceptive information.**

I. Introduction

Robotics have started to make their way in our daily life and while doing so, they have the opportunity not only to improve our daily living but also to address challenges that seemed impossible before. Challenges such as search and rescue, space exploration, and environmental monitoring are only a few of such examples. All of the aforementioned applications require robotic platforms which are capable of navigating in difficult terrains. The problem of agile locomotion over highly uneven terrains is investigated in this work.

In nature, animals display remarkable agility, robustness, and efficiency in traversing unstructured environments, inspiring the creation of quadrupedal robots. These robots offer certain advantages over aerial robots, such as direct environment interactions and larger payload capacity. Furthermore, their discrete contact points with the ground make them superior to wheeled robots in navigating uneven terrains. Despite their potential, controlling legged robots in such settings remains a significant challenge. Animals owe much of their robustness and efficiency to elastic biological elements like tendons, which provide energetic advantages [1], [2]. This led engineers to investigate the use of elastic actuation in quadrupeds. While elastic actuation can offer significant gains in performance and efficiency, it can also make system's control more complex.

Various works have investigated locomotion over unstructured terrain using quadrupeds [3], [4]. However, the problem of performing locomotion in difficult environments is far from being solved. A lot of the relevant work has been restricted into stable quadrupedal walking over uneven terrain. However, in the wild, a robot may not be capable of traversing particular parts of the terrain using a simple walking or trotting motion. In the animal kingdom, quadrupedal animals such as goats possess various dynamic skills such as jumping, limping, and climbing. Jumping is a motion that can be performed or adapted over uneven terrain, however, how to achieve this, is still an open question.

Model Based approaches are methods which rely on the use of a mathematical model to represent the dynamics and kinematics of the quadruped. These models can either be a full order model [5] or a reduced order model [6], [7]. Such strategies have often been extremely useful in achieving stable and efficient locomotion over flat terrain [5] or even agile skills such as jumping [6] on flat terrain. However, when it comes to the unpredictability of uneven terrains, these methods tend to struggle. Controllers designed using model base approaches often find it difficult to adapt their movement to the variability of an uneven terrain. Furthermore, this type of methods can be computationally demanding for high order systems such as quadrupeds. Thus, often resulting in using a simplified version of the system dynamics [6], [7] which can be limiting. Even if a full order dynamical model is the choice, it is often difficult to be used in a real-time manner to avoid the unforeseen circumstances that can arise from an uneven terrain environment. To conclude, although model based approaches can form the basis of achieving simplified motions, they often struggle in providing the level of timeliness, agility and adaptability that is required to solve the problem of uneven terrain jumping.

In recent years, there has been a notable shift towards learning-based control methods, with Reinforcement Learning (RL) gaining particular prominence. RL has been proven effective in solving not only the problem of quadrupedal locomotion over flat terrain [8] but also in locomotion over uneven terrain [3]. In particular, a considerable amount of work has used RL in conjunction with a reference trajectory to enable quadruped robots to master dynamic motions such as jumps and back-flips [9], [10], [11]. This highlights the potential of RL in empowering quadruped robots not only to learn intricate and dynamic manoeuvres such as jumping, but also to dynamically adapt their behaviour in the face of uneven environments. Consequently, learning-based approaches, particularly RL, emerge as a promising way to solve the difficult problem of quadrupedal jumping from uneven terrains.

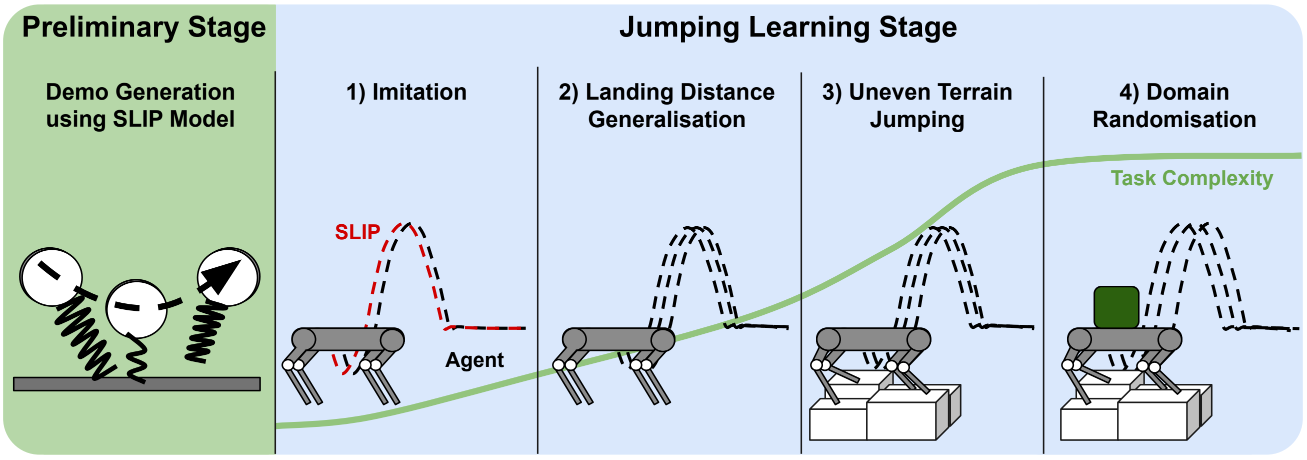


Fig. 1: Visualises a breakdown of the methodology to be followed in order to achieve uneven terrain jumping.

Depending exclusively on RL methods for addressing complex tasks like jumping, gives rise to considerable challenges. A significant hurdle emerges in the necessity for a well-designed and often elaborate reward function, requiring extensive trial and error to be tuned, especially as the number of parameters in the reward function increases [12], [13]. However, the introduction of a single demonstration in the form of a reference to the RL framework has demonstrated the potential to simplify the reward function, as evidenced by previous literature [9], [14]. Despite this improvement, methods relying on a single demonstration may encounter challenges in generalising beyond the demonstrated motion. Notably, existing studies that employ a single demonstration often confine their learning scope only to the demonstrated motion [8], [11]. Conversely, approaches incorporating multiple demonstrations [9], [15] aim to enhance generalization capabilities. However, the task of obtaining multiple demonstrations for complex tasks such as quadrupedal jumping may not always be feasible. Balancing the trade-off between the simplicity of reward function design and the ability to generalise effectively remains a challenge in the context of RL-based solutions for complex tasks like jumping.

We embrace the challenge of teaching a quadrupedal robot dynamic jumping motions like those demonstrated by quadrupedal mammals. We are particularly interested in scenarios where a quadruped robot performs jumps from and onto uneven surfaces, using a single demonstration throughout the entire learning process. Inspired by the use of curriculum learning, we address this challenge by dividing the learning process into four stages of increasing task complexity. After generating the reference trajectory, the agent's first task is to learn how to imitate the reference trajectory. Once the agent knows how to perform a jump of constant distance, we focus on generalising the landing distance and direction of the jump. In the third stage, the goal is to extend the agent's jumping capabilities into jumping from and onto uneven terrain. Finally, the learnt controller undergoes domain randomisation to ensure its robustness to the uncertainties of both the system and the environment. The primary contributions of the proposed approach include:

- **Single Demonstration Learning:** The proposed approach enables a quadruped robot to learn a jumping controller from a single demonstration, minimising the number of demonstrations required and simplifying the reward signal design.
- **Example Generalisation:** The learnt controller facilitates generalisation from a single example, encompassing various jumping distances and directions without the need of re-generating a reference trajectory.
- **Uneven Terrain Jumping:** Achieving the execution of jumps from and onto uneven terrains using a single demonstration.
- **Jumping using Parallel Elastically Actuation:** A comprehensive comparison between a rigid and an articulated soft quadruped in the task of jumping. Jumping over flat terrain with an elastically actuated quadruped demonstrates significant gains up to 15.20% in energy efficiency while jumping on average higher and with higher precision.

II. Related Work

A. Model Based Control Methods for Quadrupeds

Traditionally, model-based control approaches were the preferable control method for legged robots. Almost all model-based approaches can be decoupled into a planning and a tracking module. The planning module is used to generate a trajectory, while the tracking module is used to track the generated trajectory.

For the generation of a jumping trajectory several approaches have been proposed which often rely on Trajectory Optimisation (TO) [16], [17]. Some works have experimented in solving a trajectory using Mixed Integer Optimisation which incorporates the contact state into the optimisation problem in order to find an optimal trajectory [18], [19]. Although these approaches in a convex setting guarantee an optimal solution, they are usually computationally cumbersome and cannot be solved on the onboard computer. To alleviate this problem some other approaches had relied on generating a plan using predetermined contact schedules to aid in the convergence of optimisation [6], [17], [20]. However this usually limits the jump to a particular type, for example to a forward direction jump [16], [6].

For the tracking of the planned trajectory simplified controllers are often used. The traditional approach is to use a PD controller to track the planned trajectory [19], [6]. Other works have recently proposed the utilisation of a Model Predictive Control (MPC) for tracking the CoM trajectory and a PD controller for joint feedback [17]. In [21] the authors propose an Iterative Learning Control approach which adapts the gains of the controller to make it robust to uneven terrain perturbations during runtime. However, this approach still relies on the generation of a pre-planned trajectory using TO which is done offline and can take a considerable amount of time to be generated online.

To conclude, Model-Based approaches have been proposed for performing quadrupedal jumping. The approaches proposed often rely on a planned trajectory which is often generated offline due to its computational complexity and a simplified controller is used to track the trajectory. The planned trajectories are often generated for a specific distance and are not capable of being re-used for jumps of different distances. Very little work has focused on making these controllers robust to uneven terrain with the exception of [21] and [17] which demonstrate limited capability of performing agile motions in an uneven terrain scenario.

B. *Learning Based Control Methods for Quadrupeds*

Learning based control methods have recently received increased attention in the domain of quadruped robots. Here we explore the different works which are directly related to our methodology or that perform dynamic motions such as jumping.

A lot of the works utilise Imitation Learning as a methodology to teach a particular motion. However, these approaches usually limit themselves into learning the trajectory specified by the reference motion. In [8] and [11] the authors utilise an Example-Guided RL (ERL) methodology that utilises animal demonstrations to guide the learning process. The locomotion skills learnt in [8] are not generalised for example to different directions. Another work utilises animal videos and Deep RL to learn dynamic motions such as back-flips and walking however those motions are also as specified by the demonstration [11].

In contrast, other literature tries to generalise the motion learned through ERL. In [14] the authors investigate the use of a reference trajectory to learn a policy that performs agile tasks for animation characters. The characters are taught how to perform back-flips and how to walk but also tasks such as throwing a ball to a particular target. In the task of throwing a ball to a target, the trained policy is conditioned on the target location. Thus the agent is allowed to modify its trajectory given a particular target. Similarly in [10] the authors teach a biped robot how to jump using a single demonstration. However, they don't confine themselves to just imitating the demonstration, similar to our approach, they use policy goal conditioning to perform jumps of different distances and directions. A different approach is followed by Fuchioka et. al., where instead of using a single demonstration to teach different agile tasks, multiple demonstrations are utilised [9]. A demo trajectory is generated via trajectory optimisation and the trained policy is ensured to be generic enough so that the demonstrations can be tracked. However, this implies at first the need for multiple demonstrations, and the necessity of generating a reference every time a motion is to be altered, which may not be feasible in real time.

Other lines of work, focus on reducing the number of demonstrations as much as possible. Atanassov et. al. suggest a methodology that aids in obtaining a jumping policy without any demonstrations through the use of curriculum learning [13]. The utilisation of curriculum learning has been proven to be beneficial when no demonstrations are available. However, not utilising any demonstration, further complicates the design and tuning of the reward function. Similarly in [22], the authors propose a methodology that utilises rough partial demonstrations along with a Generative Network trained via RL to obtain dynamic jumping motions. A two-stage learning strategy is proposed in [12] which combines evolutionary strategies and DRL to alleviate the need for a pre-defined reference trajectory.

As for the task of performing agile jumping motions from and onto uneven surfaces there has not been considerable work. The primary work which focused on uneven terrain jumping is [15], which focuses on starting the robot from an uneven terrain and lands onto a flat surface. The proposed method relies on trajectory optimisation and utilises DRL as a feedback controller to correct the feed forward control actions of TO. The downside of the proposed approach is that a new trajectory must be optimised for by a TO module every time a different jumping distance is desired, which can be quite slow or infeasible in real time. In contrast, our proposed approach does not require the re-generation of a demonstration trajectory and relies only on a single demonstration. This allows for querying the controller for jumps of different distances in real time. Finally, [10] showcased jumping onto obstacles however the robot still jumps and lands from and onto a flat surface.

III. Framework Overview

As visualised in Fig. 1 the process of teaching an agent how to perform jumps over uneven terrain starts from a preliminary phase where we generate a demonstration trajectory followed by a learning phase of 4 individual stages, each with increased task complexity. The first phase of generating a reference trajectory is done by solving a numerical optimization problem for a specific jumping distance. An advantage of the discussed methodology is that we only use a single demo. Hence, although we utilise numerical optimisation for the demo generation, other data sources which are more scarce can also be used, such as animal demos. Once the trajectory is generated, the next step focuses on imitating the jumping motion using an Example-guided Reinforcement Learning (ERL) setup which will be elaborated in Sec. IV-B. As soon as the agent is capable of imitating the jump, the process proceeds to generalise the jump into different jumping distances by conditioning the policy onto a given jumping goal, more details are given in Sec. IV-C. Finally, the process introduces an uneven surface where the robot attempts to jump different distances across a platform of uneven heights more details will be given in Sec. IV-D. At this stage the robot is capable of jumping different distances from and onto an uneven terrain, the final stage involves randomizing different parameters of the robot itself and the environment to make the jumping more realistic, this is done through Domain Randomisation for which details are given in Sec. IV-E.

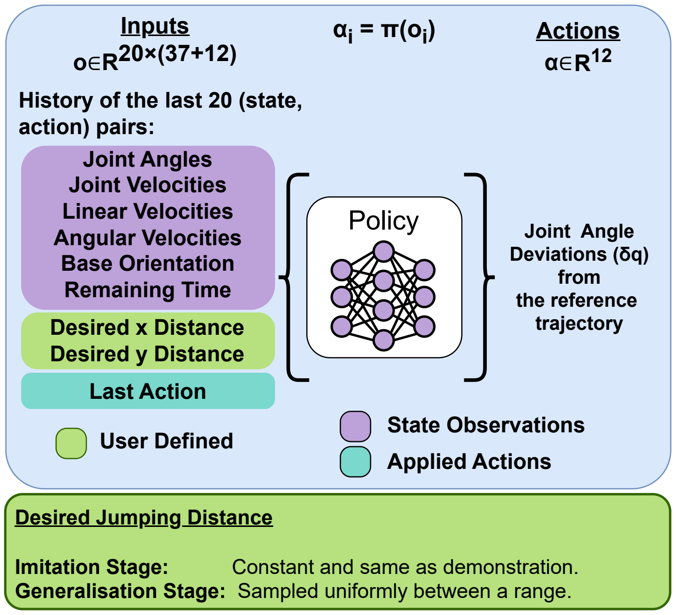


Fig. 2: Depicts the observation and action spaces of the policy.

On a high level, the policy trained, takes as input a historical sequence of observations and actions, and outputs a residual action relative to the reference trajectory. As depicted in Fig. 2, the observation history vector consists of: joint angles, joint velocities, Cartesian base velocities, angular velocities and the orientation in quaternion form. In addition, the policy receives as input a variable which ranges from 1 to 0 representing the remaining time before the episode finishes and a tuple of the desired jumping distance in the XY plane. Finally, the last filtered action applied by the agent is also given as input. During the imitation stage, the desired jumping distance is kept constant to the demo's jumping distance. During the generalisation stage, it is sampled uniformly from a specified range at the episode start. The history length is 20 time-steps. The outputted 12-dimensional action $(\boldsymbol{a}_i)$ is filtered by a low pass filter. The filtered action is then converted into a residual joint angle $(\boldsymbol{\delta q}_i)$ by linearly scaling the filtered action using the max and mean reference joint angles. Finally, the desired joint angle at time-step i, $({\boldsymbol{q}^{\text{des}}}_i)$, is given by:

$$ {\boldsymbol{q}^{\text{des}}}_i = \boldsymbol{\delta q}_i + {\boldsymbol{q}^{\text{ref}}}_i. \tag{1} $$

The above process is visualised in Fig. 3. The desired joint angle is tracked using a low level PD controller which tracks the desired joint angles at a frequency of a 1kHz while the desired joint angle is updated by the policy every 50Hz.

The agent utilised is a quadrupedal robot, specifically the Unitree GO1 robot is used to perform all the experiments. The simulations are run in a PyBullet simulation environment, which is operated at rate of 1 kHz. In addition, this work experiments with introducing elastic actuation in parallel to the motors of the quadruped robot. Parallel Elastic Actuation (PEA) has been suggested to offer advantages especially in the realm of dynamic motions. The springs introduced in parallel with the actuator, can be used as an elastic energy storage unit. The elastic elements act in a unidirectional fashion and thus the springs are compressed only when the robot squats down. The idea is that before performing the jumping motion the quadruped robot can squat down to compress the springs and later release the additional energy stored in the springs to jump more efficiently. In addition, during the landing phase the robot can potentially benefit from the springs which can be used to absorb the impact from the ground. Both systems with and without springs will be tested in the subsequent methodology. The spring constants used for each joint have been determined through trial and error and are the following: $K_{\text{Hip}} = 0$, $K_{\text{Thigh}} = 16$, $K_{\text{Calf}} = 10$.

IV. Methodology

In this section we discuss the methodology proposed for learning dynamic jumping motion. In Sec. IV-A an introduction to the notation used for formulating the RL problem is presented as well as the formulation of the optimisation problem for generating the reference trajectory. Details on the imitation stage are given in Sec. IV-B while information for the generalisation stage are given in Sec. IV-C. Finally, information regarding the uneven terrain training and the domain randomisation stage are given in Sec. IV-D and Sec. IV-E respectively. The details of each learning stage are also summarised in Appendix B Table VI.

A. Preliminaries

Reinforcement Learning Basics: The problem of learning dynamic jumping motions in this work is formulated as a Reinforcement Learning (RL) problem. The agent starts by observing its state in the environment $\mathbf{s}_i \in \mathcal{O}$ and takes an action $\mathbf{a}_i \in \mathcal{A}$, which results in the transition of the agent to a new state $\mathbf{s}_{i+1}$. The agent observes the new state $\mathbf{s}_{i+1} \in \mathcal{O}$ and uses it to take the next action $\mathbf{a}_{i+1}$. The goal in RL is to learn a policy $\pi\left(\mathbf{a}_i|\mathbf{s}_i\right)$ where given the state of the agent, the policy outputs the action the agent should take. To assess the quality of the actions taken by the agent, a reward function $\mathcal{R}_i$ is used, the better the actions taken by the agent the higher the reward value.

An analogy can be easily drawn between the above explanation of the RL paradigm and the problem of quadrupedal jumping. The agent can be represented by the robot, while the state can be observed from the onboard sensors. The action that causes the robot to interact with the environment can be seen as the torque exerted by the robot.

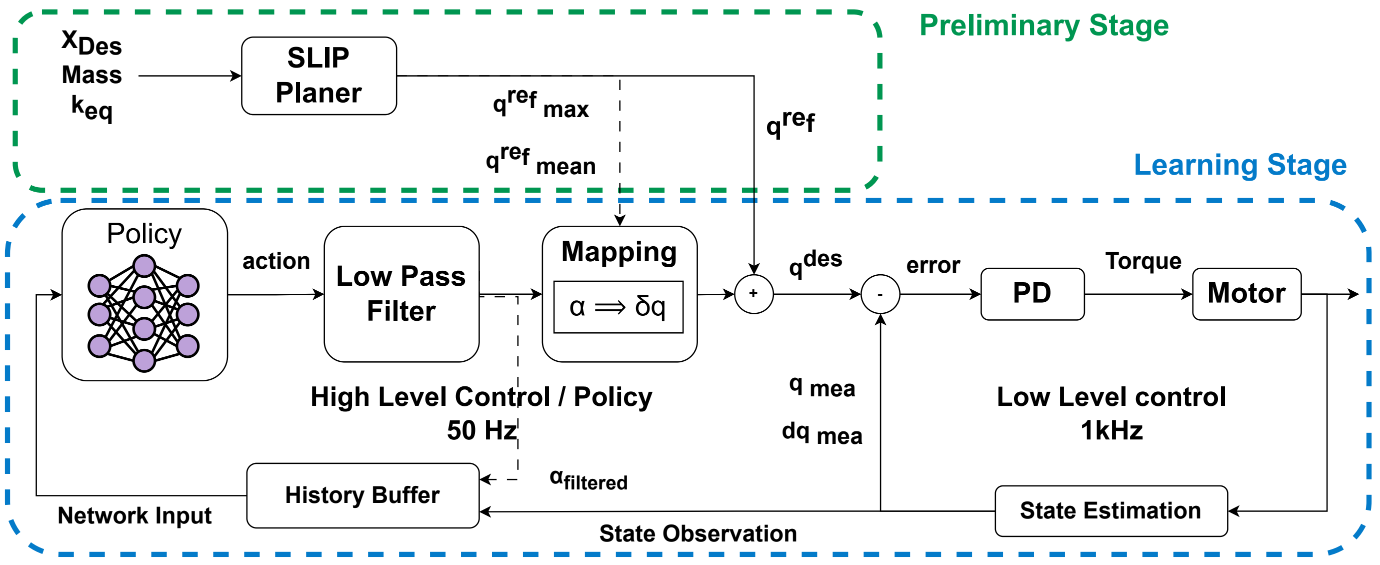


Fig. 3: Summarises the way at which the output of numerical optimisation is used by the policy trained using RL, as well as the process which a predicted action undergoes before being converted into a torque that is sent to the robot motors.

Specifically in our problem of quadrupedal jumping the policy is parameterised by a neural network which can be defined by a set of parameters $\boldsymbol{\theta}_\pi$. In addition, the network receives as input a history tuple $(\boldsymbol{h}_i)$ consisting of the last 20 state observations $(\boldsymbol{s}_i)$, actions $(\boldsymbol{a}_i)$ and desired landing distance $(\boldsymbol{p}_{\hat{\text{land}}})$. Thus the policy can be mathematically expressed as:

$$\pi(\boldsymbol{a}_i|\boldsymbol{h}_i) := \text{ANN}(\boldsymbol{h}_i; \boldsymbol{\theta}_\pi). \tag{2}$$

The objective in RL is to determine the parameters of the neural network which maximise the expected return (G_N):

$$\pi^*(\boldsymbol{a}_i|\boldsymbol{h}_i; \boldsymbol{\theta}_\pi) = \arg\max_{\boldsymbol{\theta}_\pi} \mathbb{E}\left[G_N|\boldsymbol{\theta}_\pi\right], \tag{3}$$

where the return is the cumulative sum of discounted rewards over time $G_N = \sum_{i=0}^{N} \gamma^k r(\boldsymbol{s}_i, \boldsymbol{a}_i)$ and $\gamma \in [0,1]$ is the discount factor which determines the importance of the instantaneous reward $r_i(\boldsymbol{s}_i, \boldsymbol{a}_i)$. A common approach for solving a problem in continuous action space such as the one above is policy gradient methods [23]. This family of methods utilise gradient descent to optimise the above objective function. Proximal Policy Optimisation (PPO) is utilised to train all the policies in this work due to its robustness to hyperparameter selection and its simplicity [23]. Details regarding the RL algorithm hyper-parameters can be found in Appendix B.

Reference Generation: As highlighted earlier, references from different sources can be used to teach the agent how to jump. However, here we use numerical optimisation due to its availability from prior work and showing the potential to perform jumps using a model based controller [21], [24]. Additionally, optimisation based examples allow for a comprehensive description of the agent's state. In this section the method in which the jumping reference is generated is explained. The jumping reference is used primarily in the imitation learning stage to guide the agent in learning the initial jumping trajectory. The trajectory is generated by solving a numerical optimisation problem.

Generating a reference for a system using numerical optimisation requires a dynamical model to describe the motion of the quadruped. The entire jumping motion can be decomposed into two phases: the stance phase, where the robot legs touch the ground, and the flight phase, where the robot is in the air. The Spring-Loaded Inverted Pendulum (SLIP) was used to describe the aforementioned motion due to its simplicity but also due to its common use for dynamic motions such as running and jumping. The SLIP model consists of a mass attached to a mass-less spring. During the stance phase a spring loaded mass is used to model the dynamics, while during the flight phase the dynamics of a projectile motion are used. Normally, this is enough to model the dynamics of a passive dynamical system that jumps. However, the above model does not incorporate the contribution of an actuator on the system dynamics. To model the force of the actuators, a virtual force is applied on the point mass. The virtual force is proportional to the additional acceleration provided by the actuators to the point mass that is denoted by $(\mathbf{u})$. The idea of adding a force to the SLIP model had also been previously proposed [7]. However, here the force is intentionally decomposed from the spring force of the SLIP model to be used as an optimisation variable for trajectory generation.

The formulation of the optimisation problem relies on the assumption that there is a fixed number of steps for the stance phase (N_s) and a fixed number of steps for the flight phase (N_f). However, the time step $(\boldsymbol{\delta t})$ of stance and flight phase is given as an optimisation variable to provide some flexibility. To ensure that the problem still remains tractable an initial guess of the $\boldsymbol{\delta t} = [\delta t_{\text{stance}}, \delta t_{\text{flight}}]^T$ and its bounded within a range (eq. 4g) to ensure a trajectory that can be tracked. The optimisation problem is formulated as:

$$\begin{aligned}
&\min_{\mathbf{p},\boldsymbol{\delta t},\mathbf{u}} \quad J_{\text{lift}} + J_{\text{flight}} + J_{\text{stance}} && \text{Cost Function} && \text{(4a)}\\
&\text{s.t.} \quad \mathbf{p}(0) = \mathbf{p_0} && \text{Initial Conditions} && \text{(4b)}\\
&\dot{\mathbf{p}}(0) = \mathbf{v}_0 && && \text{(4c)}\\
&\mathbf{p}(N) = \mathbf{p}_f && \text{Terminal Conditions} && \text{(4d)}\\
&\dot{\mathbf{p}}(N) = \mathbf{0} && && \text{(4e)}\\
&p^z(N_s - 1) = p^z_{\text{take off}} && \text{Take Off Condition} && \text{(4f)}\\
&\boldsymbol{\delta t}_{\text{min}} \leq \boldsymbol{\delta t} \leq \boldsymbol{\delta t}_{\text{max}} && \text{Time Step Constrain} && \text{(4g)}\\
&\forall i \in [0, 1, \ldots, N_s + N_f] && && \\
&\mathbf{p}_{i+1} = \text{Taylor}(\mathbf{p}_i, \dot{\mathbf{p}}_i, \ddot{\mathbf{p}}_i, \delta t) && \text{Dynamics} && \text{(4h)}\\
&\forall i < N_s : && \textit{\textbf{Stance Phase}} && \\
&\mathbf{p}_i \in \text{conv}\,[\boldsymbol{p_{feet}}] && \text{Support Polygon} && \text{(4i)}\\
&\ddot{\mathbf{p}}_i = \frac{\mathbf{F_s}(\mathbf{p}_i)}{m} + \mathbf{u} + \mathbf{g} && \text{Stance Dynamics} && \text{(4j)}\\
&m^{-1}(F_s)^z + u^z \geq 0 && \text{Gravity Constraint} && \text{(4k)}\\
&\forall i \geq N_s : && \textit{\textbf{Flight Phase}} && \\
&\ddot{\mathbf{p}}_i = \mathbf{g}, && \text{Flight Dynamics} && \text{(4l)}
\end{aligned}$$

where $\boldsymbol{g} = [0, -9.81]^T$ is the gravitational acceleration vector and $\boldsymbol{p}_i = [p^x_i, p^z_i]^T$ represents the position of the system in the Cartesian XZ plane at time step i. A detailed description and explanation behind the imposed constraints, cost functions as well as the used parameters can be found in Appendix A.

Following the generation of the two dimensional reference trajectory, a low pass filter is used to be smoothed. The reference is interpolated to the frequency of the RL policy which was decided to be 50Hz. Additionally, using the generated CoM trajectory a desired joint angle reference trajectory is generated by solving the inverse kinematics (IK) of the quadruped robot. Furthermore, the trajectory is extended to include a landing phase where the joint angles are kept to the humming pose and the Cartesian position is kept to be the same. During the entire trajectory the orientation and the lateral position are both kept to be zero.

The solver used for solving the optimisation problem is the Ipopt solver from the CasADi package. The solution takes approximately from 0.8 to 1s to be generated for a desired distance of 0.4m. The entire process of solving the optimisation problem, smoothing, but also generating the desired joint angles using IK takes around, 10s. Although the reference generation is done only once and offline during the proposed methodology it's worth mentioning that it may take a considerable amount of time depending on the desirable jumping distance to generate the entire reference trajectory and thus would be difficult to generate the reference trajectory online.

B. Imitation Learning Stage

In this stage the aim is to teach the agent how to perform a stable jumping motion of a fixed length. For this reason the desired landing distance which is passed as an input to the policy is kept fixed to the landing distance of the demonstration.

Learning a jumping motion, even from a reference trajectory, can be challenging, as the states mid-air are difficult to be explored by the agent adequately. As pointed out in [14] starting from a fixed initial state can be challenging when learning dynamic movements. However, in imitation learning, changing the initial state can be a strong prior in enabling the agent to explore states that lead to high reward. Thus similarly to [14] we utilise Reference State Initialisation (RSI) where the agent begins the episode from a randomly sampled state along the desired reference trajectory. This is done every 5th episode in order to collect also experiences where the agent starts from the initial state. To account for changes in the reward value when the agent starts from a different initial state we normalise the reward by the number of maximum possible actions that can be taken within the episode length. Hence assuming that in cases where the agent starts from the n^{th} time-step we assume that for the $n-1$ previous states the agent has imitated perfectly the trajectory.

During the Imitation stage the reward is described by the following reward function:

$$\mathcal{R} = \sum_{i=0}^{N} r^{\text{im}}_i + r^{\text{land}} - p^{\text{smooth}} - p^{\text{land}} + r^{\text{survival}}. \tag{5}$$

The above reward includes an imitation reward (r^{im}) and a survival reward (r^{survival}) given to the agent after reaching the last step of the episode. In addition, a reward is given for landing close to the desired landing position (r^{land}), at the end of the episode. Finally, penalties are also added for smoothing the agent's motion (p^{smooth}) but also ensuring stability after landing (p^{land}).

The imitation reward (r^{im}) encourages the agent to stay close to the demonstration's trajectory and its calculated at each time-step as follows:

$$r^{\text{im}}_i = w^{\text{q}} f(k_q, \hat{\boldsymbol{q}}_i, \boldsymbol{q}_i) + w^{\text{p}} f(k_p, \hat{\boldsymbol{p}}_i, \boldsymbol{p}_i) + w^{\text{o}} f(k_o, \hat{\boldsymbol{o}}_i, \boldsymbol{o}_i) \tag{6}$$

, where the exponential kernel function is represented by: $f(k, \hat{\boldsymbol{x}}_i, \boldsymbol{x}_i) = \exp(-k \left\|\hat{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_i\right\|_2)$. Here, $\hat{\boldsymbol{x}}$ represents the demonstration state while $\boldsymbol{x}$ represents the state of the agent. An imitation reward is given for the base position $\boldsymbol{p}_i$, the joint position $\boldsymbol{q}_i$ and the agents orientation $\boldsymbol{o}_i$. For the joint angles and the base position the L2 norm is used to compute the error between the demonstration and the agent's state while for the orientation the angle of rotation between the two quaternions is used. A weighted sum is used to give the total imitation reward at each time-step.

In addition to the imitation reward a landing reward (r^{land}) is given to reward the agent for landing close to the desired landing position which is kept constant for the stage of imitation as the desired distance of the reference trajectory. The landing position reward is given at the end of the episode and is only provided if the episode has not been terminated. The reward is computed using the following expression:

$$r^{\text{land}} = w_{\text{land}} \exp(-k_{\text{land}} \left\|\boldsymbol{p}_{\textbf{land}} - \boldsymbol{p}_{\hat{\textbf{land}}}\right\|_2), \tag{7}$$

where the desired landing position coming from the demo is denoted by ($\boldsymbol{p}_{\hat{\textbf{land}}}$) whereas the landing position of the agent is denoted as ($\boldsymbol{p}_{\textbf{land}}$). To judge whether the robot is in a flight phase the agent must be in a state where non of the legs are touching the ground and the CoM height is above the maximum stance height. The agent is considered to have landed if it was in a flight phase and any of the legs touches the ground. Once this change in phase from flight phase to stance phase occurs the base position is recorded as the landed position.

To ensure a smooth jumping trajectory a penalty is imposed which penalises the norm of joint velocities ($\dot{\boldsymbol{q}}$) and joint accelerations ($\ddot{\boldsymbol{q}}$):

$$p^{\text{smooth}} = w^{\text{pen}}_{\text{ddq}} \sum_{i=i_{\text{ air}}}^{N} \|\ddot{\boldsymbol{q}}_i\|_1 + w^{\text{pen}}_{\text{dq}} \sum_{i=i_{\text{ air}}}^{N} \|\dot{\boldsymbol{q}}_i\|_1 \,. \tag{8}$$

In addition, to make sure the agent lands stably, a penalty is applied on the XY velocity ($\boldsymbol{v^{xy}}$) of the base once the robot has landed. Mathematically, the landing penalty can be expressed as:

$$p^{\text{land}} = w^{\text{pen}}_{\text{vel}} \sum_{i=i_{\text{ land}}}^{N} \|\boldsymbol{v^{xy}}_i\|_1 \,. \tag{9}$$

Provided that the agent has reached the final step of the episode and an early termination has not been triggered, then the agent will receive a positive reward which rewards its survival during the episode. In contrast, if an early termination is triggered, the agent receives a negative reward to penalise its behavior. Hence, the following equation can be used to express the above scenarios:

$$r^{\text{survival}} = \begin{cases} -0.1 & i_{\text{ termination}} < N \\ +0.1 & i_{\text{ termination}} = N. \end{cases} \tag{10}$$

During the imitation stage, an episode is terminated early if there is an illegal contact with the ground or if the robot has fallen over. A contact is judged as illegal, if any other part of the robot apart from its feet are in contact with the ground. The agent is said to has fallen, if the position vector of the base in the z-direction has an angle bigger than 30 degrees from the global z-direction vector. Additionally, the episode is terminated early during the imitation stage if the agent has a feet contact state which does not match the contact state of the demonstration for more than 120ms.

C. Landing Distance Generalisation Stage

For the generalization stage the aim is to teach the robot how to jump different distances using the baseline acquired during the Imitation Stage. At this stage the agent is already capable of jumping a fixed distance. Thus, in this stage the agent always starts from its initial position and there is no need for RSI. To teach the agent to perform jumps of varying distances, the target landing distance ($\boldsymbol{p}_{\hat{\text{land}}}$) is provided as an input to the policy during training and is varied during the generalisation stage. The forward desired jumping distance is uniformly sampled using the following distribution $p_{\hat{\text{land}}}{}^{x} \sim \text{U}\,(0.0, 1.0)$ m while the lateral desired distance is also sampled uniformly using $p_{\hat{\text{land}}}{}^{y} \sim \text{U}\,(-0.3, 0.3)$ m.

The reward function proposed during the imitation stage is modified to allow jumps of different distances. In particular the imitation reward (eq. 6) is modified by setting the weight for tracking the base position to zero (w^p), thus only the joint angles and orientation imitation is rewarded. The penalties for landing velocity and smoothing as well as the survival reward are kept (eq. 8 - 10). Similar to the imitation stage, a reward is given for landing close to the desired landing position (eq. 7). Except now the desired landing position, ($\hat{\boldsymbol{p}}_{\text{land}}$) is identical to the varying input of the neural network. In this way the policy can learn to associate a desired jumping distance with the corresponding input.

An additional reward is given for matching the desired base velocity in the xy direction. The desired velocity of the base is approximated using the desired landing displacement and the jump duration: $\hat{\boldsymbol{v}}_{\text{xy}} = \Delta \boldsymbol{p}_{\hat{\text{land}}}/T$. Once the robot has landed the desired velocity tracked is set to be 0 m/s. The desired velocity is tracked using an exponential kernel:

$$r^{vel}_i = w^v f(k_v, \hat{\boldsymbol{v}}_{\text{xy}}, \boldsymbol{v}_{\text{xy}}). \tag{11}$$

The conditions during which an episode is terminated are different in the Generalisation phase of training. When jumping longer distances the robot would spend more time on the air compared to the demonstration given. For this reason, in contrast to the imitation stage, the episode is no longer terminated if the agents contact state does not match that of the demo. Instead the agent is left to determine through trial and error the optimal contact sequence provided the landing distance matches the desired one. The episode is still terminated if the local z-direction of the base has an angle bigger than 30 degrees from the global z-direction. Furthermore, an additional termination is introduced where the episode terminates if the landing error is bigger than a threshold. The termination threshold is linearly varied based on the desired landing position, thus the condition for termination can be expressed as:

$$|\hat{\boldsymbol{p}}_{\text{land}} - \boldsymbol{p}_{\text{land}}| > \alpha \, |\boldsymbol{p}_{\hat{\text{land}}}| + \beta, \tag{12}$$

where α is a hyperparameter of how much the allowable limits grow as the desired jumping distance increases and β determines a constant termination limit when the desired jumping distance is 0m. Intuitively the above expression enforces jumps which are shorter in jumping distance to be more precise as the landing error should be smaller, while jumps with bigger desired distance are given a higher allowance for landing error.

A detailed description of the reward hyper-parameters for the generalisation stage can be found in Appendix B at Table IV

D. Uneven Terrain Stage

In this part, we focus on the methodology used to teach the agent how to execute jumps across uneven terrain, along with insights of how the uneven terrain is generated. This stage builds upon the policies trained during the generalization phase, where the robot has already gained proficiency in executing jumps of various distances on flat terrain.

The uneven terrain setup comprises boxes of varying heights, presenting a challenge to the robot's stability. Height

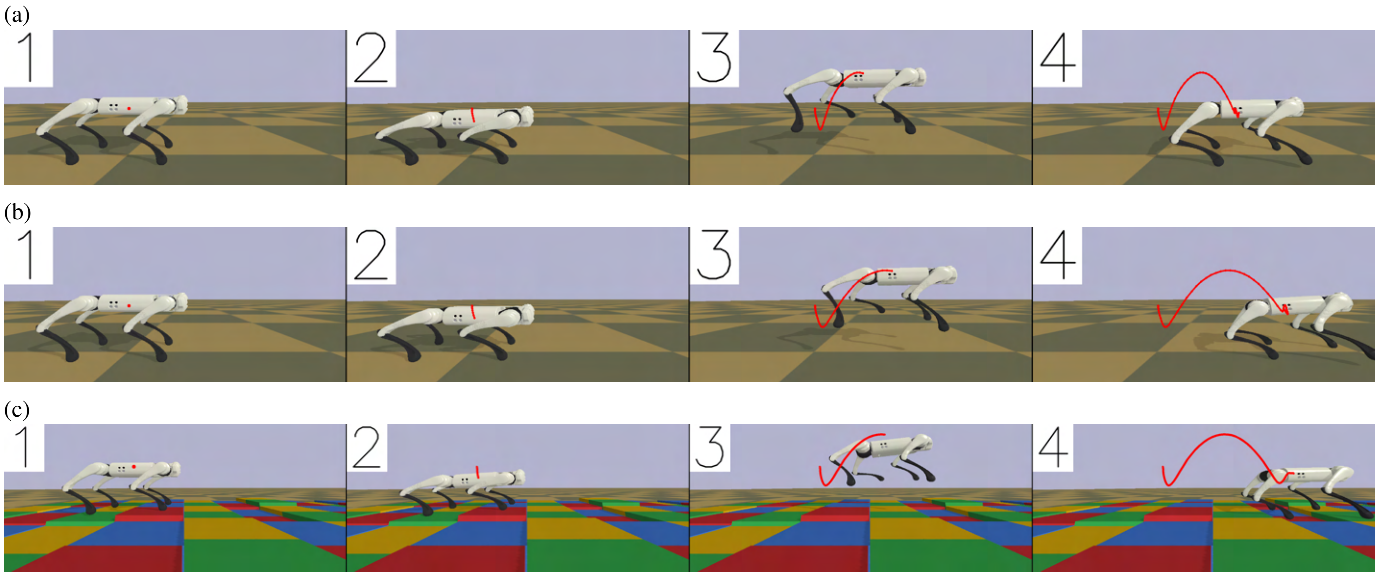


Fig. 4: Depicts a time-lapse of a jump where the robot is prompted to jump: (a) 40 cm and (b) 75 cm over flat ground. Finally, (c) shows a jump of 75 cm over uneven terrain.

perturbations are strategically applied between the robot's front and rear legs during the initial state to prevent lateral falls. However, the robot can still experience perturbations between its left and right side legs during landing. At the beginning of each episode, the robot is settled onto these varied-height boxes by solving inverse kinematics (IK). Solving IK allows to determine the joint angles that conform with the terrain's topology at the initial state. Similarly to the generalization stage, the robot is prompted to jump different distances, both in the forward and lateral direction.

The terrain is instantiated using boxes with a height of 15cm. At the beginning of each episode a height perturbation is sampled from a uniform distribution $U(-\epsilon, +\epsilon)$ m and is added to the constant boxes height to emulate an uneven terrain. Furthermore, during training, there is a 20% probability that no height perturbations are sampled and thus the terrain remains flat. This is done to avoid over-fitting the policy to jumps from uneven terrain. In addition we experiment with different maximum height perturbation boundaries (ϵ). We investigate how increasing the unevenness of the terrain affects the trained controller and if the controller trained on uneven terrain can still perform over flat ground.

The reward which was optimised for flat ground is also used to optimise for uneven terrain. This includes all the weights and length scales of the reward function as well as the termination limit parameters (α and β) that were specified in the previous stage. The reinforcement learning hyper-parameters are also kept constant.

E. Domain Randomisation Stage

An effective strategy for bridging the simulation to reality gap is Domain Randomisation (DR), as shown in other works [25], [12], [13]. The central idea behind DR is to vary parameters within both the environment and the robotic system itself, ensuring that the learned policy remains robust against uncertainties experienced in a real-world setting.

Several parameters are randomised to make the model of the agent more realistic. These include randomisation of the mass of the agent and its distribution across the robot. Furthermore, the center of mass of the robot is perturbed by attaching an offset mass on the robot's base at different locations. The motor strength as well as a frictional torque are used to make the motor model more realistic. Randomising the initial state of the agent contributes to a more robust jumping policy in response to offsets from the initial joint configuration. Contact dynamics are also diversified by randomly sampling friction coefficients for the feet and ground, along with their corresponding restitution coefficients. Finally, for the system which utilises elastic actuation, the spring parameters are randomised. This includes the spring stiffness, the spring damping, as well as the springs rest angles. Randomisation of the spring parameters is utilised to make the policy robust to a spectrum of spring parameters and thus account for inaccuracies in the parameters of the springs. More details on the specific ranges used can be found in Appendix C, Table V. By systematically varying the aforementioned parameters, the policy should become robust enough to effectively handle environments that closely resemble reality.

V. Results and Discussion

A. Imitation Stage

Tracking Performance: In the stage of imitation the primary concern was to make the robot imitate as closely as possible the reference generated from TO, thus allowing the robot to perform a jump of fixed distance. As illustrated in Fig. 7, both the robot with stiff and elastic actuation closely follow the CoM trajectory planned. During the landing phase of the trajectory, we observe some oscillations in the CoM position due to the impact. The robot deviates from the SLIP reference (Fig. 6) thus after landing it tries to recover back to its humming pose. As a result, while the robot tries to

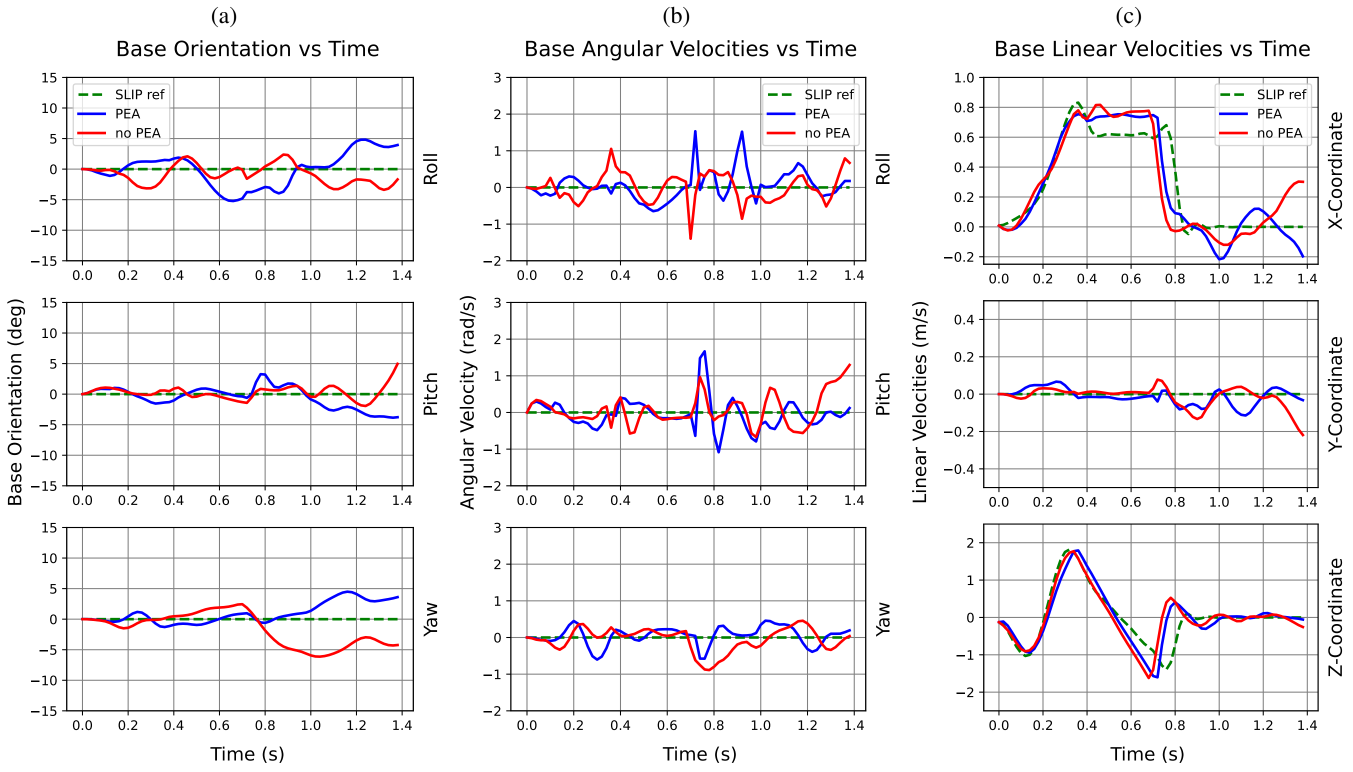


Fig. 5: (a) Base orientation, (b) angular velocity and (c) linear velocity trajectories of the quadruped while performing a jump during the imitation stage both with and without PEA.

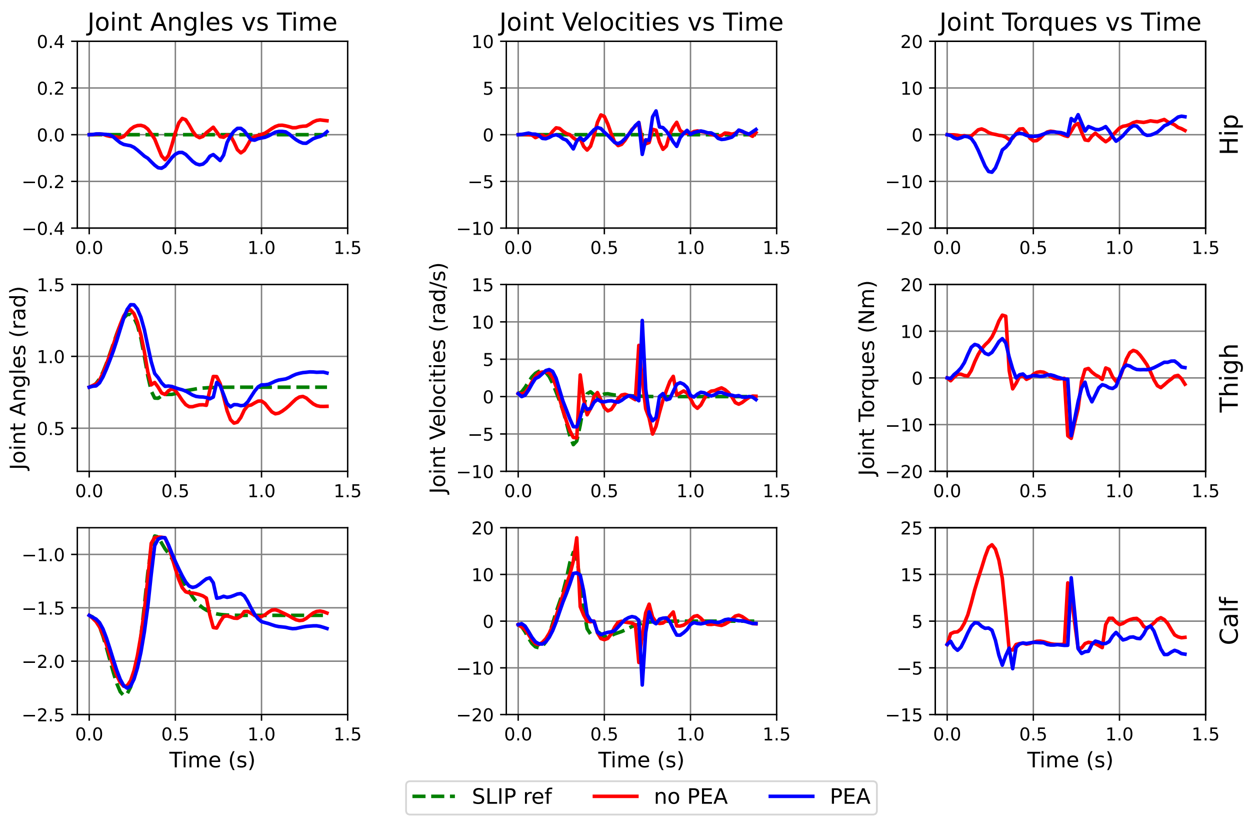


Fig. 6: Joint angles, velocities and torques of a single leg while performing a jump during the imitation stage both with and without PEA.

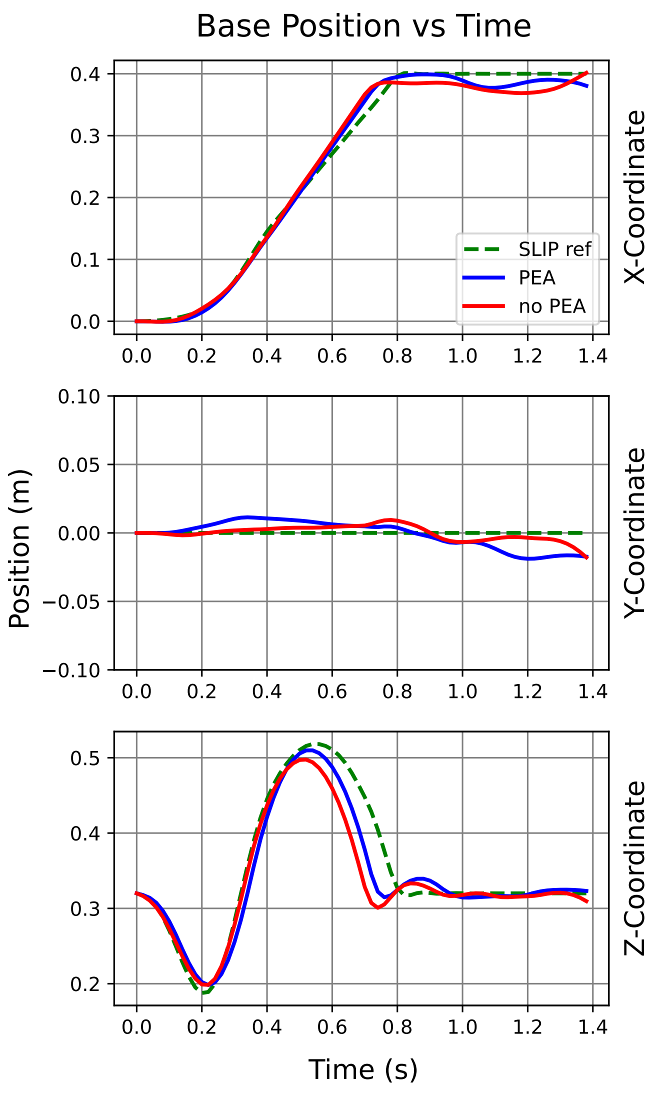


Fig. 7: Base position trajectories during the imitation stage. A video of the robot performing the jump during imitation can be found for the system without PEA [here] and for the system with PEA [here].

return to the humming pose, it causes slight oscillation to the forward linear velocity as visualised in Fig. 5c. The reference trajectory specifies that all Euler angles should be kept at zero, as visualised in Fig. 5a and 5b the Euler angles are all kept within +/- 5 deg and the robot base experiences small angular velocities. Another interesting observation from the Joint torques in Fig. 6 is that the stiff actuation system experiences its peak torque values while pushing the robot from the ground around 0.2 s, while the PEA system does not demand such large amounts of torques during the stance phase. This implies that the actuators of the robot work together during the stance face and the elastic energy is used during the lift off phase to make the robot jump.

Peak Height & Desired Landing Distance: Both policies for the robot with and without springs were trained by optimising the same reward function and by having the same hyperparameters. This allows us to compare the performance of the PEA system and the one without. An interesting observation from Fig. 7 is that the robot with PEA has peak height marginally higher than the robot without PEA. However, neither of the two reaches the peak height of the reference trajectory. In addition, the robot with PEA seems to be jumping closer to the desired jumping distance compared to the non-elastically actuated system.

Peak Power & Total Energy: Lower peak power is often associated with better energy efficiency and extended battery life. Hence, it can be an important characteristic used to differentiate between two trajectories followed by a system. The instantaneous absolute mechanical power can be computed using the product of joint velocities and the joint torques for each of the two systems. This can be used to compute the peak power of each trajectory. The mean power computed from 30 jump trajectories along with 95% confidence intervals is visualised in Fig. 8. Interestingly, the peak power occurs for both systems during the landing impact. In addition, it is evident from the plot that the system with springs experiences less power during stance phase showing that the actuators work together with the springs. The utilization of PEA results in a 27.64% decrease in mean peak power. Using a 95% confidence interval, we can estimate that PEA improves peak power by 11.38% in the worst-case scenario and up to 30.09% in the best-case scenario.

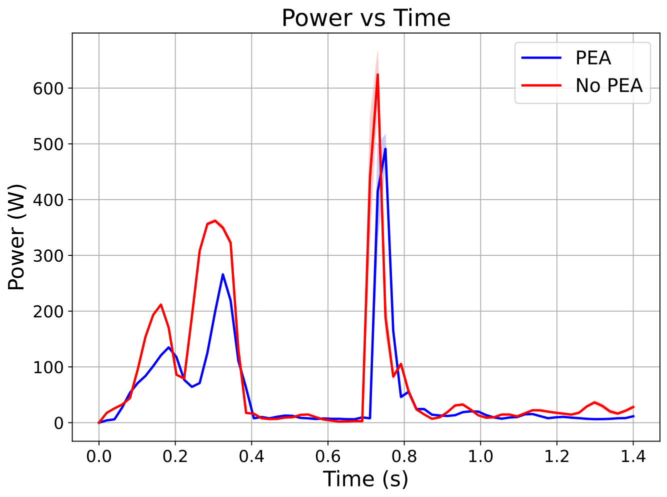


Fig. 8: Comparison of power over time for the system with stiff and elastic actuation. The plot shows the instantaneous power averaged over 30 jumps for each system, with the upper and lower bounds indicating the 95% confidence intervals for power variability.

Integrating the power over time allows the computation of the energy for each system. The robot with elastic actuation uses a mean total energy of 71.4 J while the system with stiff actuation uses 106.4 J. Consequently, PEA usage leads to a 32.74% improvement of the mean total energy. Utilising a 95% confidence interval, the mean total energy can be decreased from 30.69% to 34.73% for the best and worst-case scenarios, respectively, when using elastic actuation.

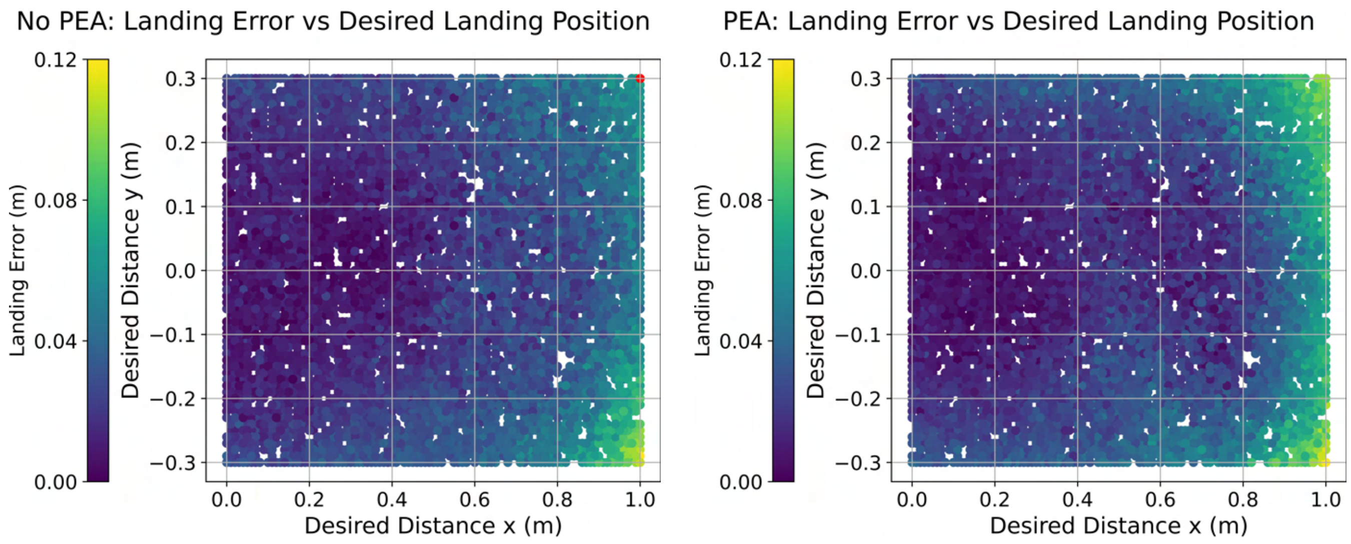


Fig. 9: Illustrates a landing error comparison between the stiff and elastically actuated system for jumps in the xy plane.

B. *Generalisation Stage*

The aim at this part was to use the policy learned through imitation as a starting point and enable the agent to achieve jumps of different distances. In the following section the policies both for the system with parallel elasticity and without are assessed in terms of landing distance, peak height, energy efficiency and peak power. A video of the system without PEA performing forward jumps of various distances can be found in [here] while for the system with PEA can be found [here]. Videos of diagonal and lateral jumps for the system without PEA can be watched [here] while for the system with elastic actuation can be viewed [here].

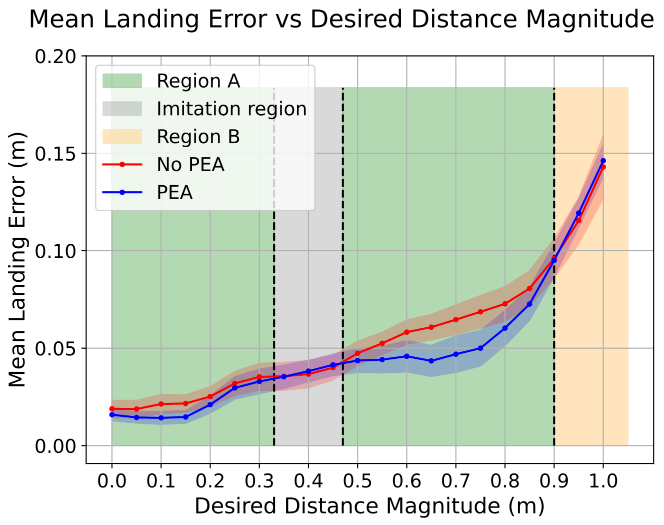


Fig. 10: Depicts the distributions of mean landing error against desired distance magnitude for the two systems.

Tracking Performance: As depicted in Figure 9, both systems are capable at this stage to perform jumps in the forward but also in the lateral direction after given only a single demonstration which demonstrated how to perform a forward jump. As the desired distances in the x and y directions increase, there is a corresponding rise in landing error for both systems. Notably, jumps approaching the maximum desired distance of 1m exhibit a mean landing error of 12cm.

The jumps demonstrated in Fig. 9 where discretised into bins with an interval of 5 cm and the average landing error was computed for each system. The distributions of both systems for the mean landing error against the desired landing distance magnitude are depicted in Fig. 10. As expected Fig. 9 shows the mean landing error does increase with the rise of desired distance magnitude. Another interesting observation is that the mean landing error remains almost identical for both systems in the imitation region shown in grey at Fig. 10. In region A, the system with PEA seems to have marginally lower mean landing error. In contrast, in region B the system with PEA has a marginally higher mean landing error. This leads us to speculate that the elastic actuation system tends to become more susceptible to instability during longer jumps, contributing to increased landing errors.

Peak Height: In this section we discuss the difference of peak height between the policy with and without springs during the generalisation stage. As illustrated in the map at Fig. 11 the system with elastic actuation seems to perform jumps with higher peak height compared to the system without parallel elastic actuation. The system with PEA during the 10 000 jumps performed has an overall mean peak height of 52 cm while the system with stiff actuation has a peak height of 57 cm. Thus, the system with PEA jumps on average 8.77% higher than the one without.

The desired distance magnitude of the jumps performed by each system was discretised into bins of 5cm to evaluate whether there is difference between the peak height variation of the two systems. The peak height averaged over a range of 5 cm of desired magnitude is plotted for each system in Fig. 12. The aforementioned figure illustrates that the system with stiff actuation decreases its peak height for larger distances. In contrast, the robot with parallel elasticity seems to maintain its peak height constant irrespective of the increase in the desired distance magnitude.

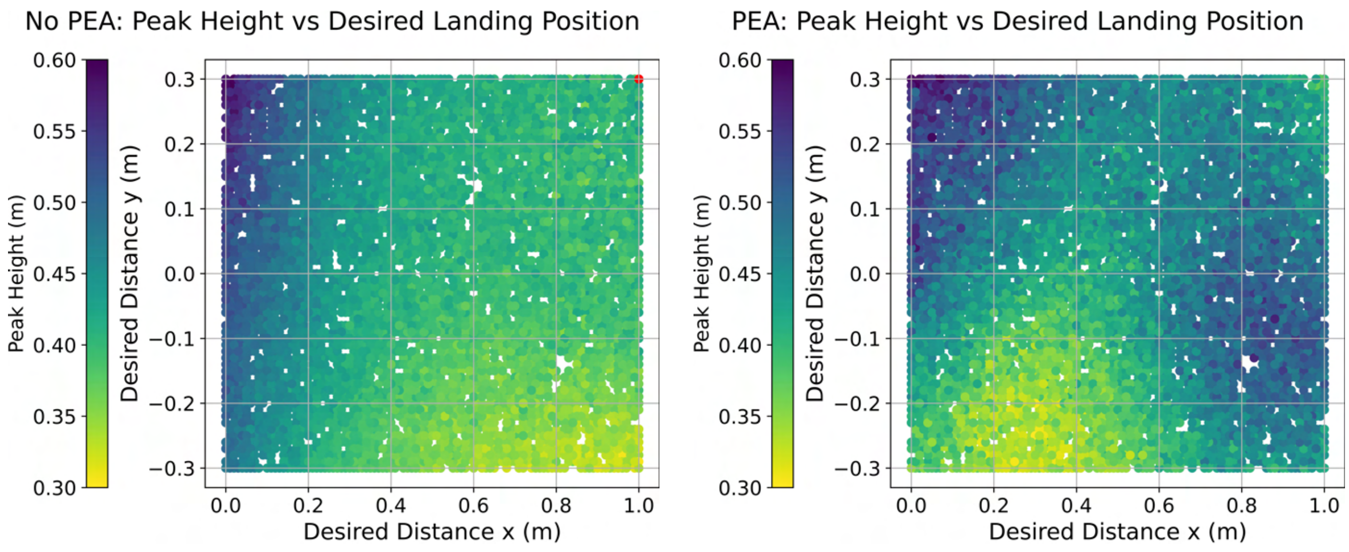


Fig. 11: Depicts the peak height variation between the stiff and elastically actuated system for jumps in the xy plane.

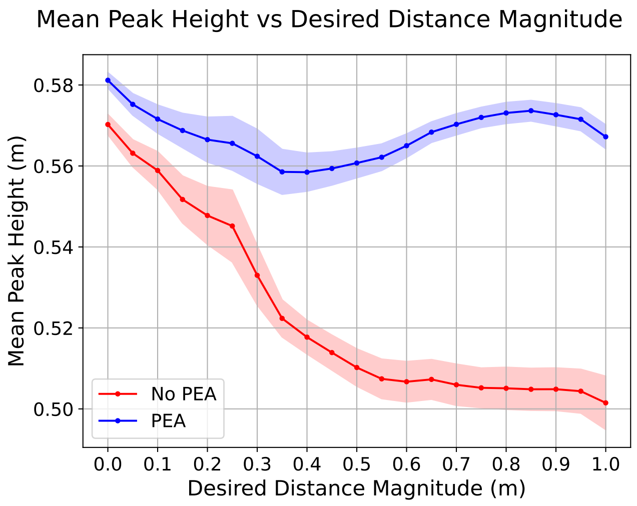


Fig. 12: Visualises the variation of peak height over desired distance magnitude for both systems. Shaded region visualises the upper and lower bound with a confidence interval of 95%.

Peak Power & Total Energy: Both systems where assessed in terms of their peak power and their total energy as a way of assessing the energy efficiency of the jumps performed.

The total energy expended for a jump against the desired jumping distance in x and y is visualised in Fig. 14 for the two systems. The figures demonstrate that for the system with PEA the expenditure of energy is less for medium to low jumps desired distance jumps. In comparison, it seems like the system without parallel elasticity consumes more energy compared to the system with elastic actuation. In addition, the energy map demonstrates that energy peaks for the robot with stiff actuation occur not only for large jumping distances but also for jumps which are performed in the lateral direction and upward direction. In contrast, the system with PEA seems to be more efficient for medium to short range jumps.

To clearly visualise the difference of energy between the two systems the total energy is plot against the desired distance

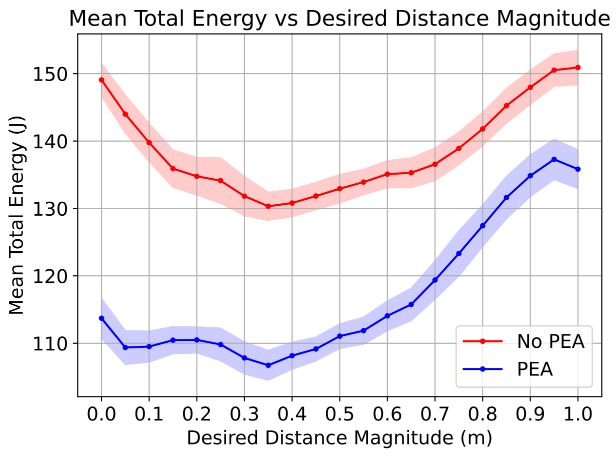
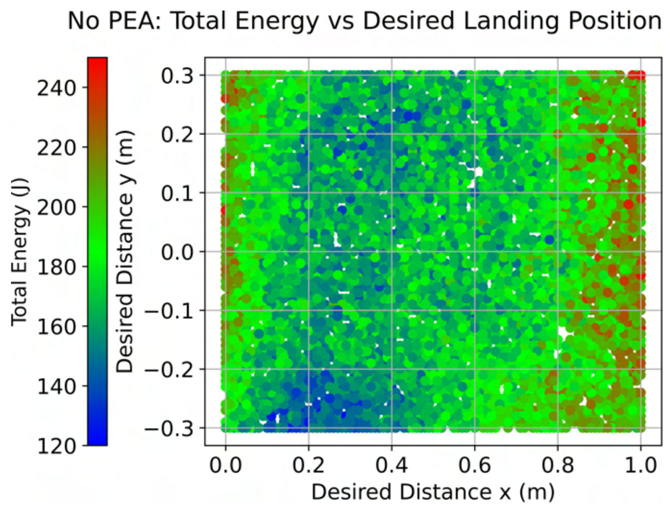


Fig. 13: Illustrates how total energy varies against the desired distance magnitude for the systems with and without springs.

magnitude (see Figure 13). The average energy for performing a jump within the demonstrated range is 117.0 J with stiff actuation and 138.6 J with PEA which corresponds to an improvement of 15.2% in energy efficiency. The jumps are discretised into bins of 5 cm of desired distance magnitude and the average total energy of the jumps is computed. The figure illustrates that the energy for the system without parallel elasticity is higher than the system with PEA. Additionally two peaks in energy are observed for the system with stiff actuation at 0 m and 1m desired distance magnitude. In contrast, for the system with PEA an increase in total energy is observed with an increase in desired distance magnitude.

An important consideration is the peak power experienced by each of the system while performing a jump. Figure 15 visualises the peak power against desired jumping distance for each of the systems. The peak power appears to increase as the desired jumping distance increases. It seems that the system with PEA appears to consistently experience lower peak power for jumps less than 0.5m. In contrast the system with stiff



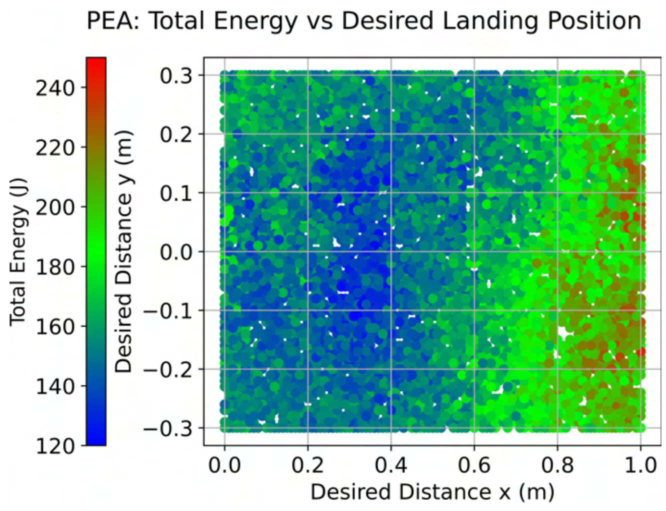
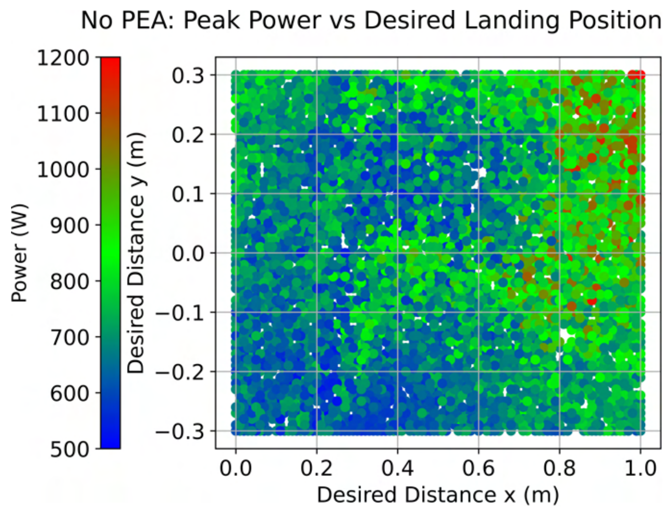


Fig. 14: Demonstrates the total energy consumed by the stiff and elastically actuated system while performing different jumps.



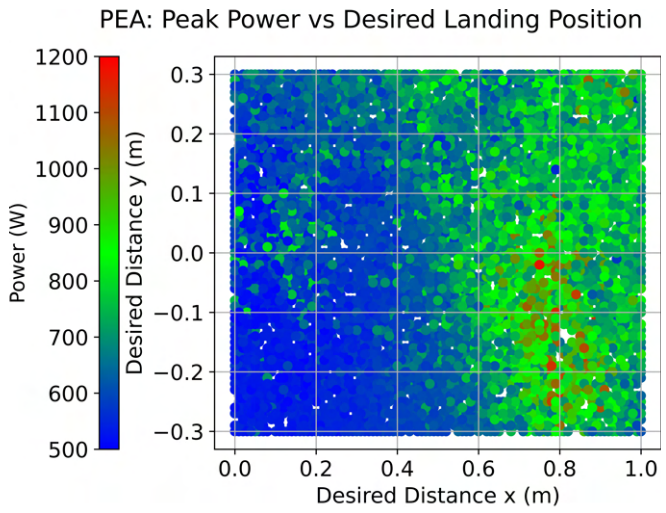


Fig. 15: Visualises the peak power experienced by the stiff and elastically actuated robot for different jumps.

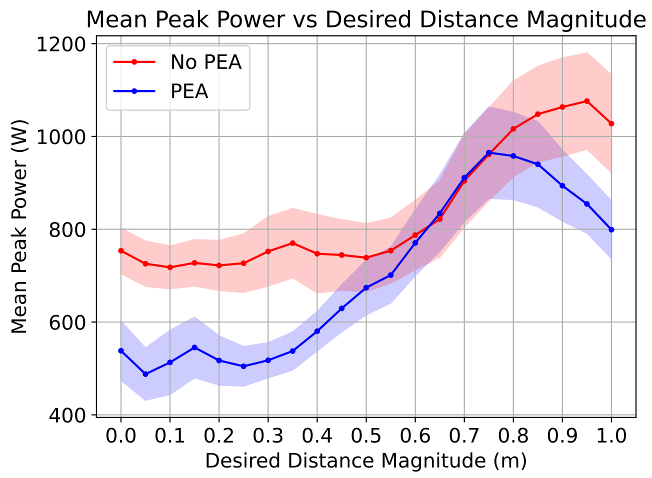


Fig. 16: Displays how mean peak power varies with the desired distance magnitude for the systems with and without PEA.

actuation tends to have higher peak power in that region especially in forward jumps. For all the jumps the system with PEA experiences a mean peak power of 702.1 W. In comparison, the system without PEA experiences a mean peak power of 833.8 W which corresponds to a 15.79% increase in peak power.

To better understand how peak power varies with the desired distance, the peak power is plot against the desired distance magnitude (Fig. 16). This figure indeed verified that for jumps less than 0.5m the PEA system experiences less peak power. For jumps between 0.6 - 0.75m the PEA and stiff system both experience approximately equal peak power. For larger jumps the mean peak power for the system with PEA decreases where as the stiff system still experiences higher peak powers.

C. Uneven Terrain & Domain Randomisation Stage

In this section we discuss the results from the stage of uneven terrain. A video of the system without PEA jumping over uneven terrain can be found [here], whereas for the system with PEA can be found [here]. This part of training utilises the policy trained from the generalisation stage. By fine-tuning the policy capable of performing jumps of different distances on flat ground the system is now able to perform

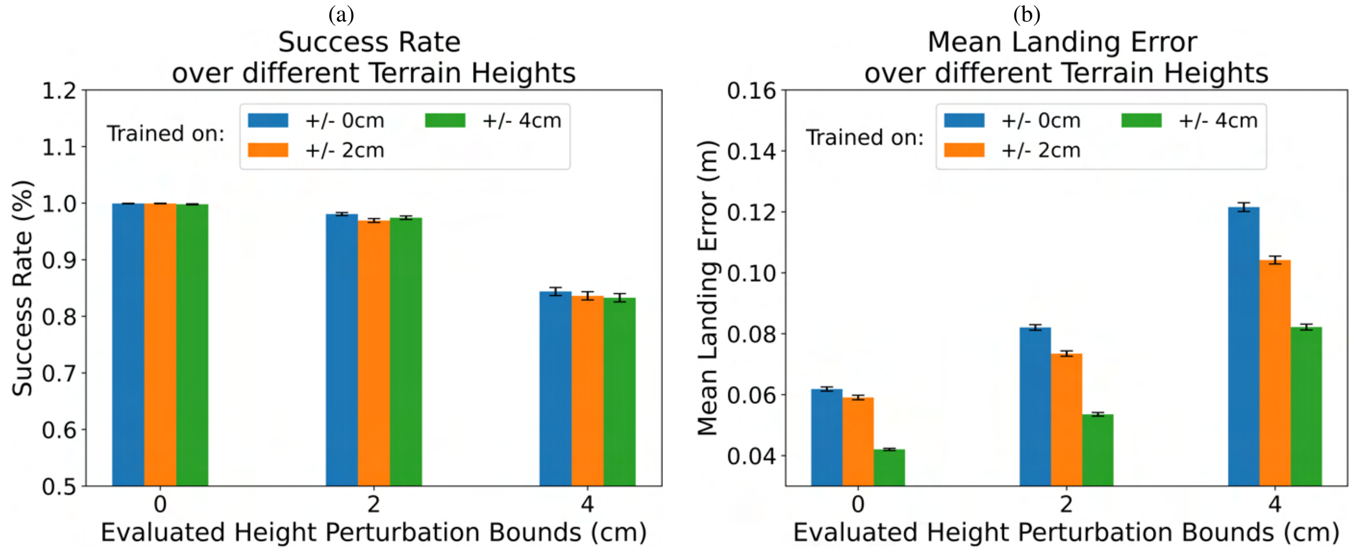


Fig. 17: Visualises the performance of three policies a flat ground policy, a policy trained over +/-2cm uneven terrain and a +/-4cm uneven terrain policy in terms of (a) success rate and (b) landing error.

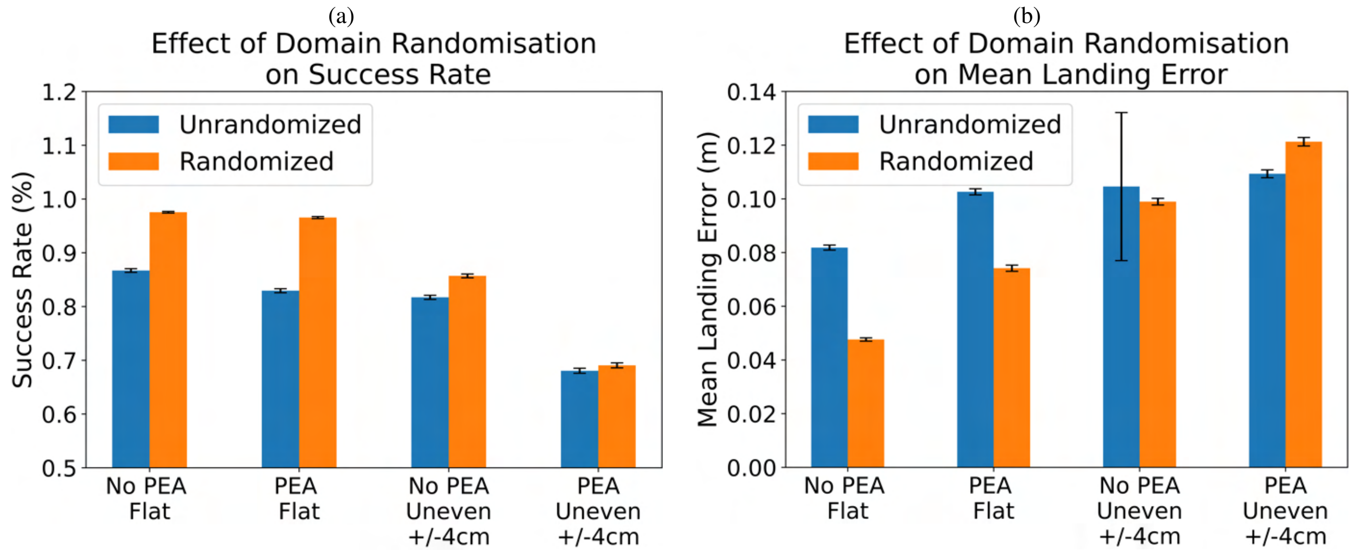


Fig. 18: Demonstrates a comparison in terms of (a) success rate and (b) landing error between the policies before and after domain randomisation. Four interesting policies were considered, the systems with and without PEA with their respective cases over flat and uneven terrain.

jumps over uneven ground. During training the agent is exposed into performing jumps over a variety of environments. The difficulty of the environments is dictated by the maximum magnitude of height perturbations (ϵ) added to the terrain. As visualised in Fig. 17, three policies are compared to each other: a) the flat terrain policy, and two uneven terrain policies trained with (b) ϵ = 2cm and (c) ϵ = 4cm. All three policies are evaluated after training on all three environment setups and their performance is assessed in terms of success rate and landing error. As visualised in Fig. 17a, the success rate decreases for all three policies as the difficulty of the terrain increases. In contrast, as shown in Fig. 17b, as the difficulty of the terrain increases the landing error also increases. This implies that as the environment becomes more difficult the jumps become less precise and the probability of failure increases. By exposing a policy over uneven terrain during its training stage, is observed that the landing error decreases while the success rate remains comparable to the policy trained over flat ground. Furthermore, is observed that the policy trained on perturbations of +/- 4cm outperforms the rest of the policies in terms of precision across all evaluated environments and not just in the environment which was trained for. In order to bridge the gap between simulation and reality, as explained in Sec. IV-E, several parameters were randomised. A video showcasing the system without PEA jumping during the domain randomization stage can be found [here].To infer how the non-randomised policy differs from the randomised policy the two are compared in terms of success rate and landing error. Domain randomisation was performed on the policies with the most challenging uneven terrain environment

($\epsilon = +/-4cm$) as well as the policy with and without PEA over flat ground. A comparison between the baseline policies and their randomised versions is depicted in Fig. 18. All results were obtained by evaluating all 8 policies in a randomised environment. From the aforementioned plot, we can conclude that the randomised policies generally have higher success rate and lower landing error than the non-randomised ones. However, there is one exception: the policy with PEA over uneven terrain does not exhibit an increase in success rate after randomization. This policy also exhibits the lowest success rate and the highest landing error, and it uniquely shows an increase in landing error after randomization. Conversely, the policy with the lowest landing error is the randomised flat terrain policy without PEA after randomization. The policy with the highest variation in landing error is the uneven terrain policy without springs before randomization.

From these observations, we can deduce that jumping with PEA over uneven terrain is more difficult compared to jumping with the rigid robot. This is understandable because the motors of the system need to overcome the stiffness of the springs to adapt to the unevenness of the terrain, which can be more challenging. Further investigation into the optimal spring parameters and rest angles is essential for better understanding how to enhance the performance of a system with PEA over uneven terrain.

VI. Conclusion

To conclude, the proposed methodology started by generating a trajectory using a 2D-SLIP dynamical model. The reference trajectory is used to train a controller which imitates the jumping trajectory and thus allows the robot to perform a jump of constant distance. Then we proceeded by generalising the landing distance that the agent jumps by conditioning the trained policy to a desired landing distance. Lastly the capabilities of the controller were extended over uneven terrain by exposing the controller trained on flat ground to an environment of uneven terrain during the training process.

In order to assess whether is beneficial to use elastic actuation during a jump, a comparison was made between the system with PEA and without it. During imitation the system with PEA experiences 27.64% less peak power compared to the system without. In addition, the system with PEA utilises on average 32.74% less energy per jump. In the phase of generalisation the system with springs has 15.20% lower mean energy and observes 15.79% less peak power compared to its stiff counterpart. In terms of peak height the system with PEA jumps on average 8.79% higher than the stiff system. Finally, the mean landing error with PEA decreases by 11.11%. The above leads us to conclude that the system which utilises elastic actuation in a parallel arrangement is more precise and more energetically efficient when compared with the system with stiff actuation.

Some interesting conclusions can also be drawn from investigating jumps over uneven terrain. The difficulty of the terrain was associated with the magnitude of the height perturbations added to a flat terrain. As the difficulty of the terrain increases the success rate of the trained controller. In contrast, as the difficulty of the terrain increases the landing error increases. Furthermore, it is shown that policies which are trained on terrains with larger magnitude of height perturbation have lower landing error while keeping their success rate comparable to that of the baseline policy. Finally, in order to make the jumps of the agent more realistic several parameters were randomised. As commented earlier, domain randomisation seems to increase the success rate of the non-randomised policy and decrease its landing error. It seems that the policy with PEA finds it more difficult to jump over uneven terrain compared to the policy without springs. Further investigation is required into the parameters of the PEA to enhance its performance over uneven terrain.

While the aforementioned methodology has yielded to a controller that enabled us to perform jumps of various distances both over flat and uneven terrain using just a single demonstration it also comes with certain shortcomings as well as potential for improvement. One of the limitations of the proposed controller structure is that it only uses proprioceptive information. The use of information from exteroceptive sensors, such as a height map of the jumping terrain in the vicinity of the robot, can be used to greatly improve the performance over uneven terrain. The use of height map data would allow performing anticipatory motions during landing that help the robot stabilise it self more easily. Furthermore, can a curriculum be used which combines the training of all 4 learning stages? This would avoid the incremental fine-tuning of the policy and make the process simpler. Another direction that could potentially be explored is how much the model chosen for imitation affects the training speed as well as the performance of the controller. Would a trajectory generated using the full dynamical model of the system lead to better performance and faster imitation? Furthermore, it is evident that the energetic efficiency of the system with springs compared to the stiff system was better. However, when going from the imitation stage to the generalisation a decrease in the energetic benefit provided is observed. Can the use of multiple demonstrations in the generalisation stage be used to enhance the energetic efficiency even more? Addressing the aforementioned questions could enhance the design of controllers, enabling them to navigate even the most complex terrain with greater success, while also improving the efficiency of the methodology and the energetic performance of the controller itself.

References

[1] T. J. Dawson and C. R. Taylor, "Energetic Cost of Locomotion in Kangaroos," *Nature*, vol. 246, no. 5431, pp. 313–314, Nov. 1973, number: 5431 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/246313a0

[2] S. Shield, N. Muramatsu, Z. Da Silva, and A. Patel, "Chasing the cheetah: how field biomechanics has evolved to keep up with the fastest land animal," *Journal of Experimental Biology*, vol. 226, no. Suppl_1, p. jeb245122, Apr. 2023. [Online]. Available: https://doi.org/10.1242/jeb.245122

[3] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, p. eabc5986, Oct. 2020. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abc5986

[4] P. Fankhauser and M. Hutter, "ANYmal: A Unique Quadruped Robot Conquering Harsh Environments," *Research Features*, vol. 126, pp. 54–57, 2018.

[5] C. Dario Bellicoso, F. Jenelten, P. Fankhauser, C. Gehring, J. Hwangbo, and M. Hutter, "Dynamic locomotion and whole-body control for quadrupedal robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 3359–3365, iSSN: 2153-0866. [Online]. Available: https://ieeexplore.ieee.org/document/8206174

[6] Q. Nguyen, M. J. Powell, B. Katz, J. D. Carlo, and S. Kim, "Optimized Jumping on the MIT Cheetah 3 Robot," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 7448–7454, iSSN: 2577-087X. [Online]. Available: https://ieeexplore.ieee.org/document/8794449

[7] M. Ernst, H. Geyer, and R. Blickhan, "Spring-Legged Locomotion On Uneven Ground: A Control Approach To Keep The Running Speed Constant," in *Mobile Robotics*. Istanbul, Turkey: World Scientific, Aug. 2009, pp. 639–644.

[8] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning Agile Robotic Locomotion Skills by Imitating Animals," Jul. 2020, arXiv:2004.00784 [cs].

[9] Y. Fuchioka, Z. Xie, and M. van de Panne, "OPT-Mimic: Imitation of Optimized Trajectories for Dynamic Quadruped Behaviors," Nov. 2022, arXiv:2210.01247 [cs]. [Online]. Available: http://arxiv.org/abs/2210.01247

[10] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Robust and Versatile Bipedal Jumping Control through Reinforcement Learning," May 2023, arXiv:2302.09450 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2302.09450

[11] Q. Yao, J. Wang, S. Yang, C. Wang, H. Zhang, Q. Zhang, and D. Wang, "Imitation and Adaptation Based on Consistency: A Quadruped Robot Imitates Animals from Videos Using Deep Reinforcement Learning," arXiv, Tech. Rep. arXiv:2203.05973, Mar. 2022, arXiv:2203.05973 [cs] type: article. [Online]. Available: http://arxiv.org/abs/2203.05973

[12] F. Vezzi, J. Ding, A. Raffin, J. Kober, and C. Della Santina, "Two-Stage Learning of Highly Dynamic Motions with Rigid and Articulated Soft Quadrupeds," Sep. 2023, arXiv:2309.09682 [cs]. [Online]. Available: http://arxiv.org/abs/2309.09682

[13] V. Atanassov, J. Ding, J. Kober, I. Havoutis, and C. Della Santina, "Curriculum-Based Reinforcement Learning for Quadrupedal Jumping: A Reference-free Design," Jan. 2024, arXiv:2401.16337 [cs]. [Online]. Available: http://arxiv.org/abs/2401.16337

[14] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–14, Aug. 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3197517.3201311

[15] G. Bellegarda and Q. Nguyen, *Robust Quadruped Jumping via Deep Reinforcement Learning*, Nov. 2020.

[16] C. Nguyen and Q. Nguyen, *Contact-timing and Trajectory Optimization for 3D Jumping on Quadruped Robots*, Oct. 2021.

[17] C. Nguyen, L. Bao, and Q. Nguyen, "Continuous Jumping for Legged Robots on Stepping Stones via Trajectory Optimization and Model Predictive Control," Sep. 2022, arXiv:2204.01147 [cs]. [Online]. Available: http://arxiv.org/abs/2204.01147

[18] A. K. Valenzuela, "Mixed-integer convex optimization for planning aggressive motions of legged robots over rough terrain," Thesis, Massachusetts Institute of Technology, 2016, accepted: 2016-07-01T18:23:33Z. [Online]. Available: https://dspace.mit.edu/handle/1721.1/103432

[19] Y. Ding, C. Li, and H.-W. Park, "Kinodynamic Motion Planning for Multi-Legged Robot Jumping via Mixed-Integer Convex Program," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 3998–4005, iSSN: 2153-0866. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9341572

[20] Z. Song, L. Yue, G. Sun, Y. Ling, H. Wei, L. Gui, and Y.-H. Liu, "An Optimal Motion Planning Framework for Quadruped Jumping," Jul. 2022, arXiv:2207.12002 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2207.12002

[21] J. Ding, V. Atanassov, E. Panichi, J. Kober, and C. D. Santina, "Robust Quadrupedal Jumping with Impact-Aware Landing: Exploiting Parallel Elasticity," *IEEE Transactions on Robotics*, pp. 1–20, 2024, conference Name: IEEE Transactions on Robotics. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10552408

[22] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius, "Learning Agile Skills via Adversarial Imitation of Rough Partial Demonstrations," Nov. 2022, arXiv:2206.11693 [cs]. [Online]. Available: http://arxiv.org/abs/2206.11693

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017, arXiv:1707.06347 [cs]. [Online]. Available: http://arxiv.org/abs/1707.06347

[24] J. Ding, P. Posthoorn, V. Atanassov, F. Boekel, J. Kober, and C. D. Santina, "Quadrupedal Locomotion With Parallel Compliance: E-Go Design, Modeling, and Control," *IEEE/ASME Transactions on Mechatronics*, pp. 1–10, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10550040/

[25] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots," Jun. 2018.

Appendix A
Reference Generation

Equations from 4b to 4e are used to constrain the initial and final states of the generated trajectory. Additionally, an equality constraint (eq. 4f), is used to ensure that the height position before take off is close to the maximum height of the robot. For the propagation of the state forward in time a 2^{nd} order Taylor Series is used to estimate the next state given the current one. The 2^{nd} order Taylor series is defined as:

$$\boldsymbol{p}_{i+1} = \text{Taylor}(\boldsymbol{p}_i, \dot{\boldsymbol{p}}_i, \ddot{\boldsymbol{p}}_i, \delta t) = 0.5\ddot{\boldsymbol{p}}_i\ \delta t^2 + \dot{\boldsymbol{p}}_i\ \delta t + \boldsymbol{p}_i. \tag{13}$$

The rest of the constraints are used conditionally depending on which phase of the trajectory is being evaluated. During the stance phase the center of mass (CoM) position is kept within the support polygon of the feet (eq. 4i). This is a commonly used stability criterion in legged robots to ensure that the robot remains stable during a stance pose. However, it also comes with the inherent assumption that the feet position remains fixed during the stance phase. The dynamics that dictate the stance phase for a spring loaded mass system are given in eq. 4j. Furthermore, during stance phase the acceleration of the CoM in the z direction is constrained to be less than or at most equal to the gravitational acceleration. This gives rise to the constraint in eq. 4k as the robot is unable to pull it self from the ground and thus cannot attain a vertical acceleration greater than the gravitational acceleration. In contrast to the stance phase, during the flight phase is assumed that the CoM position cannot be influenced by the control input and thus the system is autonomous during that phase. Hence, the only force acting on the system during the flight phase is assumed to be gravity (eq. 4l).

The cost to be minimised during optimisation can be decomposed into three individual terms. A cost during stance phase (J_{stance}) is used to penalise large control inputs while it also imposes a penalty on the jerk of the position to ensure a smooth stance trajectory. During the take off phase there is a negative cost ($J_{\text{take off}}$) which rewards the vertical velocity and thus prompts the robot to jump. Finally, during the flight phase a cost (J_{flight}) prompts the solution to be more realistic by keeping the vertical position of the system close to that of take off. This also makes the optimisation problem more tractable. The explicit definitions for the individual costs of the cost functions in eq. 4a are given by:

$$J_{\text{stance}} = \sum_{i=1}^{N_s} w_u \left\|\boldsymbol{u}(i)\right\|_2 + w_{jerk} \left\|\dddot{\boldsymbol{p}}(i)\right\|_2 \tag{14a}$$

$$J_{\text{take off}} = -w_{\text{take off}}\ \dot{p}^z(N_s - 1) \tag{14b}$$

$$J_{\text{flight}} = \sum_{i=N_s}^{N_s+N_f} w_{\text{flight}}(p^z(i) - p^z_{\text{take off}}). \tag{14c}$$

The parameters used for generating the reference trajectory are stated in Table I.

Table I: Trajectory Planner Parameters

Planner Parameter	**Value**
Input Weight (w_u)	0.3
Jerk Weight (w_{jerk})	0.001
Flight Weight (w_{flight})	3000
Take Off Weight $(w_{\text{take off}})$	200
Stance Phase Steps (N_s)	20
Stance Phase Steps (N_f)	20
Take of Height $(p^z_{\text{take off}})$	0.42
Equivalent Spring Stiffness (k_{eq})	2000
Initial Position $(\boldsymbol{p_0} = [x_0, z_0]^T)$	$[0.0, 0.32]^T$
Final Position $(\boldsymbol{p_f} = [x_f, z_f]^T)$	$[0.40, 0.32]^T$
Initial Velocity $(\boldsymbol{v_0} = [v^x_0, v^z_0]^T)$	$[0.0, 0.0]^T$

APPENDIX B
REINFORCEMENT LEARNING ALGORITHM

This section outlines the hyper-parameters and parameters used for the reinforcement learning environment. The noise added to the observations of the controller are shown in Table II. The hyper-parameters used for the PPO algorithm are visualised in Table III. Finally the reward hyper-parameters for the system with PEA and without PEA for both the imitation and generalisation stage are shown in Table IV. The training details for every learning stage are shown in Table VI.

Table II: Noise of Policy Observations

Observation	**Noise Magnitude**	**Units**
Joint Angles per Leg	[2.0, 2.0, 2.0]	deg
Joint Velocity	[1.25, 1.25, 1.25]	rad/s
Linear Velocity	[0.4 0.10 0.4]	m/s
Angular Velocity	0.09	rad/s
Orientation	1.8	deg
Remaining Time	0.00	s
Desired Jumping Distance	[0.0, 0.0]	m

APPENDIX C
DOMAIN RANDOMISATION DETAILS

This section outlines the randomisation hyper-parameters used to randomise the policy for the flat ground and uneven terrain. The values used for the randomisation of the policies are illustrated in Table. V.

Table III: Proximal Policy Optimisation (PPO) Parameters

RL Algorithm Parameters	**Value**
Total Time-steps	10×10^6
Linearly Decreasing Starting Learning Rate	1×10^{-4}
Batch Size	4096
Mini Batch Size	128
General Advantage Estimator (GAE) Lambda	0.95
Discount Factor (γ)	0.99
Number of Epochs	10
Clip Range	0.2
Maximum Gradient Norm	0.5
Value Function Coefficient	0.5

Table IV: Reward Parameters

Imitation Stage		
System Setup	**With PEA**	**Without PEA**
Parameter Name	**Parameter Value**	
Joint Position Weight (w^q)	0.30	0.30
Joint Position Length Scale (k_q)	5.00	5.00
Base Position Weight (w^p)	0.35	0.35
Base Position Length Scale (k_p)	500	500
Base Orientation Weight (w^o)	0.35	0.35
Base Orientation Length Scale(k_o)	50	50
Penalty Weight Landing Velocity (w^{pen}_v)	2.5×10^{-3}	2.5×10^{-3}
Penalty Weight Joint Velocity $(w^{\text{pen}}_{\text{dq}})$	5.0×10^{-5}	5.0×10^{-5}
Penalty Weight Joint Acceleration $(w^{\text{pen}}_{\text{ddq}})$	2.1×10^{-6}	2.1×10^{-6}
Generalisation Stage		
System Setup	**Without PEA**	**With PEA**
Parameter Name	**Parameter Value**	
Joint Position Weight (w^q)	0.15	0.20
Joint Position Length Scale (k_q)	5.00	5.00
Base Position Weight (w^p)	0.00	0.00
Base Position Length Scale (k_p)	500	500
Base Orientation Weight (w^o)	0.45	0.50
Base Orientation Length Scale(k_o)	50	50
Goal Position Weight (w^{goal})	0.20	0.20
Goal Position Length Scale(k_{goal})	20	20
Penalty Weight Landing Velocity (w^{pen}_v)	1.5×10^{-2}	1.5×10^{-2}
Penalty Weight Joint Velocity $(w^{\text{pen}}_{\text{dq}})$	5.0×10^{-5}	5.0×10^{-5}
Penalty Weight Joint Acceleration $(w^{\text{pen}}_{\text{ddq}})$	2.1×10^{-6}	2.1×10^{-6}
Termination Limits Growth (α)	0.10	0.10
Termination Limits Constant (β)	0.05	0.05
Linear Base Velocity Weight (w^v)	0.30	0.20
Linear Base Velocity Length Scale (k^v)	5.00	5.00

Table V: Domain Randomisation Table

Randomised Variable	Noise Magnitude	Units
Leg Mass Error Range	$\mathrm{U}\left[-30\%, 30\%\right]$	n/a
Offset Mass	$\mathrm{U}\left[0.0, 2.0\right]$	kg
Base Mass Error Range	Accordingly	n/a
Offset Mass Position - x	$\mathrm{U}\left[-0.1, 0.1\right]$	m
Offset Mass Position - y	$\mathrm{U}\left[-0.05, 0.05\right]$	m
Offset Mass Position - z	$\mathrm{U}\left[-0.05, 0.05\right]$	m
Initial State	$\mathcal{N}\left(\mu = 0.0, \sigma = 1.7\right)$	deg
Joint Friction	$\mathrm{U}\left[0.0, 0.04\right]$	Nm
Motor Strength	$\mathrm{U}\left[0.8, 1.2\right]$	n/a
Ground Friction Coeff.	$\mathcal{N}\left(\mu = 0.8, \sigma = 0.1\right)$	n/a
Feet Friction Coeff.	$\mathcal{N}\left(\mu = 0.8, \sigma = 0.1\right)$	n/a
Ground Restitution Coeff.	$\mathrm{U}\left[0.0, 0.2\right]$	n/a
Feet Restitution Coeff.	$\mathrm{U}\left[0.4, 0.6\right]$	n/a
Springs Stiffness Error Range	$\mathrm{U}\left[-20\%, 20\%\right]$	n/a
Springs Damping Error Range	$\mathrm{U}\left[-20\%, 20\%\right]$	n/a
Springs Rest Angle Error Range	$\mathrm{U}\left[-20\%, 20\%\right]$	n/a

Table VI: Training Details of each learning stage

Imitation	
Reward	• Reward tracking of reference for: position, orientation, desired jumping distance and joint angles. • Reward survival of episode. • Penalise XY base velocity once landed. • Penalise joint velocity and acceleration once on air. • For Details see Table IV.
Observations	History of last 20 states of: • Joint Angles • Joint Velocities • Linear Velocities • Angular Velocities • Base Orientation • Episode Remaining Time • Desired Jumping Distance • Last Action
Desired Distance	Kept fixed to the demonstration desired distance.
Environment	Flat Terrain
Landing Distance Generalisation	
Reward	Starting from Imitation Reward: • Removed tracking of Position. • Added tracking of Desired velocity. • Added adaptive landing termination. • See Table IV for details.
Observations	Same as Imitation stage.
Desired Distance	Uniformly sampled at the episode start. X-Desired Distance: 0.0 to 1.0 m Y-Desired Distance: -0.3 to 0.3 m
Environment	Flat Terrain
Uneven Terrain Jumping	
Reward	Same as Landing Distance Generalisation Stage.
Observations	Same as Imitation stage.
Desired Distance	Same as Landing Distance Generalisation Stage.
Environment	Uneven Terrain. • Terrain is randomly sampled at the episode start. • There is a 20% probability of sampling flat terrain. • Maximum terrain perturbation boundaries defined by ϵ.
Domain Randomization	
Reward	Same as Landing Distance Generalisation Reward.
Observations	Same as Imitation stage.
Desired Distance	Same as Landing Distance Generalisation Stage.
Environment	Flat or Uneven Terrain depending on trained policy.