Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Reinforcement Learning for Quadrupedal Robot Control with Novel Kinematics

Bc. Andrej Kružliak

Supervisor: Rastislav Marko, MSc.
Field of study: Cybernetics and Robotics
January 2024

Acknowledgements

I would like to thank my supervisor, Rastislav Marko, MSc., for his guidance and support. The always precise, constructive criticism and morale boost kept me progressing through the whole thesis in a rapid manner. Another big thanks go to the whole team of Panza Robotics, who made this thesis possible by creating an amazing robotics masterpiece called Artaban, creating a welcoming working atmosphere, and making my high-school robotic dreams come true. I appreciate the initial conversations with the second supervisor doc. Ing. Karel Zimmermann, Ph.D. which led to the settling of this Diploma thesis in cooperation with the external company, Panza Robotics s.r.o. Thanks go to my family, close loved ones, and friends who understand the role of university in my life and showed me great support through the whole academic journey.

Declaration

Abstract

This thesis aims to implement and assess an innovative kinematics solution of the quadruped robotic platform Artaban by Panza Robotics, in a simulation environment within a deep reinforcement learning setting. The innovative kinematics include two distinct parallel mechanism types. One transfers the torque from the motor through a universal joint mechanism called Cardan mechanism, and the other is a four-link mechanism on the rear legs. The innovative kinematics are implemented and tested in three kinematic configurations: the original with distinct front and rear legs, a uniform front-legged configuration, and a front-legged Cardan configuration optimizing the transmission of torque. The Cost of Transport (CoT) metric, judging the amount of effort per robot velocity, is used for assessing the performance of gaits produced by those configurations. The original configuration encountered simulation issues due to improper loop-closure, leading to non-viable gaits. These challenges were absent in the front-legged configuration, which successfully generated a valid gait. The subsequent implementation of the Cardan mechanism yielded an even more efficient and visually pleasing gait. The Cardan mechanism's gait was identified as the most efficient from the point of view of the CoT metric, highlighting the mechanism's potential for enhancing robotic locomotion. The results obtained from the front-legged configuration with the Cardan mechanism are expected to translate effectively to the original kinematic configuration once a stable simulation of the rear legs is achieved.

Keywords: deep reinforcement learning, Isaac Sim, Nvidia, quadruped legged robotics, Artaban robotic platform, Cardan mechanism, parallel mechanism, PPO, Proximal policy optimization, Actor-Critic methods

Supervisor: Rastislav Marko, MSc.
Panza Robotics, s.r.o., Trnava, Slovakia

Abstrakt

Táto diplomová práca sa zameriava an implementáciu a porovnanie inovatívneho kinematického riešenia pre štvornohú robotickú platformu Artaban od firmy Panza Robotics, v simulačnom prostredí za pomoci hlbokého posilovaného učenia. Inovatívna kinematika zahŕňa dva typy paralelných mechanizmov. Jeden prenáša krútiaci moment z motora cez univerzálny kĺb zvaný Kardanov mechanizmus, druhý je štvortyčovým mechanizmom na zadných nohách. Inovatívna kinematika je implementovaná a testovaná v troch kinematických konfiguráciach: originálna s odlišnými zadnými a prednými nohami, jednotná konfigurácia s oboma pármi nôh vo forme predného páru a jednotná konfigurácia s oboma pármi nôh vo forme predného páru s modelom Kardanového mechanizmu na každej nohe, optimalizujúc prenos krútiaceho momentu. Metrika účinnosti pohybu, ktorá posudzuje vynaložený výkon v pomere k výslednej rýchlosti robota, je použitá na posúdenie účinnosti chôdze, ktorá bola vyprodukovaná pre spomínané kinematické konfigurácie. Originálna konfigurácia robota preukazuje simulačné problémy kvôli neúplnému dodržiavaniu kinematického obmedzenia na slučkový uzáver zadného štvortyčového mechanizmu, vedúc k nerealistickým druhom chôdze. Avšak, tieto simulačné problémy sa neprejavujú pre konfiguráciu s prednými nohami, ktorá produkuje realistickú chôdzu. Nasledujúca implementácia Kardanového mechanizmu poskytuje značne efektívnejšiu a prirodezene vyzerajúcu chôdzu robota. Chôdza za pomoci Kardanového mechanizmu je identifikovaná ako najefektívnejšia z pohľadu metriky účinnosti pohybu, zdôrazňujúc potenciál mechanizmu na zlepšenie robotickej chôdze v budúcnosti. Očakáva sa, že výsledky získané z konfigurácie s prednými nohami s Kardanovým mechanizmom bude možné po dosiahnutí stabilnej simulácie zadných nôh efektívne preniesť do pôvodnej kinematickej konfigurácie.

Kľúčové slová: hlboké posilované učenie, Isaac Sim, Nvidia, štvornohá robotika, robotická platforma Artaban, Kardanový mechanizmus, paralelný mechanizmus, PPO, Proximal policy optimization, Actor-Critic metódy

Preklad názvu: Posilované učenie pre ovládanie štvornohého robota s inovatívnou kinematikou

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

The locomotion of a legged robot is much more complex and computationally demanding than it is for tracked robots but provides many more opportunities when walking on either uneven, unknown, or otherwise difficult terrain. Puddles, tall grass, differences in terrain surface friction, and overall sensor imperfections add to the difficulty of legged locomotion. Historically, approaches to legged robot locomotion have not always contained machine learning. Various classical locomotion approaches include utilizing Central Pattern Generators (CPG) [3] or incorporating engineering dynamics through optimal control methods like Model Predictive Control (MPC) [4, Chapter 4], [5]. These methods aim to fulfill specific constraints, such as the Zero Moment Point, as well as satisfying trajectories for robot legs or the center of gravity. Even though the progress in classical predictive control without machine learning is impressive, the progressively cheaper computing and more efficient memory management in reinforcement learning algorithms brought a fresh breeze into legged locomotion. Reinforcement learning (RL) is not a new paradigm, but the new and cheaper computation hardware opened opportunities for more researchers and projects. RL algorithms win at popular games such as GO, help with protein folding, and legged robot locomotion, among other things, is no exception.

The current state-of-the-art solutions to the legged locomotion problem usually contain some form of terrain property sensing, i.e., exteroceptive sensing combined with the readings from the robot's IMU, motor torques, etc., i.e., interoceptive sensing. Although recent papers on quadrupedal robotics [6, 7] suggest that simply using so-called privileged information in simulation training is sufficient for *blind locomotion.* Although this means that recovery from slippery terrain or stairs is robustly possible, the quality of such gait is not outstanding from the point of view of gait speed. Miki et al. [7] show that using privileged learning in tandem with vision provides even more robust, quick, and smooth traverse of stairs and other difficult terrains. The notion of privileged information is explained in more detail in Section 2.

Simulating the environment for such a complex problem as locomotion may seem excessively hard. One might assume that to robustly simulate a robotic gait on a complex surface, such as sand, we must simulate all the particles that come into contact with the robot. Or that whatever representation of the environment, unless it simulates the real world exhaustively and completely, the trained policy for walking will be only asymptotically robust [8]. Therefore, people such as Rodney Brooks, the founder of the iRobot company, said *"Simulations are doomed to succeed"* [8]. This draws attention to the common caveats that become progressively more and more tempting for researchers the more their application fails, such as tweaking the simulation environment until the results are favorable. Although recent publications consistently prove that sim-to-real transfer is possible and actually *tweaking the simulation environment* by granting some parts of the policy access to the ground truth information is beneficial if done right. Publications extensively described in Section 2 show that the reinforcement learning approach via simulation is taking the main stage in robotic locomotion, not with a whimper but a bang [9].

1.2 Goals

The main goal of this thesis is to train a quadruped gait with the model of the Artaban robotic platform and its special kinematics and compare it to the conventional approach to kinematics. The main goal consists of partial goals, such as implementing the closed-loop actuation for the specific linkage the robot possesses, training a gait with this specific linkage, then implementing a kinematic configuration without this specific linkage (conventional), training the gait with it, and comparing. The last partial goal is implementing the specific torque transmission mechanism called the Cardan mechanism, training another gait, and again, comparing. For the comparison, a metric called Cost of Transport is chosen.

Chapter 2

Related Work

2.1 Blind legged locomotion

It is possible to solve the problem of legged locomotion problem without the exteroceptive data. This “blind” locomotion, as mentioned in work by T. Azayev et al. [10], uses the combination of different so-called expert policies. The expert policies are modeled by independently trained Artificial Neural Networks (ANNs) multiplexed by a Recurrent Neural Network (RNN) conditioned on the state history. The agent navigates a partially observable environment consisting of discrete environment parts that differ in slopes, may contain small bumps, hills, pipes, or consist of some form of stairs. The problem was modeled and solved using a hexapod, a six-legged robot in a physics simulation environment, MuJoCo. The agent’s navigation in a partially observable environment is formally modeled as POMDP (Partially Observable Markov Decision Process). In this work, a two-layer recurrent policy is proposed. The multiplexing of independently trained policies shows better results than using a single RNN policy, resulting in a walking gait that is somewhat an average over all terrains. Such policy has trouble adjusting to the current environment and results in a suboptimal gait for the whole sequence of environments. Deep Reinforcement Learning (DRL) is utilized with a slightly modified Proximal Policy Optimization (PPO) as an optimization strategy to train the model.

2.2 Legged locomotion with exteroceptive information

With the computational power and sensors available, a more informed approach results in better and more robust locomotion that produces good sim-to-real results, such as in Miki. et.al. [7], or G. B. Margolis [11]. In Miki et al., the locomotion aims at high generality of locomotion in virtually (and also physically) any terrain. The so-called perceptive locomotion approach is preferred since the robots usually come with all the sensors already installed onboard. The quadruped ANYmal from the ETH Zurich Robotic Systems Lab was the physical platform. The result of this paper is that using privileged learning, a more robust locomotion controller can be constructed that has minimal problems traversing rugged terrain, such as hills, grasslands, dirt roads, and so on. So was demonstrated on a 2.2 km hike through Etzel Hill with 120 m elevation without a single fall. The exteroceptive data is incorporated using Teacher and Student policy training. The first policy, the teacher is trained using so-called privileged information, such as noiseless terrain measurements, ground friction, and external disturbances. All of that privileged information is completely known and available to the teacher policy. The student policy is trained to reproduce the teacher policy's actions without using the privileged data. The world dynamics are modeled as POMDP, and the employed optimization technique for the teacher policy, modeled as a Gaussian policy, is PPO. Since the model is partially observable, a belief state encoder accounts for the hidden states of the environment. The authors provide a graphical overview of the belief encoder and decoder architectures, shown in Fig. 2.1.

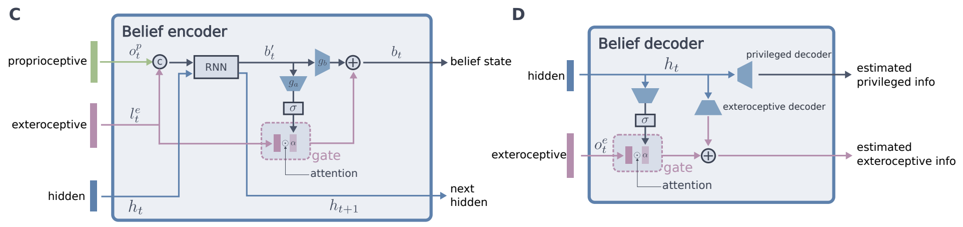


Figure 2.1: Description of the encoding (C) and decoding (D) of the proprioceptive and exteroceptive information into and from the belief state respectively.

Here, the proprioception, exteroceptive noisy observations, and hidden states are encoded by the RNN into an intermediate belief state. From this, an attention vector is computed that controls the amount of the exteroceptive information that enters the final belief state. A Gated Recurrent Unit (GRU) is the RNN architecture of choice.

In Margolis et al. [11], a similar teacher-student policy is implemented with

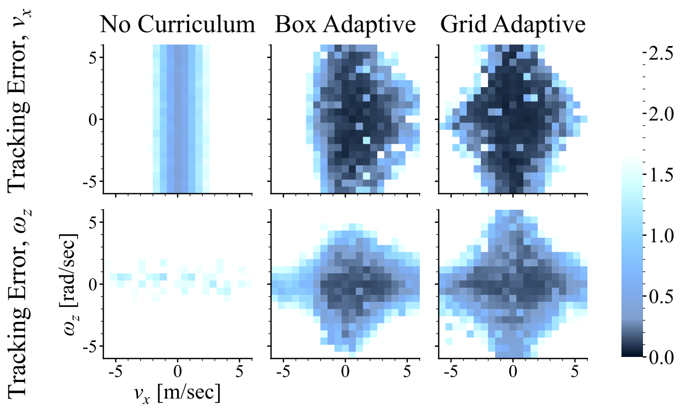


Figure 2.2: Heatmap representing the tracking error of converged curricular strategies

the addition of a curriculum regarding also the target speed, not only terrain and domain randomization, as it was in [7]. Such a curriculum makes it easier to recover rewards from high-speed movement training, which is the goal of the paper. The probability of obtaining the reward in higher-speed scenarios rises by slowly ramping up the target speed as the robot progresses. This solves the problem of sparse rewards for agile movements. The curriculum update rule the authors provide is called the Grid Adaptive Curriculum Update Rule. The update rule works by adding neighboring commands to the speed and direction the agent is given. The update adds probability density to the neighboring commands of the agent's speed and direction if those commands have not already been added. The neighboring commands are defined as those close to the agent's original command in a grid. If the agent succeeds in a difficult command and the reward threshold is met, the neighboring commands will be added to the update to make the task even more challenging. Such curriculum learning, of which tracking errors are shown as presented by the authors in Fig. 2.2, enables high-speed locomotion.

Tsounis et al. [12] address in their work the challenge of legged locomotion in uneven terrain by creating a two-level system merging two distinct methodologies: model-based motion planning with reinforcement learning to train neural network policy for terrain-aware locomotion. Their approach combines dynamic feasibility criteria with Markov decision processes. The two-level system comprises a Gait Planner (GP) and a Gait Controller (GC), integrating both exteroceptive and interoceptive data. The GP acts as a terrain-aware planner, while the GC functions as a motion planner and controller. This structure allows for responsive and adaptive control of the robot's movement. The authors use a Linear Program (LP) for transition feasibility, which respects the robot's movement capabilities while simplifying the training process. Their approach reduces the computational demands typically associated with DRL since invalid robot configurations are not allowed by definition, so the policy does not need to spend time learning the valid configurations first. However, this work was published in 2019. Since then, major progressions have been made in computational efficiency (such as [13]), which makes this approach slightly less relevant. However, applying the key takeaway in using the knowledge about the robot's kinematics as a prior and incorporating it into the training process reduces the training time even if the computational efficiency grows independently.

2.3 General world model learning

A more advanced approach to the locomotion problem may be to use a more general agent. Using encoders or similar architectures that learn the world model, we can get a more general approach to solving the locomotion task. Such an approach was chosen by Hafner et al. from DeepMind with the DreamerV3 agent. This agent achieved great results in [14] sparse-reward problems like physical control and DMLab. As the first out-of-the-box algorithm, it also collected diamonds in the video game Minecraft. Collecting the diamonds in this game is a strongly sparse reward requiring multiple intermediate steps to go right beforehand. The part that is related to this work is the Control Suite division, where different locomotion tasks were studied. The agent consists of 3 NNs: the world model, which predicts future outcomes of potential actions; the critic that judges the value of each situation; and the actor who learns to reach those valuable situations. The learning in this model is purely reinforcement learning without expert data or manually-crafted curricula. The world model is trained using autoencoding of actions via a Recurrent State-Space Model (RSSM). The actor chooses what to do next in the game based on what it has learned so far, without thinking too far into the future, limited by a discount factor. The critic helps the actor learn by predicting how good each decision will be in the long run. To explore more options in a sparse-reward world, an entropy regularizer is employed to scale the rewards appropriately so small rewards do not get too amplified and large rewards get more limited.

2.4 Training criteria and parameters

Choosing a quality criterion function for optimization is vital for good learning results. This concerns the creation of a reward function in such a way that the agent's goal is aligned with ours. Previous papers [7, 10, 11] dedicated whole appendix sections to describe their exact choice of rewards. The rewards were usually hand-crafted and embodiment-specific, which makes sense, given the scope of this project being quadrupedal locomotion. If only the distance is used as a reward, then erratic behavior might emerge from the reinforcement learning because there are no constraints on jumping or other abrupt movements. By giving restrictions on the tilt of the top plane of the robot, the authors actively encourage the robot to stay level, making future use of cameras much more robust. An embodiment-non-specific reward is the Cost of Transport (CoT). The CoT has various definitions but generally refers to a ratio of power consumption divided by the robot's momentum. So the lower the cost, the better momentum per effort is obtained, resulting in movement with the least effort.

In more difficult terrains, the CoT is not easily computed and needs to be estimated from training data. This was done in the work by Prágr et al. [15]. The authors discuss a method for predicting the CoT based on the terrain it traverses. The method uses a combination of terrain features and low-level ML algorithms, such as Hoeffding trees, Support Vector Regression, etc., to learn how different types of terrain affect the robot's energy consumption. The framework has two stages: a learning phase, where the robot collects data about the terrain and energy consumption, and an inference phase, where the learned model is used to predict the energy consumption of new terrain using an aerial view. The method uses temporary feature storage to maintain a dictionary of terrain features extracted from the robot's field of view, which gets randomly pruned when the capacity is reached. This allows for training the robot locally instead of keeping an ever-growing feature map.

In the 2021 paper "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning," Rudin et al. [13] describe how massive parallelism helps with training time and training efficiency. The implementation style and insights are closely followed up on in this work. The main focus is on the effects of using a large number of parallel robots (massive parallelism) in training simulations on policy performance. Initially, a baseline is set with 20,000 robots and 50 steps, leading to a large batch size and effective policy but with longer training times. Experiments were then conducted to find a balance between the number of robots, batch size, and training duration. It was observed that too many robots led to a decrease in policy performance, explained by a reduced time horizon for each robot. Conversely, below a certain number of robots, a gradual decrease in performance was noted. The study aims to find an optimal trade-off between training time and policy effectiveness. The whole reinforcement learning pipeline is parallelized and delegated to GPU, including the physics simulation, reward and observation calculations, policy inference, and backpropagation.

Chapter 3

Quadruped robot kinematics

Kinematics is a pivotal aspect of robotics. It studies the motion of robotic systems without accounting for the forces causing the movement. Kinematics, given that the model is idealized relates rigid parts of the body with joints that move them. The joints may take different forms, such as revolute, prismatic, spherical, and so on. Understanding robot kinematics is crucial for designing and controlling articulated robots. The discipline of designing kinematic chains is important and strongly present in legged robotics, although wheeled robots have kinematic constraints and, therefore, problems of their own [16].

In robotics, one is mainly concerned with forward and inverse kinematics. Forward kinematics (FK) aims to provide a solution to the position of an end-effector, given the angles of the joints preceding the end-effector. The inverse kinematics (IK) aims to provide a solution to the joint angles given the position of the end-effector. The inverse kinematics is a more complicated question, mainly in the situation of multiple joints, where a subset of the joints is redundant, i.e., $n_{\text{joints}} > n_{\text{DOF}}$. The problem of inverse kinematics is usually independent of gait planning. I.e., the robot is tasked with an end-effector trajectory (points in space with a time-stamp that demand some position of a leg in space and time), and some solver of IK provides a solution, usually based on proximity of the solution to the current joint configuration. The solving of IK is a complicated subject and a discipline of its own, moreover if solved under specific constraints.

The dynamics are of great importance when the manipulator is not fixed at its base. In contrast to a classic 7 DOF robotic arm, a legged robot has a more substantial need for incorporating dynamics, to ensure stability and safety of the surroundings. A 7 DOF robotic arm may simply track the error in joint angles and, if needed, apply larger torques to minimize the end-effector position error. This is also possible in floating-base robots, although any fast movement with a heavy effector further away from the center of mass moves the whole robot in space in an unwanted manner. Therefore, even though the end-effector error may still be minimized in a floating-base robot, the whole robot may have inadvertently moved and may have introduced errors into other degrees of freedom, such as rotations around its pivotal axis. A control system that accounts for kinematics and dynamics can be created by meticulously engineering a modular solution for kinematics, dynamics, motion planning, and path planning or by trying to solve the kinematics, dynamics, and motion planning in one compound problem in the form of a policy using deep reinforcement learning.

3.1 Kinematics of quadrupedal robots

This section provides an overview of conventional approaches to building a legged robot, particularly a quadrupedal robot. One part of solving a quadruped's kinematics is providing formulas or algorithms to solve the FK and IK problems given a robotic actuator. Another part is actually designing the kinematic chain that will serve as the robotic actuator. In the following subsections, I provide different approaches to building the articulation chain. A quadruped robot usually consists of 12 degrees of freedom (DOF) with three DOF per leg. A general quadruped schematic is provided in Fig. 3.1. The motors/joints are usually regarded in alphabetical order from the base (B in the figure) to the end effector (E in the figure) as J_A, J_B, and J_C when talking about joints, or M_A, M_B, and M_C when pointing to motors. The orange dashed arrows depict the axis of rotation of the motor, and the spinning arrow signifies the positive direction of rotation according to the right-hand rule. The joints are sometimes referred to by the type of rotation they provide to the robot, i.e., the M_A would be a rolling motor, M_B and M_C would be pitching motors. In this thesis, regarding the Artaban platform, motors M_A are often written about as *rockers*, motors M_B as *shoulders* or *hips* and M_C as *knees* or *wrists*.

3.1.1 Fully revolute actuation chain

This is perhaps the most straightforward design of the articulation chain. For clarity, in this work, a revolute joint is understood as a joint with strictly one

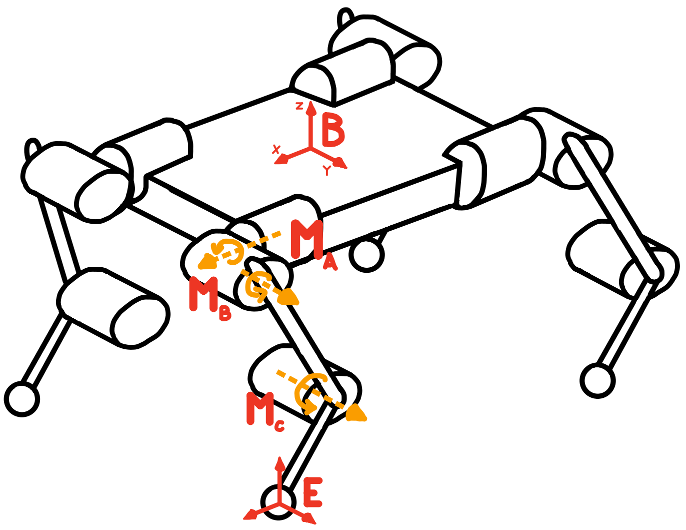


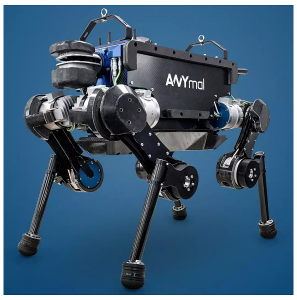
Figure 3.1: A general quadruped schematic

rotational axis. One revolute joint is at the forward-facing axis of the robot, and one or more revolute joints are on the robot's side-facing axis. The upside of this approach is that the design and manufacture of the robot parts and kinematic equations are straightforward. The downside is that the motors on each joint are often coupled with planetary transmission, encoders, and other accompanying components, all adding up to the weight of each leg and, therefore, moving the center of mass (CoM) of each leg. This, as mentioned in the introduction of this section, brings further problems while solving the gait because each movement of a relatively heavy point outside of the robot's center of gravity introduces unwanted forces. A concrete example of this joint configuration is the ANYmal robot series by ANYbotics. ANYmal is actuated using series elastic drive [17] in each joint, all encompassed in a similarly named actuator ANYdrive.

3.1.2 Mixed revolute-linear actuation chain

As mentioned in the previous subsection 3.1.1, mounting a motor with all its relevant components directly on the actuated joint might result in a massive leg with an undesirable mass distribution. Therefore, moving the heavy actuator parts closer to the robot's CoM is an advantageous step. On the other hand, this approach often brings restrictions on the range of motion since the driving power of articulated joints is transported using some mechanism to undriven mechanical joints. One approach to designing a mixed revolute-linear actuation is using a screw drive to actuate a lower joint, as

(a) : ANYmal with ANYdrives visible


(b) : ANYmal B with ANYdrives visible

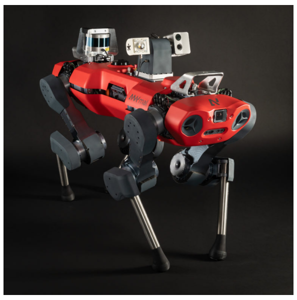

(c) : ANYmal C, not visible ANYdrives

Figure 3.2: Robots ANYmal by ANYbotics serve as a good example of a fully revolute actuated kinematic chain forming a leg. Newer models still adhere to the fully revolute actuation serving as a proof of robustness. The pictures are sourced from [1].

done in the robotic platform Spot by Boston Dynamics. The revolute motion of the motor is translated through the screw shaft that moves a carrier that moves the knee pivot, similar to the way the piston in a combustion engine is moved, i.e., the head of the piston (here the carrier - 414) is moved up and down, and the engine crankshaft (here the lower leg 412) rotates around the pivot (here the knee pivot - 426). The numbers in the former explanation correspond to those in the graphic in Fig. 3.3a, as published in the patent [18].

Another approach is transferring the torque from an electric motor through a belt instead of a screw drive. This has the advantage of easy manufacturing and installation but limited torque and risk of slippage. Such a belt mechanism is used to transfer the torque from an upper drive to another joint in the quadruped robot Cheetah by the Massachusetts Institute of Technology. This mechanism is also mentioned in this section because the actuation is transferred to another joint in a linear fashion, compared to the Anymal robot that has drives installed directly in the location of the joint itself. The upper drive is close to the CoM of the body, and the belt transfers the torque to an undriven mechanical joint, rotating the lower link of the quadruped leg. The belt mechanism is clearly visible in Fig. 3.3b. The Cheetah's degrees of freedom are driven by a motor-integrated gearbox drive [17].

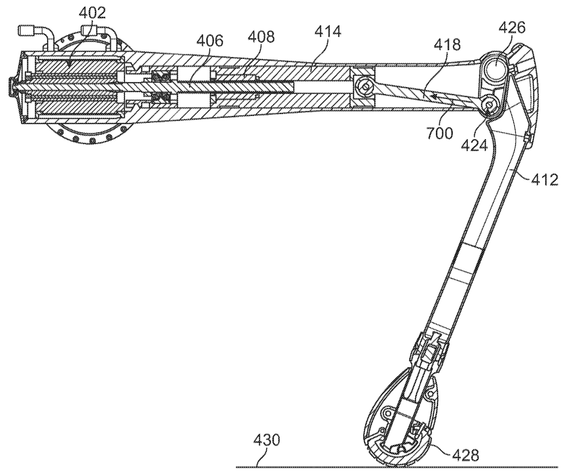
3.1.3 Mixed revolute-universal actuation chain

The approach of the platform Artaban by Panza Robotics is similar to the prior examples in the sense of the mixture of articulation chain elements. However, Artaban's legs consist of a unique patented mechanism based on a universal joint, often called the Cardano joint or Cardan joint. This joint, in its basic form, is quasi-spherical in the sense the joint can transfer axial rotation between two links under an arbitrary angle, see Fig. 3.4a. This joint is most known for its use in the driving shaft of automobiles with all four wheels drive to transfer torque from the engine to the other wheels opposite the engine location.

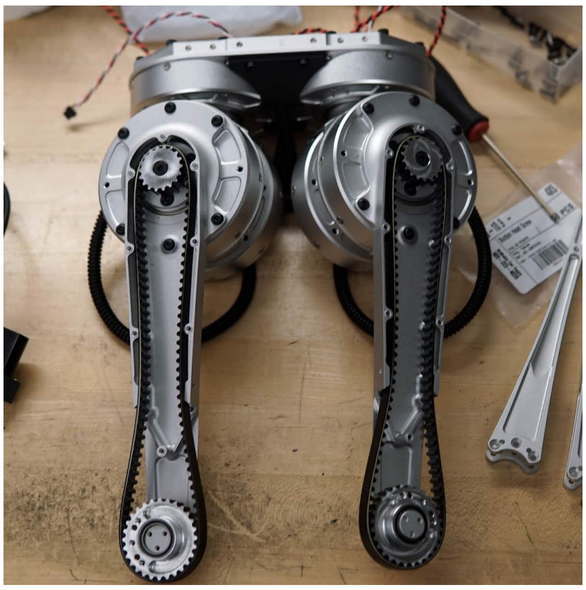
Each Cardan mechanism introduces a parallel four-bar linkage into the articulation chain. Front legs have one four-bar linkage surrounding the Cardan mechanism, while the rear legs have two foud-bar linkages. That is, one surrounding the Cardan mechanism and another one pulling the lower link through the connecting pantograph link. More detail is provided in the Chapter 5. The example of the

3.1.4 Closed-loop fully revolute articulation chain

Even though the previous subsections do not provide an exhaustive overview of possible kinematic solutions, the robot platform Minitaur is an honorable mention in its unique leg design. The whole articulation chain is closed-loop, in the sense that each leg starts and ends with a directly driven motor [17] attached to the body. The two motors are connected via a four-bar linkage. In the middle of this chain is the end-effector. This end-effector is offset in one dimension connected by another bar. Therefore, the author of [17] references the linkage as a five-bar. However, the author of the original mechanism states [22], that even though it is technically a five-bar mechanism when viewed from a side view, the actual linkage is four-bar in its geometry and offset in the depth dimension that is not visible when viewed from the side.

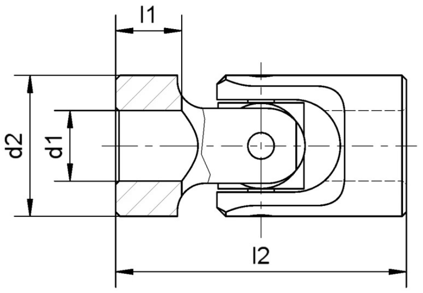


(a) : Screw drive actuation patent drawing by Boston Dynamics [18]

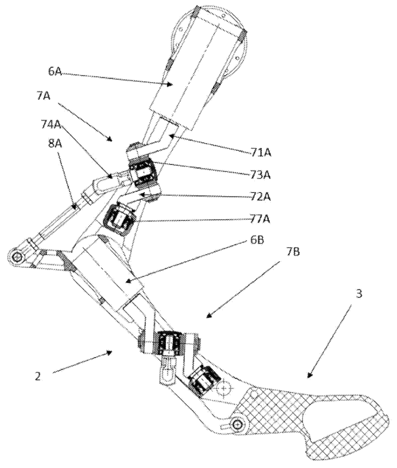


(b) : Assembly of the Cheetah robot's legs, showcasing the belt drive actuation [19]

Figure 3.3: Examples of a mixed revolute-linear actuation chain

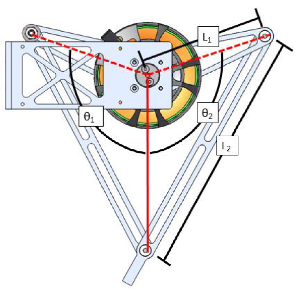


(a) : Exemplary showcase of the Cardano joint in the basic form, source: [20]

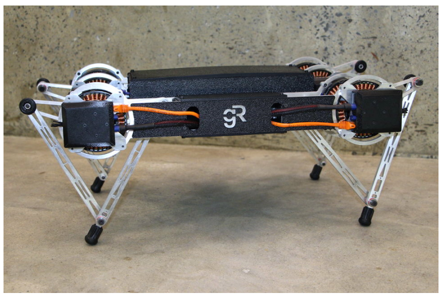


(b) : An example of usage of two Cardan joints in a robotic leg, patented by Panza Robotics s.r.o. [21]

Figure 3.4: Cardan joint use for robotic leg articulation



(a) : Showcase of the 5-bar linkage forming a closed-loop articulation, sourced from [22]



(b) : The robot Minitaur as constructed by Ghost Robotics [23]

Figure 3.5: The close-up and overall picture of Minitaur's closed-loop articulation chain by Ghost Robotics

Chapter 4

Reinforcement learning for learning quadrupedal gait

Reinforcement learning is a general paradigm in the realm of machine learning that formulates a learning strategy in which an agent learns based on interaction with its environment. Reinforcement learning is well suited for problems in which the task definition is relatively simple, but the task execution is complicated. The reinforcement learning tasks might be defined as *walk forward, win the game by maximizing score, keep a system variable steady,* etc. The discipline that aims to formalize the definition of task complexity into tractable format is often called *reward shaping.* Reward shaping puts deliberate penalties, rewards, and constraints on the agent's behavior in order to shape it in a desired direction.

4.1 Supervised, unsupervised and reinforcement learning

Supervised learning trains a classificator or other agent on a labeled dataset that defines a mapping from inputs to outputs. The goal is to learn this mapping. The objective is to generalize well on data that are out of the training distribution. Good examples of supervised learning are image classification, speech recognition, or other general classification problems. Supervised learning can also be used for regression problems where the output is not strictly a mapping from inputs to discreet outputs but a prediction of continuous values.

Unsupervised learning, in contrast, operates without the availability of a labeled dataset. The algorithm is used to discover patterns and structures in the data without prior insight. Unsupervised methods include generative models such as autoencoders, clustering techniques such as k-means, dimensionality analysis such as principal component analysis, and many more.

Reinforcement learning does not entirely fit any of those categories but rather is a category itself. It might be reasonable to categorize reinforcement learning as unsupervised learning because the agent learns by itself without the supervision of labels. However, the agent in the RL setting is provided with a reward signal that is absent in unsupervised learning. Therefore, one could view RL as supervised learning because the rewards aim to at least partly substitute the supervision of labels. However, this is not true either. Reinforcement learning aims to learn a strategy - a policy through time that differs from the act of prediction of the input-output setting of the supervised learning.

4.2 Introduction to deep reinforcement learning

R. S. Sutton and A. G. Barto, in their textbook Reinforcement Learning: An Introduction, define deep reinforcement learning (DRL) as a "group of nonlinear methods that include some form of an artificial neural network (ANN) trained by backpropagation and variations of the stochastic gradient descent (SGD)" [24, Chapter 9.12].

The usual approach to RL is collecting observations from an environment, creating state-action pairs, computing a value based on the value function and then storing the values in a tabular fashion. An example of this RL framework is Q-learning, Value and Policy iteration, state-action-rewrad-state-action tuple called SARSA, etc. Reinforcement learning is often split in its taxonomy into model-based and model-free branches [25]. The model in question is the model of the environment. In designing a RL system, we have to decide whether it is beneficial or imperative for the agent to learn the model of the environment.

In model-based RL, the model of the environment is represented by the state transitions and rewards. An example of model-based RL might be to learn the Model representation for a model predictive control (MPC) or Expert iteration.

Model-free frameworks are usually represented by a Policy optimization method or a Q-learning method. In policy optimization, the agent learns the policy as $\pi_\theta(a|s)$ whereas in Q-learning the agent learns the policy in a form of an approximator $Q(s, a)$ directly. This is mot suited for discrete environments. For this thesis, the policy optimization method is of greater importance than Q-Learning. Policy Optimization Policy optimization methods optimize θ by gradient ascent/descent directly on the objective function $J(\pi_\theta)$ or maximize the local approximation of $J(\pi_\theta)$ [25]

Actor-critic framework signifies a family of actor-critic methods where a so-called *advantage* plays a key role in the training process. Actor-critic methods combine the basic notions of policy and value learning. The policy is responsible for choosing actions for the agent, and the value function is responsible for criticizing this choice. Therefore, the policy is referred to as an actor, and the value function as a critic, hence the name actor-critic methods. [24, Chapter 13.8].

The advantage of a state-action pair is the difference between the state-action pair value and the state's value function. More formally as

$$A(s, a) = Q(s, a) - V(s). \tag{4.1}$$

The value function $V(s)$ represents the value of the state in the environment before the actor takes an action, and $Q(s, a)$ represents the value of the state after action a is taken. The meaning of the advantage function in eq. 4.1 is that when the agent is in a favorable state ($V(s)$ is large) and takes an unfavorable action ($Q(s, a)$ is small) then the advantage is small, possibly negative, and vice versa. However, the calculation of $Q(s, a)$ is a problem of its own. Therefore, the estimation of the reward at the end of the episode is used to approximate the real value of Q as the sum of the expectation of the value of $V(s_{t+1})$ and the reward of a given action-state pair as

$$Q(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1}}[V(s_{t+1})]. \tag{4.2}$$

Therefore the advantage from eq. 4.1 may be rewritten [26, Page 8] with additional subscripts for clarity as

$$A(s_t, a_t) \approx r(s_t, a_t) + V(s_{t+1}) - V(s_t). \tag{4.3}$$

This approximation of advantage, as stated above in eq. 4.3 may then be used as the surrogate advantage for policy optimization denoted in eq. 4.21 in the next section on Proximal Policy Optimization. In the framework of actor-critic methods, policy estimation and value estimation are both learned simultaneously. The difference between the rewards the agent collects during the episodes and the expected rewards in the form of the agent's learned value function is how the actual advantage is calculated for every episode. Advantage, therefore, helps to identify the important actions that maximize the reward throughout the episodes [27, Chapter 12].

4.3 Proximal Policy Optimization

In this work, Proximal Policy Optimization is used to optimize the policy of the deep reinforcement learning setting. Although the work is not focused on the theoretical foundations of the PPO algorithm, I provide a whole section that delivers a medium insight into the optimization strategy of choice.

4.3.1 Introduction to policy gradient methods

In the reinforcement learning framework, an agent interacts with the environment, collects observations, and receives rewards as a function of those observations. The rewards are a proxy function for a goal we want the agent to pursue. Therefore to pursue the goal, agent must maximize the collected reward. This notion was described by Richard S. Stutton and Andrew G. Barto in the canonical book Reinforcement Learning: An introduction, second edition [24] as *The reward hypothesis:*

> That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

The Proximal Policy Optimization algorithm (PPO), as defined by J. Schulman et al. [28] is a family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment and optimizing a so-called surrogate objective function using stochastic gradient ascent. To understand better this widely used method in the Reinforcement Learning field, we must first look at the method's foundations. The main underlying concept is the policy gradient method, which is also employed in popular methods such as Deep Deterministic Policy Gradient (DDPG), REINFORCE, Twin Delayed Deep Deterministic (TD3), Trust Region Policy Optimization (TRPO), and many others. TRPO had a major influence on the state-of-the-art PPO that is still considered state-of-the-art despite the defining paper being published in 2017. This underlines the importance of the method, which has supported the RL frameworks for more than 6 years and still continues to do so. As of writing this thesis, the PPO has vastly outnumbered other policy gradient methods' defining papers in the number of citations, according to Papers With Code. [29]

Policy gradient method

In order to define the policy gradient method, we need to define some useful reinforcement learning concepts, such as *policy* π_θ, *MDP*, *on-policy* vs. *off-policy methods*, *reward function* vs *objective function*, etc.

A *policy* π is a mapping from perceived states of the environment to actions taken when inhabiting those states. It is the cornerstone of reinforcement learning and may range in complexity from a simple lookup table to complex functions. In Markov decision process (MDP) understanding, a policy is a *mapping from states to probabilities of selecting each possible action* [24]. A policy for an MDP is written formally as

$$\pi(a|s), \tag{4.4}$$

where a represents actions and s represents states. The policy, therefore, represents the probability of choosing any action a given any state s.

A *Markov decision process* (MDP), usually understood as a finite Markov decision process, formalizes a sequential decision-making problem in which the Markov property applies for the state transitions. Markov property states that the history of taken actions does not influence the probability of choosing the next action because the probability of choosing the next action is only dependent on the current state. More formally, the next state s' depends on the choice of action a from within the state s.

An *on-policy* method evaluates and attempts to improve the same policy that delivered the observations, in contrast to an *off-policy* method, that aims to improve its policy that was not used to deliver the observations. In other words, the on-policy method judges and improves itself based on its own data, whereas the off-policy method judges and improves itself on data generated by some other source.

A *reward function* is understood as a mapping from state-action pairs (s, a) to a scalar reward r. Reward function, as opposed to an objective function, defines the relation of actions and individual rewards, whereas an *objective function* defines the optimization criterion that defines the computation of the cumulative reward over time, usually including some discount factor γ. An example of an *objective function* can be formulated as

$$\sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{4.5}$$

where s, s' are the current state and some next state, a, r are current chosen action and the associated reward, γ is a discount factor and $v_\pi(s')$ is the value of being in the state s' given some policy π. An objective function as stated above 4.5, is the result of the *Bellman optimality equation* that unravels the optimal value equation into its cardinal terms as follows:

$$
\begin{aligned}
v_*(s) &= \max_{a\in\mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t|S_t=s, A_t=a] && (4.6) \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1}+\gamma G_{t+1}|S_t=s, A_t=a] && (4.7) \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1}+\gamma v_*(S_{t+1})|S_t=s, A_t=a] && (4.8) \\
&= \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_*(s')] && (4.9)
\end{aligned}
$$

where G_t is discounted reward at timestep t, R_t is reward at timestep t, v_* is the optimal policy, $q_{\pi_*}(s,a)$ is the optimal action-value function, the steps (4.6) and (4.7) represent the unrolling of the discounted reward as

$$
\begin{aligned}
G_t &\doteq R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\dots \\
&= R_{t+1}+\gamma(R_{t+2}+\gamma R_{t+3}+\dots) \\
&= R_{t+1}+\gamma G_{t+1},
\end{aligned}
$$

and the steps (4.8) and (4.9) state the Bellman optimality equation according to [24, Chapter 3.6]. This covers the basic theory needed to build on towards policy gradient methods, eventually summarizing the main points of the PPO algorithm.

A typical objective in reinforcement learning could be defined as *maximizing the expected reward over a trajectory while following a parametrized policy*. The trajectory may represent different notions in different disciplines, like planning, reinforcement learning, physics, etc. In the context of reinforcement learning, trajectory represents the whole agent vector for a set of time stamps. The agent vector is a custom concatenation of values, such as body rotation, velocity, and acceleration, optionally a contact vector, actuator positions, laser scans, etc. This vector of relevant information is usually regarded to as an observation, or an observation vector o. A trajectory τ describes a set of observations over time. Therefore, the objective function may be written as

$$J(\theta) = \mathbb{E}_\pi[r(\tau)]. \tag{4.10}$$

If we adhere to simplifying the actual problem to a finite MDP, we may state that the MDP has at least one optimal policy that gives the maximum reward and is stationary and deterministic [24, Chapter 3.6]. In order to find the set of the optimal parameters θ^* that maximizes the objective function, we may employ a set of optimization strategies, namely some form of gradient descent/ascent. The basic form of gradient ascent is

$$\theta_{t+1} \leftarrow \theta_t + \alpha\nabla J(\theta_t). \tag{4.11}$$

Using the *Policy gradient theorem* [24, Chapter 13.2] that basically applies the log-derivative identity

$$\frac{\nabla_\theta \pi_\theta}{\pi_\theta} = \nabla_\theta \log \pi_\theta \tag{4.12}$$

which does not look obvious in this formulation, but it does in the formulation of scalar logarithm derivative

$$\frac{\frac{\partial}{\partial x} f(x)}{f(x)} = \frac{\partial}{\partial x} \log f(x). \tag{4.13}$$

When applied to the gradient of the objective function 4.10 we get the following:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\pi_\theta}[r(\tau)] &= \nabla_\theta \int_{-\infty}^{\infty} \pi_\theta(\tau) r(\tau) d\tau && (4.14)\\
&= \int_{-\infty}^{\infty} \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau && (4.15)\\
&= \int_{-\infty}^{\infty} \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau && (4.16)\\
&= \mathbb{E}_{\pi_\theta}[r(\tau) \nabla_\theta \log \pi_\theta(\tau)] && (4.17)
\end{aligned}$$

At the beginning of this section, we defined trajectory τ as a set of observations over time. One time step on a trajectory is understood as a transition from one state to another. The probability of a specific trajectory is therefore a product of states taken to reach the end state of the trajectory. In the log-likelihood form, the product transforms into a sum, therefore we can write

$$\nabla_\theta \log \pi_\theta(\tau) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t). \tag{4.18}$$

Using the result (4.18) above and substituting it into the Policy gradient theorem (4.17) we get

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[r(\tau)] = \mathbb{E}_{\pi_\theta}[r(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t)] \tag{4.19}$$

which shows that now all terms are dependent on θ, except the reward function $r(\tau)$ which is still dependent on the taken trajectory τ. This is the only "expensive" problem, in the sense that the solution to a given problem via the policy gradient method is directly dependent on how costly it is to sample the trajectory reward function. The most basic use of policy gradient method is in the historically significant method REINFORCE, which directly utilizes the result (4.19) by approximating the long-term result of the reward function $r(\tau)$ with the discounted reward G_t at each step t as

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[r(\tau)] = \mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} G_t \nabla_\theta \log \pi_\theta(a_t | s_t)]. \tag{4.20}$$

A whole part is dedicated to this result in Sutton's et al. publication [24, Chapter 13.3]. However, a small clarification must be made for the interested reader: the authors chose the usage of a more fractional notation containing the likelihood ratio instead of the logarithm notation. In other words, they use the left part of the identity (4.12) rather than the right side.

Trust region policy optimization (TRPO)

To explain the PPO method we shall first go through the main points of Trust region policy optimization (TRPO) by Schulman et al. [30]. This publication brings the usefulness of trust region from gradient descent/ascent into the policy updates. Schulman et al. describe "an iterative procedure for optimizing policies, with guaranteed monotonic improvement." The trust region generally describes a neighborhood of policy or another object of optimization in its abstract optimization space. The trust region is constructed based on a step size/radius within which the policy update is still acceptable. This is computed based on so-called *advantage function* A_π and KL divergence of current and proposed policy distribution. The advantage function holds a similar meaning to the expected discounted reward G_t from previous sections, for example, in (4.20). It is constructed as the difference between the state-action value function Q_π and the value function V_π

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s). \tag{4.21}$$

The advantage is then used to compute the expected return of the new policy based on the advantage of the previous policy, using the Kakade & Langford identity [31] as

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tilde{\pi}}[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)] \tag{4.22}$$

where $\eta(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t)]$. Next in the publication follow various definitions of terms such as *discounted visitation frequencies* $\rho_\pi(s)$, local approximation of η using ρ_π, noted $L_\pi(\tilde{\pi})$, theorem for the relationship between total variation divergence D_{TV} and the Kullback-Leibler divergence D_{KL}. That results in the formulation of the TRPO constrained optimization problem

$$\pi_{i+1} = argmax_\pi[L_{\pi_i}(\pi) - CD_{KL}^{max}(\pi_i, \pi)] \tag{4.23}$$

where the C is a constant constructed from analytical parameters [30].

4.3.2 Proximal Policy Optimization method (PPO)

The PPO method was defined in the introduction of this section as a gradient method optimizing a surrogate objective function. That is true, although not exclusive to PPO. TRPO also optimizes some surrogate/proxy objective function, and while doing so, it calculates a set of analytical parameters and in the numerical optimization steps, uses the inversion of the Fisher information matrix (numerical approach to the optimization problem in 4.23 was not discussed for brevity). This is not only costly from the point of inversion of a large matrix, but also from the point of computing the Fisher information matrix which consists of second-order derivatives. Therefore, using strong first-order stochastic optimization tools like Adaptive Moment Estimation (ADAM) is not possible.

The PPO implementation drifts away from the usage of second-order derivatives and compared to the TRPO, loses some analytical rigor due to introducing heuristics rather than analytical parameters. Although the cost of analytical rigor comes with a boost in overall performance [28, Section 6.2]. PPO comes in two versions, as proposed by the author [28]. Either PPO-Clip with a clipped surrogate objective function or PPO-Adapt with an adaptive Kullback-Leibler divergence penalty. An important aspect of PPO and TRPO is that updating the policy "slowly and carefully" is even more important than in the case of supervised learning. In supervised learning, if a policy update is too large and therefore degrades the prediction accuracy, the next training episodes will correct for this problem because the data is independent of the network parameters. In the case of reinforcement learning, a policy update that is too large may produce undesirable performance of the policy. The agent with the undesirable policy in return may fail to collect useful observations and, therefore, fail to recover from this local extreme. This briefly illustrates the stark need for smart policy updates.

PPO-Clip

PPO-Clip uses a clipping of surrogate advantage (expected improvement in the policy) which is based on a hyper-parameter and, therefore does not block the backpropagation. This means that the optimization function is differentiable and of the first order, which provides a significant efficiency boost. A surrogate objective function L, compared to the previously described reward objective function J, optimizes a given probability ratio not cumulative reward. Authors denote this ratio r which is ambiguous with the notation of *reward over trajectory* $r(\tau)$ as it is not directly the reward meanwhile using the same notation. The notation of choice here will be as similar as possible, R. Schulman et al. describe the clipped surrogate objective as

$$L^{CLIP}(\theta) = \mathbb{E}_t[min(R_t(\theta)A_t, clip(R_t(\theta), 1-\epsilon, 1+\epsilon)A_t]) \tag{4.24}$$

where $R_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t$, ϵ is a hyperparameter proposed as $\epsilon = 0.2$. The ratio $R_t(\theta)$ provides a weight for the advantage function A_t, similar to the expected discounted reward in TRPO. $R_t(\theta)$ clipping is done from both sides, depending on whether A_t is positive or negative, up to $1 \pm \epsilon$.

PPO-Adapt

PPO-Adapt is another approach that can be used instead of clipping, but may also be used as an addition. The main principle is limiting the rate of the KL divergence of the policy distribution before and after the policy update by introducing a penalty coefficient for the KL divergence and adapting the coefficient to achieve a target divergence value, denoted as d_{targ} by the authors. The surrogate objective in this case is as follows:

$$L^{ADAPT}(\theta) = \mathbb{E}_t[R_t(\theta)A_t - \beta D_{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \tag{4.25}$$

where the dot in the notation $\pi(\cdot|s_t)$ represents $\pi(\cdot)$ as a function itself given some state s_t, unrelated to a specific value. To update parameters d_{targ} and β we do

$$\begin{aligned} d &= \mathbb{E}_t[D_{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \\ \text{if } d &< d_{targ}/1.5, \beta \leftarrow \beta/2 \\ \text{if } d &> d_{targ} \times 1.5, \beta \leftarrow \beta \times 2 \end{aligned}$$

where values 1.5 and 2 for parameter updates were chosen heuristically and do not matter much in the long run [28, Chapter 4].

4.3.3 Popular PPO implementations

The popular implementations of PPO are `rsl_rl` [13], `skrl` [32], `stablebaselines3` [33], `OpenAI Spinning Up` [25], `CleanRL` [34] and `rl_games` [35] which is the implementation used in this work. Those repositories differ mainly in the implementation details and the training backends (Torch, JAX, TensorFlow, etc.).

Chapter 5

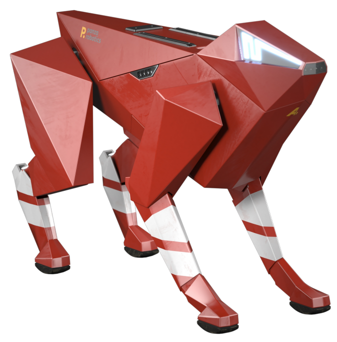
Materials & Methods

5.1 Robotic platform Artaban

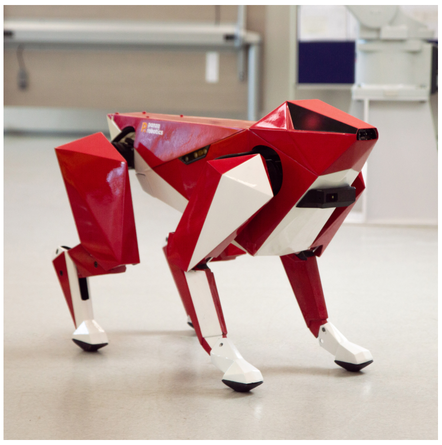
The platform used for this thesis is Artaban v0.2 by Panza Robotics, s.r.o. The word platform is a deliberate choice as the main goal of this robot, currently, is to serve as a mobile sensor. Plans for widening the repertoire of the platform with manipulation actions via either an additional robotic arm on its back or with its own body (holding doors with the leg, etc.) are already in the works. The platform has four built-in camera modules that serve internal navigation purposes. One camera module is in the front of the robot, two on the sides, and one in the back. Each module consists of two evenly spaced RGB cameras for stereovision and one infrared depth sensor, together working in a configuration generally known as an RGB-D camera. The robot is shown in Fig. 5.1b, wearing yet another Time-of-flight (ToF) camera module in its *"dog collar"*, strictly for foot placement purposes.

5.1.1 Platform description

Artaban, the robotic platform, is a four-legged robot with 12 degrees of freedom (DOF). Even though the robot has 12 DOFs, the overall number of joints is 36, 9 for each leg. With 3 degrees of freedom and 9 joints per leg, it is imperative that the rest of the joints exist only as some form of torque transfer. The unactuated joints are moved according to kinematic constraints



(a) : Concept render of Artaban robot in red color



(b) : The second physical prototype Artaban v0.2

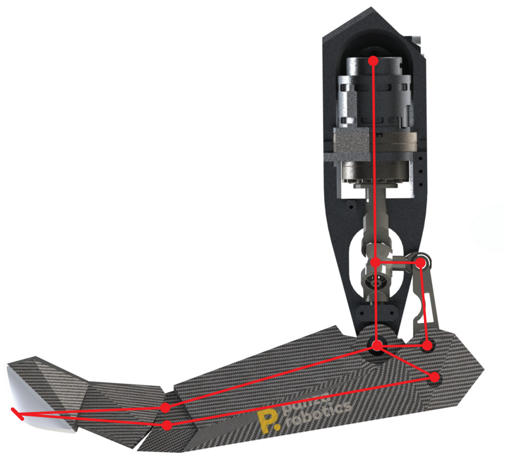
Figure 5.1: Artaban robotic platform by Panza Robotics, s.r.o.

and according to the motion of the actuated joints. Each leg is constructed out of 3 segments: upper, middle, and a lower segment.

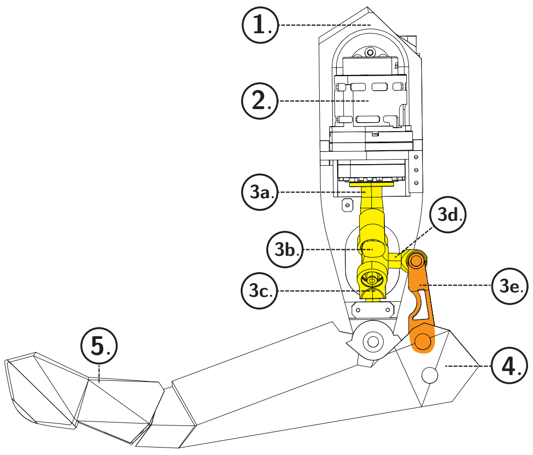
Front leg

The front leg consists of a total sum of 9 joints, out of which 8 are shown as red circles in Fig. 5.2a. The ninth joint moves the whole leg to or from the body. Presently, and for the scope of this thesis, the middle and lower segments of the front leg (labels 4. and 5. in Fig. 5.2b) are rigidly fixed together, creating a stiff link without moving joints.

The reason for picturing the joints between the middle and lower segments is that the front leg is planned to contain a linear actuator that would also articulate the lower link. The actuation would be somewhat similar to a human wrist or dog paw. The elbow joint that connects the upper and middle link (labels 1. and 4. respectively in 5.2b) is physically constructed out of two joints but can be modeled as one. The Cardan joint, represented in yellow color, transfers the angular motion of the motor (label 2.) through the drawbar in orange color (label 3e.) to the lower link in a quasi-sinusoidal motion. This, as a result, moves the middle and lower link together (labels 4. and 5. respectively) up and down.



(a) : Red lines show the chain of articulation links and joints over a photorealistic visualization



(b) : A blueprint of the front leg. Labels: 1) upper segment, 2) motor, 3.a) motor yoke, 3.b) cross joint, 3.c) counter piece yoke, 3.d) clevis fork, 3.e) drawbar, 4) middle segment, 5) lower segment

Figure 5.2: Front leg of the Artaban robotic platform

5.1.2 Rear leg

The rear leg similarly consists of a total sum of 9 joints, out of which 8 are shown as red circles in Fig. 5.3a. The ninth joint moves the whole leg to or from the body. However, in contrast to the front leg, the rear leg has two sets of four-bar linkages, one in red color and one in orange color. The front leg has the middle and lower segments fixed together, compared to the rear leg where the middle and lower segments (5. and 6. respectively in Fig. 5.3b) move according to kinematic constraints. The reason for the different coloring of the two four-bar linkages in Fig. 5.3a is that the red one can be modeled with a quasi-sinusoidal model, whereas the orange one needs additional computation based on the current model of inverse kinematics. This orange four-bar linkage creates a closed loop in the articulation chain that brings computational difficulties when simulated in a physics simulator.

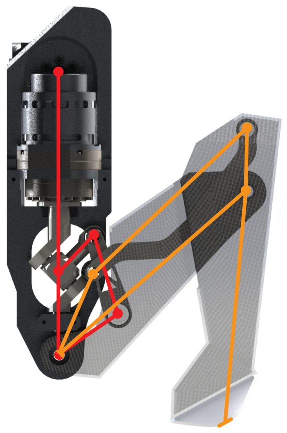
5.1.3 Cardan mechanism detail

The Cardan joint was already briefly mentioned in the section 3.1.3 and at the beginning of this chapter as a quasi-spherical joint. That is true in the use case of automobiles, where this joint must transfer torque from the engine to the rear axle under different angles relative to the compression of the rear suspension of the car. However, in our robot, the setting is inverse. The rotation of the motor spins the Cardan mechanism around and moves the clevis fork in a sinusoidal motion up and down. The motion closely follows a cosine approximation in the form

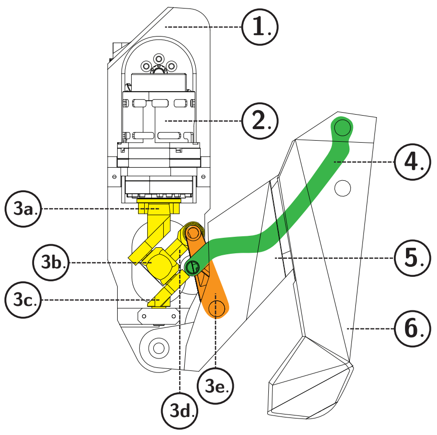
$$\phi_c = \frac{\pi}{2} + \frac{\pi}{4}\cos(\vartheta_m) \tag{5.1}$$

where ϕ_c is the angle of the clevis fork and ϑ_m is the angular position of the motor. The resulting angle of the knee follows equations relating to the four-link mechanism. Those equations are mentioned in the Appendix A.

The Cardan, in this and a few different configurations, is a patented mechanism under Slovak and International patent law [21]. The mechanism and legs configuration are shown in Fig. 5.5.

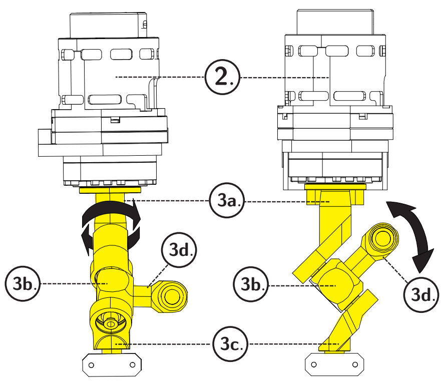


(a) : Red lines show the chain of articulation links and joints over a photorealistic visualization

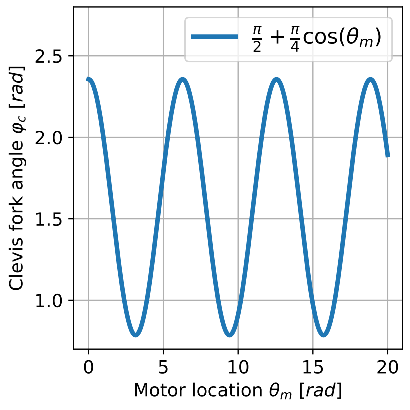


(b) : A blueprint of the front leg. Labels: 1) upper segment, 2) motor, 3.a) motor yoke, 3.b) cross joint, 3.c) counter piece yoke, 3.d) clevis fork, 3.e) drawbar, 4) pantograph, 5) middle segment, 6) lower segment

Figure 5.3: Artaban robotic platform by Panza Robotics, s.r.o.

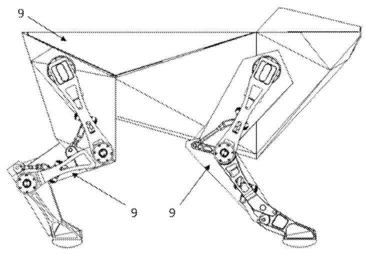


(a) : Visualisation of the resulting motion of clevis fork when the motor shaft spins. The motor spinning (left side) results in the motion of the clevis fork up and down (right side)

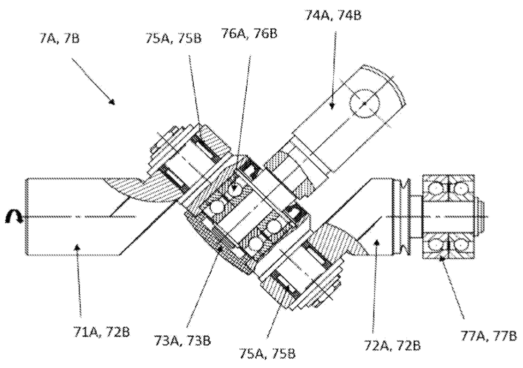


(b) : The cosine approximation of the motion transfer from the motor (2) to clevis fork (3.d)

Figure 5.4: The Cardan mechanism with the notation: 2) motor, 3.a) motor yoke, 3.b) cross joint, 3.c) counter piece yoke, 3.d) clevis fork



(a) : A snippet from the patent [21] picturing the configurations of front and rear legs using the Cardan mechanism



(b) : A snippet from the patent [21] picturing the actual Cardan mechanism used on the robot.

Figure 5.5: Snippets picturing the configurations in the original patent filed by Panza Robotics s.r.o.

5.2 Simulation environment

In reinforcement learning, a simulation environment serves as a virtual platform where agents learn to perform tasks or make decisions. It's a controlled setting where agents interact, taking actions and receiving feedback in the form of states and rewards. It is desired that the simulator models reality as closely as possible.

When choosing a simulation environment, one must review its priorities and demands on the simulation pipeline. Until recently, this meant choosing between robustness and ease of use. The choice of a simulation environment for this RL task is highly motivated by factors such as parallelization, GPU-accelerated simulation, user-friendliness, availability, etc. The most used simulators are CoppeliSim, ROS with Gazebo, Pybullet, MuJoCo, RaiSim, and NVIDIA IsaacGym / IsaacSim. Using a ranking approach by Kim et al., [36], IsaacGym is the best, followed by RaiSim, Unity ML, and Pybullet. It is important to note that in the year of publishing the report by [36] (2021), MuJoCo was still closed-source with quite expensive licensing, which changed when DeepMind acquired the simulator engine and made it publicly available. IsaacGym also supports GPU-Accelerated Simulation of environment interaction, which enables major speed-ups in training, on top of parallelization and GPU acceleration of the backpropagation processes.

The authors use different simulator traits to assess each platform's quality and overall usefulness. All of the traits are summarized into a *Total Score*. The traits are given as follows:

- **Reproducible**: Capability to yield the same results consistently for the same experiments.
- **Parallel**: Simultaneous collection of rollout data from multiple simulation environments.
- **Photorealistic Rendering**: Graphic rendering of the environment aiming for realistic photography, often used for training of robotic vision.
- **Accelerated**: Faster or slower simulation than real-time.
- **Modular**: Component-based SW architecture. Good for code reuse and integration.
- **GPU-Accelerated Simulation**: Fast simulation based on GPU acceleration. It allows collecting the rollout data fast without a CPU cluster and maximizes the performance by removing CPU-GPU data exchange.
- **Open Source**: The availability of the complete source code on the internet, ready for personalization and further development by anyone.

The authors provide a ranking of those simulation environments in the Fig. 5.6.

The top three are IsaacGym, Unity ML, and RaiSim. Unity ML is notoriously known for its heavy learning curve due to the prerequisite of learning

Simulation Environment	Reproducible	Parallel	Photorealistic Rendering	Accelerated	Modular	GPU-Accelerated Simulation	Open Source	Total Score
ROS & Gazebo				✓	✓		✓	1.55
CoppeliaSim	✓			✓				2.0
Pybullet	✓	✓		✓			✓	3.4
MuJoCo	✓	✓		✓				3.0
RaiSim	✓	✓	✓	✓				3.8
IsaacGym	✓	✓	✓	✓		✓		**4.8**
Surreal	✓	✓		✓				3.0
Unity ML	✓	✓	✓	✓				3.8

Figure 5.6: A comparison and ranking of the most well-known simulators

the workings of the Unity engine and its API. RaiSim is a simulator from the workings of ETH Zürich. They officially provide the simulator on request for educational purposes; however, my request was repeatedly left without an answer. Therefore IsaacGym is the best candidate in all directions. The IsaacGym is currently discontinued as a standalone application (*Isaac Gym Preview*; since 2023, it comes as a part of the Isaac Sim Omniverse family by NVIDIA.

5.2.1 Omniverse & Isaac Sim

The simulation environment of choice is, therefore, Isaac Sim. The nomenclature surrounding the Omniverse family is somewhat confusing, therefore, I follow with a brief overview. The Omniverse by NVIDIA itself is a concept that refers to the all-encompassing general omniverse which is a philosophical concept. A universe so big, there is none greater. The NVIDIA Omniverse is a platform for the creation and simulation of 3D worlds with an emphasis on easy shared access. Every Omniverse module works with a common file format called *OpenUSD* [37], an acronym for Open Universal Scene Description. The platform provides a number of *connectors* for accessing third-party modules, such as the *Unity engine*, 3D modeling programs by *Autodesk*, popular architecture CAD program *ArchiCAD*, *Houdini*, the *Adobe Suite*, popular *Blender* 3D editor and many more. The majority of those connectors are mainly useful for game developers and graphics enthusiasts. The most relevant extension for this work is the *Isaac Sim*.

5.2.2 Isaac Sim

NVIDIA built their own physics engine, called *PhysX* [38], which is the default engine throughout the Omniverse. An important property of the

PhysX engine is This simulator has “essential features for building virtual robotic worlds and experiments. It provides researchers and practitioners with the tools and workflows they need to create robust, physically accurate simulations and synthetic datasets” [39]. Furthermore, it provides ROS/ROS2 communication interfaces, sensor integration such as RGB-D cameras or Lidar, domain randomization, segmentation, and more. The developer’s choice was to enable the use of Isaac Sim only for NVIDIA RTX cards that are best suited for efficient ray tracing since the simulation network aims to be photorealistic. The simulator can be run locally or in the cloud. For the local setup of Isaac Sim, one needs a CUDA-enabled RTX card with as much VRAM as possible; ideally, more than 6 GB. 16 GB of computer RAM is the viable minimum.

Isaac Sim has the ability to communicate with multiple extensions that are either production-made or provided by the community. For the future possible adaptation of this work, the *replicator, occupancy map, contact and range sensor* extensions are quite useful. The replicator allows the creation of similar data to a provided dataset or generates data on demand, given specifications. This is mostly useful for simulating production pipelines to train mobile or static robotic manipulators in the simulation. A general Isaac Sim scheme can be seen in Fig. 5.7.

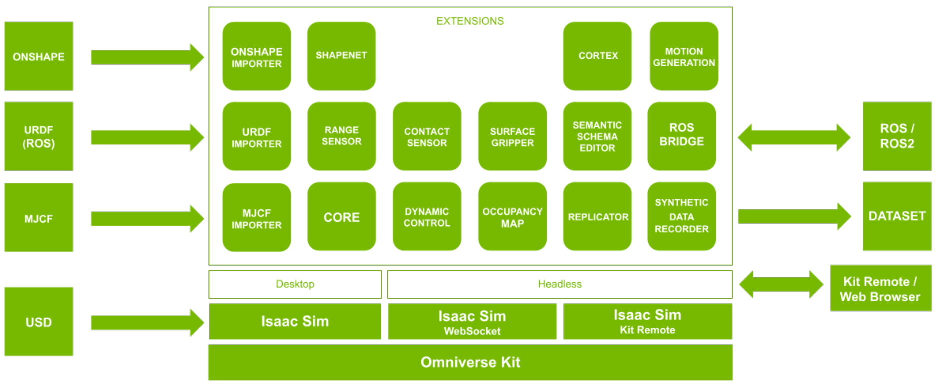


Figure 5.7: A non-exhaustive visualization of the Isaac Sim relationship with the extensions, inputs, outputs, and bridges

An important feature of Isaac Sim for this thesis is that it supports closed-loop articulations for robotic manipulators. Other names for closed-loop articulation might be parallel articulation, parallel robot, or parallel mechanism. Other simulators support importing robot meshes and other primitives in formats such as URDF or other XML-based formats. Those formats require that the robot’s articulation is defined in a tree structure, therefore making closed loops impossible on import. Popular ROS-native open-source simulator Gazebo allows closed-loop linkage with special add-ons complementing the URDF syntax, resulting in an *SDF*, an abbreviation for the Simulation Description Format. Although this format is native for Gazebo, it is possible to use bridges between Gazebo and Isaac Sim to share meshes directly [40], although there are differences in default frame orientations for meshes which complicate any transfer. Therefore, the URDF of our robot is loaded into the standalone Isaac Sim, where the linkage is closed, and the robot is saved as a USD file for further work. This is, in theory, quite straightforward, although the incompleteness of the documentation by NVIDIA makes any learning process taxing. That is caused mainly by the fact that the production of the Omniverse tools is still ongoing and still a work in progress. Isaac Gym is an implementation of the OpenAI gym format [41], that utilizes the backend of Isaac Sim, mostly the NVIDIA-made physics engine PhysX for collisions and world simulation. The labeling of the Gym environment for reinforcement learning has changed multiple times throughout the development process. It is generally regarded as Isaac Gym, although first, there was a standalone implementation called Isaac Gym Preview, which was used to create the influential paper exercising the massive parallelism Isaac Gym offers [13]. This framework was standalone and limited in API. The whole Isaac Gym implementation was recently integrated into the Isaac Sim in the form of an extension and, as of November 2023, is named Omniverse Isaac Gym or, in short, OmniGym.

5.3 Gym convention

The general Gym format was pioneered by OpenAI in 2016 as a "toolkit for developing and comparing reinforcement learning algorithms" [42]. This led to an overall standardization in how the general and academic public approached reinforcement learning and made benchmarking of algorithms strikingly easy. OpenAI handed the versioning and further development of the original Gym to an outside team; therefore, the current correct name for Gym is Gymnasium, but Gym will still be used for brevity.

5.3.1 Definition of Gym environment

The Gym environment is the cornerstone of the Gymnasium API framework. This environment encompasses the main loop of the reinforcement learning paradigm and provides a platform-wide standardization of the training pipeline. The Gym environment is simulator-agnostic. The most important API [41] methods are:

1. `step()` method updates the simulation by applying chosen actions, stepping the simulation, returning observations, rewards, information

whether the environment has terminated or truncated due to the latest action, and optional debugging data.

2. `reset()` method restarts the environment to the original state for the next rollout.

3. `render()` method renders the environment for the human viewer or a virtual camera export, for example, in the RGB array format.

4. `close()` method that signalizes the environment is being closed because of internal reasons, such as finishing the training or the occurrence of an exception.

Another required part of the gym format is defining the `observation_space` and `actions_space` for purposes of normalization and asserting whether data observed in or input to the simulation are in a correct range.

5.3.2 Omniverse Gym for Reinforcement Learning

NVIDIA provides its own Gym extension that builds on top of the Gym standard. Since reinforcement learning algorithms involve collecting large amounts of data and a great deal of policy updates, the need for parallelization arises very quickly. While policy updates can be efficiently parallelized on the GPU, parallelizing data collection is more challenging. Traditional pipelines handle simulation and reward/observation calculation on the CPU, limiting the potential throughput of GPU to policy inference only, due to communication bottlenecks. The usual way of collecting observation data and computing the physics for the environment was, until recently, done on CPU time. PCIe data transfer can be significantly slower than GPU processing. Therefore, collecting data from the environments in a serial manner on the CPU and then transferring the data from the memory to GPU for policy optimization is the slowest point in the deep reinforcement learning pipeline. The pipeline is illustrated in Fig. 5.8. Using multiple CPU cores and processes helps to mitigate this issue but is limited by the number of cores and RAM, ultimately using the serial processing power for a task that is in nature parallel. Multiple environments collect data independently; they do not affect each other. Omni Isaac Gym offers extensive parallelism, enabling GPU-based data collection and policy updates, reducing data copying, and making the simulations drastically more sample-efficient.

NVIDIA also provides the OmniIsaac Gym Environments (OIGE) [43] repository that contains examples of different RL tasks using the Gym API

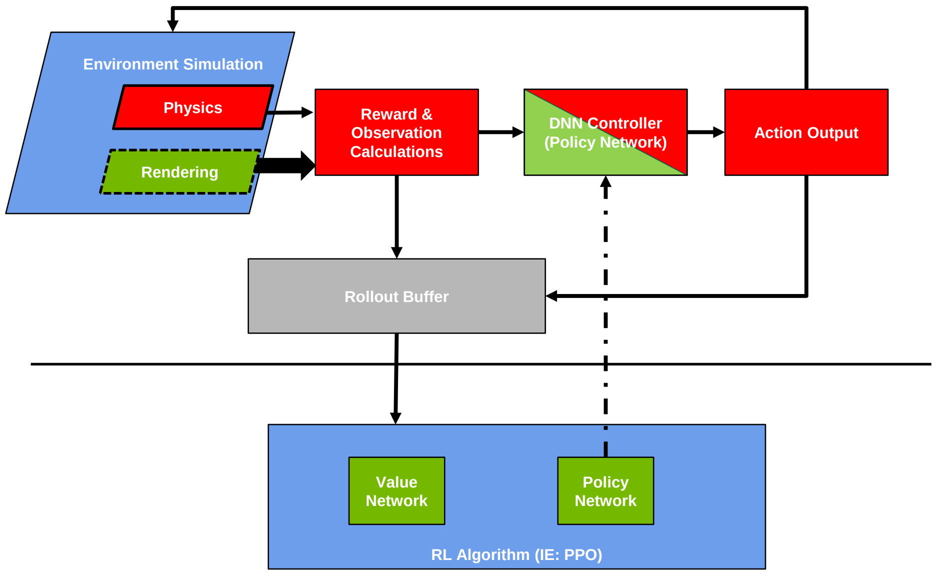
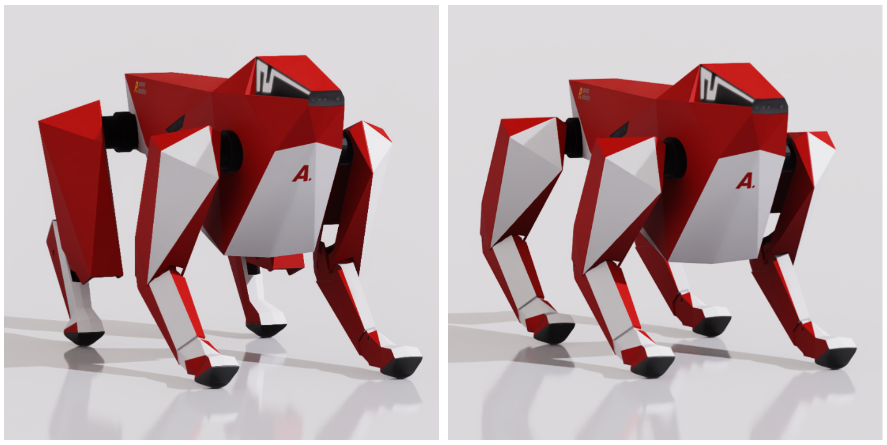


Figure 5.8: Illustration of CPU bottlenecks in previous DRL techniques, red color represents CPU computation, green GPU computation, sourced from [2]

and the OmniIsaac Gym extension. One example is the Cartpole environment, which is a typical control engineering toy problem. The goal may be, for example, to balance the inverted pendulum at a local extreme. This may be done in different ways, computed or trained by different reinforcement learning algorithms. The efficiency of those algorithms is then easily compared with other algorithms without the need for each researcher to do the heavy lifting in the form of physics and environment API implementation on their own.

5.4 Artaban simulation environment

This section describes the implementation of the robotic platform Artaban. It describes the creation of the model and the gait training. Training of the gait can be divided into three distinct parts. The training of the robot with an identical pair of front and rear legs, training of the robot with different front and rear legs, and training of the robot modeling the torque transmission of the Cardan joint. This can be theoretically done for both front-legged Artaban configuration (Fig. 5.9b) as well as for the original configuration, pictured in Fig. 5.9a. However, the original configuration brings some simulation difficulties that are examined in greater detail later.



(a) : The original Artaban version with different rear and front legs

(b) : The front-legged Artaban version with all legs as in the front

Figure 5.9: Two editions of Artaban with different kinematics, as rendered in Isaac Sim

5.4.1 Environment implementation

The environment is implemented on top of OmniIsaac Gym Environments (OIGE) taking inspiration from the environment for `Anymal`. This environment uses the Isaac Sim API (`omni.isaac.core`), taking advantage of the powerful PhysX physics engine, and Isaac Gym API (`omni.isaac.gym`) using the additional environment development based on the Gym format. The Isaac Gym adds to the methods described in 5.3.1, among other things, two methods:

1. `pre_physics_step()` method takes care of what needs to be done before the physical simulator is stepped, e.g. compute position targets based on actions, compute torque or position transformations of a transmission mechanism. In this work, this is where the Cardan joint position transmission is computed.

2. `post_physics_step()` method does what the usual `step()` method does in the original Gym framework, like computing rewards, processing observations and states, resets, extras, etc.

The Isaac Sim was developed with robotics in mind. That is also reflected in its API, which provides a more functional and physically stable simulation of robot-like structures. When two rigid bodies are connected by a joint in the simulation, they still may collide with each other. However, this behavior is often not desirable when simulating robotic structures. Therefore, an object

`Articulation` and `ArticulationView` are provided. An articulation is “an alternative, and often superior approach to creating mechanisms versus adding joints to rigid bodies” [44]. The main practical advantage is its extensive range of methods that allow getting and setting of relevant parameters, variables, and properties such as measured joint forces, measured joint efforts, joint angular velocities, link velocities, transformation between links, calculating inertial tensors, and much more. An implementation advantage is that the joints in an articulation are simulated in a reduced-coordinate system. This means that instead of simulating each rigid body in world coordinates, the positions and orientations are computed relatively to the root joint of the articulation and “that reduced coordinate articulations are guaranteed to exhibit no joint error” [45]. In articulation, two links that are connected together by a joint do not collide by default, but unconnected rigid bodies do collide. This is regarded as self-collision. For our robotic platform Artaban, self-collision is taken care of by software joint limits. Those limits are set in a way that prevents collision with other robot parts if respected. Since checking of self-collision uses up some computation time, it is recommended to leave it off by default unless the robotic learning task requires so. Collisions between two articulations are, naturally, respected.

Closing an articulation loop

The articulation is defined as a tree structure in terms of the rigid body links and joints between them. In order to create an articulation loop, some joint in the loop needs to be excluded from the articulation. When a joint is excluded from articulation, the physics simulator lowers this joint’s priority in the resulting position computation. Therefore, other joints are prioritized, and this joint is computed only according to usual joint restrictions. The joint that needs to be excluded from articulation is shown in red color in Fig. 5.10. The joints in blue are actuated; the rest of them are green. Those green joints are included in the articulation, although they are not powered but only moving according to kinematic restrictions. The actuation of the lowest part of the leg is done purely via physics interactions in the simulation. That means there is no driven (actuated) joint moving the lowest link directly, only indirectly via the middle link (blue line connecting blue and green joint) and pantograph (upper green line), using nomenclature described in Fig. 5.3b.

The fact that the joints excluded from articulation are load-bearing and great torques are generated in order to keep the robot standing proves to be a major problem for the simulator. There are various choices for the joint type in the articulation:

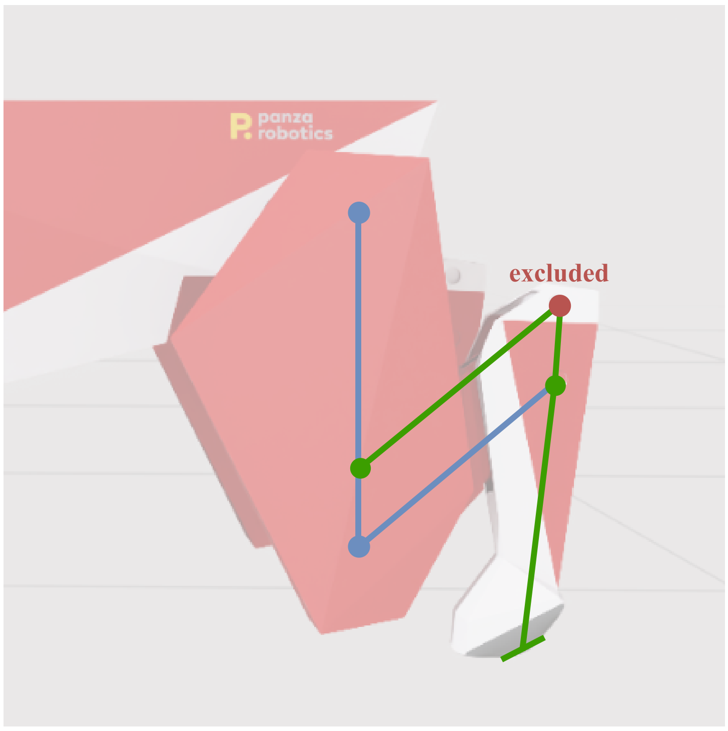


Figure 5.10: Illustration of the articulation loop implementation, overlapped on top of a simulation render of the rear left leg

1. **Fixed joint** - fixes two links together under an arbitrary angle
2. **Revolute joint** - allows rotation only around one axis
3. **Prismatic joint** - allows linear motion in the direction of the joint axis
4. **Spherical joint** - allows rotation in a cone centered in the joint axis
5. **D6 joint** - allows configurable motion - specification of individual degrees of freedom either to move freely or to be locked together
6. **Distance joint** - introduces distance constraint between two origins, allowing also elastic limits via stiffness setting, simulating, e.g., a rubber rope. If no stiffness is introduced, the distance is kept strictly as defined.

5.4.2 Reinforcement learning pipeline

The RL pipeline consists of the environment, the agent, and an optimization algorithm. The library of choice is the `rl_games` that implements the PPO and the Actor-Critic model. In Fig. 5.11 on the left, the training environment is depicted. OIGE API stands for Omni Isaac Gym Environments API. The

Gym Wrapper represents that the whole environment is built on Gym API. The agents are spawned in parallel, thanks to the creation of instancable assets. Insatncable assets are a specification of OpenUSD where the meshes and collider maps are imported into the simulation only once, and the rest of the agents use the central instancable asset as a reference, becoming only instances of the agent.

The agents are then run in the environment given a random velocity command in the XY horizontal plane and yaw. The policy decides what actions to take, given observations. The actions represent a difference from the current goal and are clipped by a maximum magnitude. I.e., given that the drives are set to a position control mode, the actions add or subtract from the latest position goal of each drive. The position goal is then fulfilled by a PD controller via the PhysX API, as the simulation is stepped.

Following, the observations are collected, and the cycle continues. The observations are collected until the end of an episode into mini-batches and are used for computing rewards, values, and advantages into the loss function that, through stochastic gradient descent (SGD), optimizes the policy. A single observation vector consists of:

- base linear velocity: $v_{\text{base}} \in \mathbb{R}^3$
- base angular velocity: $\omega_{\text{base}} \in \mathbb{R}^3$
- projected gravity: $g_{base} \in \mathbb{R}^3$
- commands: $v_x^{\text{cmd}}, v_y^{\text{cmd}}, \omega_z^{\text{cmd}}, \in \mathbb{R}$
- DOF positions: $\theta_{\mathcal{J}} \in \mathbb{R}^{|\mathcal{J}|}$
- DOF velocities: $\omega_{\mathcal{J}} \in \mathbb{R}^{|\mathcal{J}|}$
- actions: $a^{\pi} \in \mathbb{R}^{|\mathcal{A}^{\pi}|}$

Where $|\mathcal{J}|$ represents the number of joints and $|\mathcal{A}^{\pi}|$ represents the number of actions. The notation is different for those two parameters because the robot may have more joints than it can articulate. It is beneficial for the RL pipeline to gather observations about all of the joints, not only the articulated ones.

The value function is optimized on its own axis according to the discrepancy in its value function for the predicted and recorded state. The details of PPO-Clip were explained in Section 4.3.2.

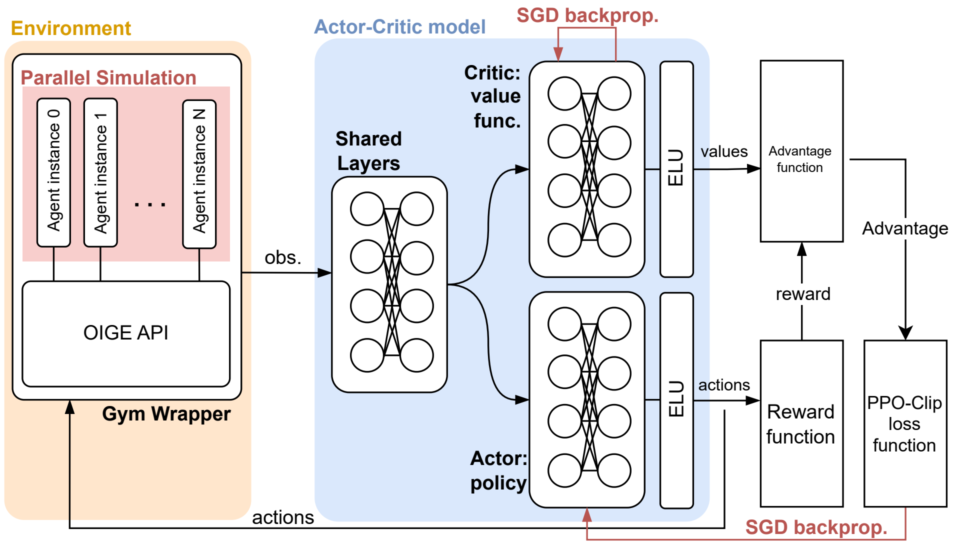


Figure 5.11: Tentative diagram of the RL pipeline, showing the environment in peach orange, agent model in blue, and the PPO blocks on the right

5.4.3 Reinforcement learning pipeline parameters

Machine learning and, most importantly, deep learning, whether reinforcement learning or supervised learning, are a matter of parameter optimization. Network parameters are optimized in a learning procedure. However, the hyperparameters of the reward function and optimization algorithm are optimized in a brute-force way. The number of parameters is, therefore, counted in double digits at best and is subject to the combinatorial explosion. Any reward or optimization algorithm space search is very time-consuming and resource-demanding, therefore, the training pipeline is kept the closest to default hyperparameters as possible. Some tweaks to the parameters are nevertheless needed. The reward function implementation and parameters take inspiration from works described in the Chapter 2, mainly [13] and [7].

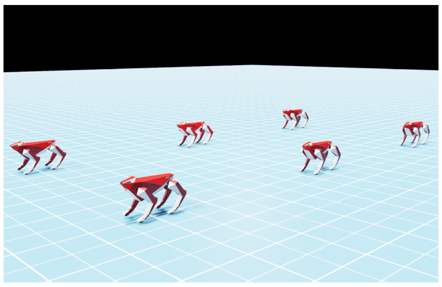
Although for a stronger intuition on what effect each reward implementation and scale have on the training result, two types of experiments were undertaken: **reward decomposition** and **reward ablation**. Reward decomposition aims to see which rewards are sufficient for gait training (one-hot analysis), and reward ablation aims to see what qualitative effect each of the rewards provides (one-cold analysis). Reward decomposition sets all rewards except the chosen one to zero. This is done for each non-zero reward in a configuration file, generating a set of trainings, which are benchmarked on a Cost of Transport qualitative basis. Reward ablation follows a similar process. The whole reward configuration is left as-is except one reward, which is set to zero. This is done for all non-zero rewards, similarly as before, generating a set of trainings that are benchmarked on a Cost of Transport qualitative basis. The word ablation originates from the Latin *ablā-* or *ablatus*, meaning "removal" or "carrying away." Ablation is a neurosurgical method for treating neurological diseases and an experimental neurological method for finding causation between actions and brain regions [46].

Chapter 6

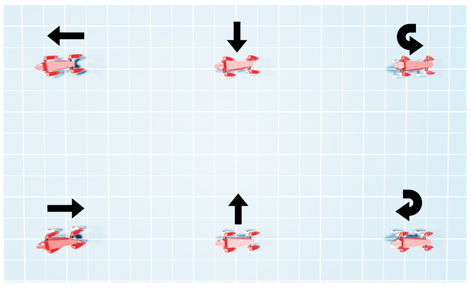
Experiments & Results

The reward function provides the agent with instrumental smaller goals to pursue its ultimate goal. In this thesis, the main agent's goal is to follow velocity commands showing some form of gait. To compare different gaits against each other, I reserve the Cost of Transport (CoT) metric for the sole purpose of evaluation. CoT was not used as a reward signal during the training.

6.1 Evaluation environment



(a) : Configuration of robots in the evaluation environment



(b) : Top-view of the evaluation environment with command directions

Figure 6.1: Evaluation environment configuration

In Fig. 6.1 is shown the configuration of the evaluation environment. This environment serves as a benchmark for comparing different gaits. The commands are the same for all tested policies. The six robots in Fig. 6.1

represent all the same policy but are getting different velocity commands in the order shown in Fig. 6.1b.

6.2 Reward function benchmark

The following reward and other benchmarking were run on the front-legged kinematic configuration in order to eliminate pathologies brought by imprecise simulation of the loop closure on the original kinematic configuration. The agent was spawned in 6 instances, with commands as in 6.1b. The Cost of Transport (CoT) was computed for linear velocity commands only (i.e., angular velocity commands were not taken into account) in order to avoid mixing angular and linear velocity (i.e., $[m/s]$ and $[rad/s]$), and by doing so making the physical meaning of CoT inconsistent.

The Cost of Transport metric is used in the form

$$\mathrm{CoT} = \frac{\sum_{j \in \mathcal{J}} \tau_j}{|v_{\mathrm{XY}}| \cdot m_{\mathrm{r}} \cdot g}, \tag{6.1}$$

where $\mathcal{J}$ is the set of joints, τ_j is effort expended for joint j per second $[Nms^{-1}]$, i.e., magnitude of torque per second, $|v_{XY}|$ is the magnitude of the velocity in the horizontal XY plane $[ms^{-1}]$, m_r is the mass of the robot in $[kg]$ and g is the gravitational constant $[ms^{-2}]$. The purpose of the CoT metric is to generate a ratio of effort to velocity that approximates optimal movement strategies; the mass and gravitational constant serve a normalization role, and the metric is unitless $[-]$.

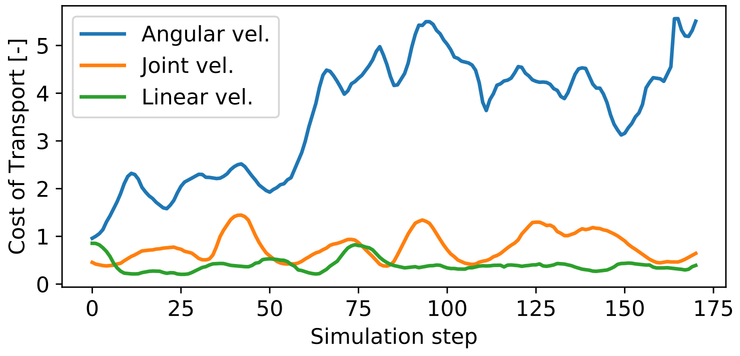
Reward decomposition and reward ablation studies were conducted to enhance understanding and establish benchmarks for different rewards, as described in 5.4.3. The scales given for different rewards are denoted in Tab. 6.1. Note that some of the rewards have different scales (highlighted in bold for clarity) for the front-legged and original kinematic configurations of the robot. This is due to different demands of the kinematics. For example, the robot with original kinematics (with rear legs different from the front) had a common failure mode of dropping down to the limits of the knees. The gait was stable but not desirable since such gait would quickly destroy the chassis of the legs. Therefore, the penalty for deviation from set knee height and base height was amplified. The evaluation for the reward benchmarking experiments was run for 200 steps. The first 30 simulation steps are ignored, because that corresponds to the spawning time of the agents. The spawning introduces sudden forces on touch-down with the floor and brings unnecessary noise to the results.

Reward name	Front-legged	Original
Action rate	-0.01	-0.01
Angular Z vel.	0.5	0.5
Base height	-0.1	-2.0
Cosmetic	-0.06	-0.06
Collision	-0.5	-0.5
Fallen over	-1.0	-1.0
Feet force	-0.01	-0.01
Joint acc.	-0.0003	-0.0003
Joint vel.	0.15	0.25
Knee height	0.0	-1.0
Linear XY vel.	1.0	1.0
Linear Z vel.	-2.0	-2.0

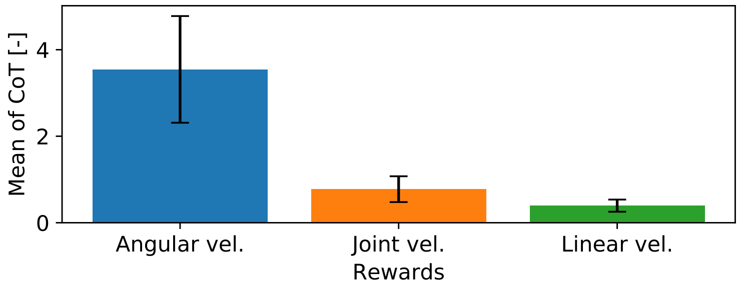
Table 6.1: Rewards scale overview for front-legged and original kinematic configuration

6.2.1 Reward decomposition

Decomposing the reward landscape into separate parts helps to identify which rewards are sufficient to incentivize some form of gait. The reward decomposition reveals only three rewards that were able to create a valid gait. A valid gait is understood as a series of motions that move the robot without forbidden collision. A forbidden collision is a collision between the ground and any upper part or the base of the robot. As shown in Fig. 6.2, only the rewards related to either the velocity of the robot or the velocity of the joints led to a valid gait. Unsurprisingly, a reward that reinforced linear motion in the XY plane led to the most efficient gait from the reward decomposition experiment. This gives an insight into the importance of the velocity rewards. However, reward decomposition does not bring useful information about other rewards that are meant to tweak the quality of the gait without the gait existing. The time-series and bar graphs in the 6.2 represent the performance of the gait with the inspected reward used as the only reward signal. For example, the time-series of the *Angular vel.* in 6.2a displays the Cost of Transport metric value over time when the scale rewarding the *Angular velocity* was the only one used (reward function decomposed into parts, in this instance inspecting the *Angular vel.* reward).



(a) : CoT time-series for runs that incentivized valid gaits with the corresponding rewards as the name



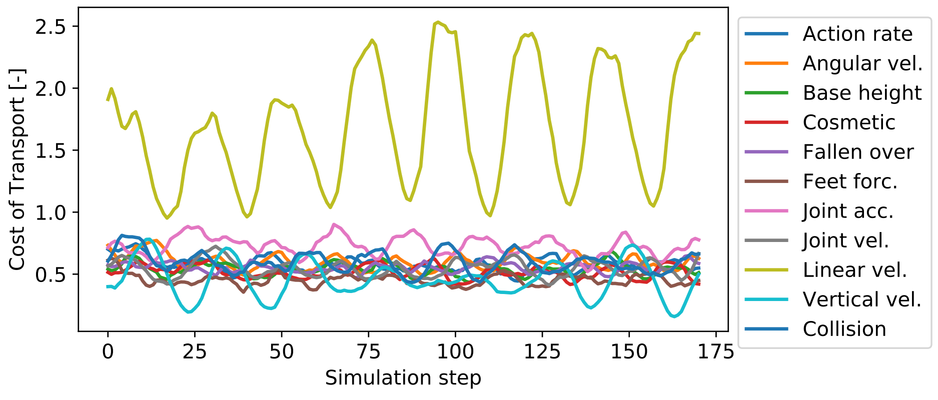
(b) : Mean ± standard deviation of CoT for valid gaits with the corresponding rewards as the name

Figure 6.2: Reward decomposition results overview

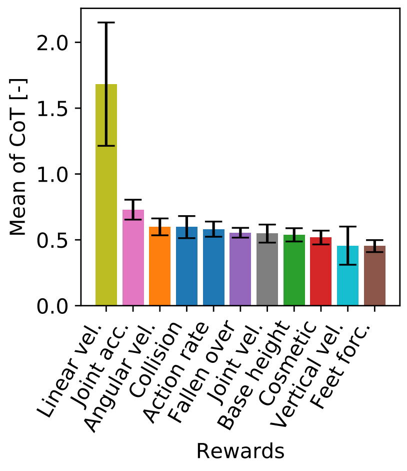
6.2.2 Reward ablation

The reward ablation approach to understanding the reward landscape provides an orthogonal perspective compared to reward decomposition. In other words, when decomposing the reward function to single reward signals, often the robot does not learn a valid gate. Therefore, those runs cannot be judged on the basis of the Cost of Transport. Given that at least some other reward alone leads to a valid gate, the other rewards, when ablated, show an average increase or decrease of the CoT metric, hinting at their importance.

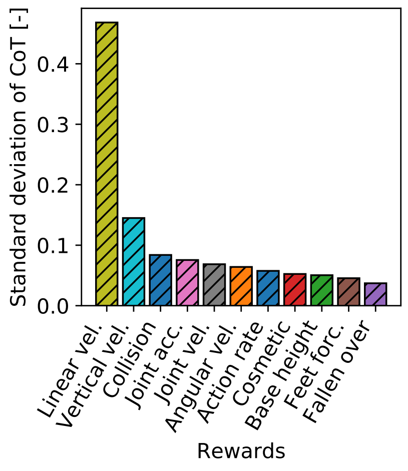
In Fig. 6.3a, the training run with the Linear XY velocity reward signal ablated shows greater variation and mean value compared to the rest of the reward signals. This confirms the observations from the reward decomposition. Figures 6.3b and 6.3c provide a clearer visual understanding of the time-series in Fig. 6.3a. Vertical movement of the robot's base, erratic jumping, and collisions of the lower links with the ground result in an uneven gait. This translates into larger variance, and therefore, tweaking the top-ranking rewards in the 6.3c helps with the motion smoothness.



(a) : CoT time-series for runs that incentivized valid gaits



(b) : Training runs sorted by the magnitude of the CoT mean, carrying the name of the corresponding ablated reward



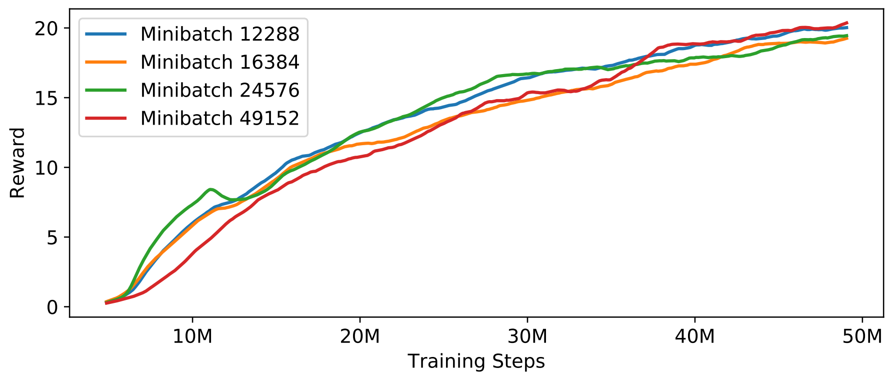
(c) : Training runs sorted by the magnitude of the CoT standard deviation, carrying the name of the corresponding ablated reward

Figure 6.3: Reward ablation results overview, the names in the legend and bar graphs represent the name of the inspected reward

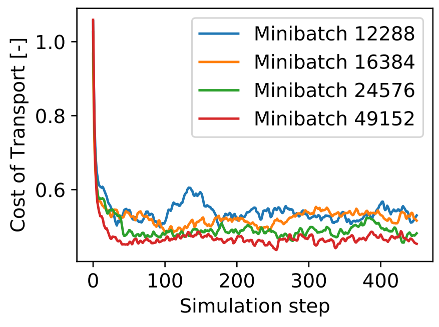
It is important to note that the purpose of the reward benchmarking experiments is to gain a data-driven intuition on the importance of individual rewards. Reward decomposition and ablation do not show the whole picture of the interdependencies of individual rewards. It is often desired to design a reward function in such a way that different reward signals balance each other. For example, giving a penalty on the knee height difference and simultaneously rewarding joint velocity creates an interdependency.

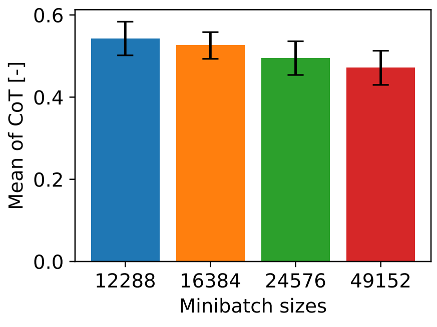
6.3 Influence of batch size on gait quality

The work of Rudin et al. [13, Section 2.2.1], among other things, inspects the meaning of batch size in the massive parallelism setting of GPU training. The gist of this section is that the batch size, consisting of $n_{agents} \times n_{steps}$ cannot be too small or too large. Too small means that the agents would not have enough simulation steps to understand the temporal effects of their actions. Too large means that the agents would have too many simulation steps to observe, more than what they could gather new insight from, hence wasting simulation time. A minimal horizon length is described in the publication. This length is a number set arbitrarily and describes the minimal number of simulation steps that contain enough temporal information. However, the simulation can run for longer. The maximum length proposed in the publication is 20 seconds; I kept this hyperparameter set to this value. Runs with mini-batch sizes computed as $n_{agents} \times n_{horizon} \times \frac{1}{m}$ where m is the number of mini-batches ranging 1 to 4 were conducted and are shown in Fig. 6.4.



(a) : Cumulative rewards obtained by agents with different mini-batch sizes





(b) : Mini-batch runs performance in the CoT metric

(c) : The mean value of mini-batch runs performance in the CoT metric

Figure 6.4: Evaluation environment configuration

An interesting observation is that even though not much difference can be seen between the cumulative rewards in Fig. 6.4a, the larger mini-batch size helps minimize the CoT metric. The number of epochs was smaller than in [13], 1000 in this work compared to 1500 in the paper. However, during the training runs, the overall trend was that if reward scales were balanced and the simulation environment was implemented correctly, the gait took form in the first 1000 episodes, and the rest of the additional episodes was only fine-tuning or even overfitting the policy that led to pathological behaviors. The trend of better performance with larger mini-batch sizes still stands. According to those findings, the agents were trained in a configuration that was possible to run on my computer setup - 2048 agent instances at 49152 `minibatch_size`.

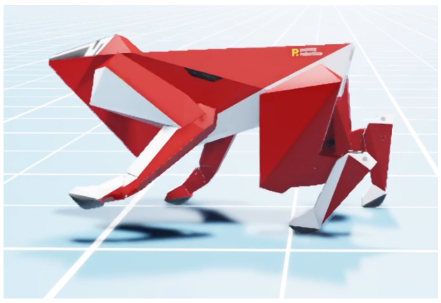
6.4 Training progression

This section traverses the iterative training process, points out simulation infidelities, describes differences in training of different kinematic configurations, and shows what worked and what did not work. The first training checkpoint was to learn some form of valid gait with the original Artaban configuration, as shown in Fig. 5.9a with the excluded articulation joint as pictured in Fig 5.10.

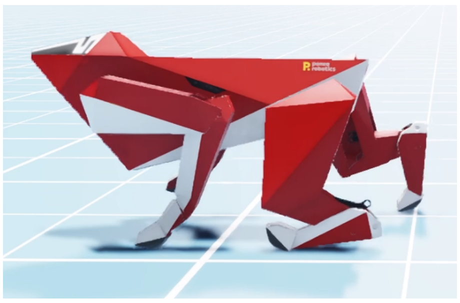
6.4.1 Original kinematic configuration - rear-legged

Isaac Sim approaches the articulation loop in a manner where the positions of the joints included in the articulation are computed with higher priority than the ones that are excluded. However, the joint excluded from articulation still has a major role in the location of the leg and carries a major load. This proves to be a problem for the simulator, as it either lacks in holding the joint firmly together (i.e., introduces position error, which is allowed in the simulation of bare joints, contrasted with the articulation joints - see Section 5.4.1), or generates parasitic forces when the solver iteration count is set to larger values. The parasitic forces result in an instability of the simulation, and the whole robot erratically flies away in a manner that does not match the laws of physics. The lack of joint integrity is obvious to the naked eye because the pantograph is visible during walking, which is not a valid behavior, see Fig. 6.5b.

This effect is observable thanks to the model of direct kinematics used for



(a) : Artaban before stepping on the rear left leg



(b) : Artaban after stepping on the rear left leg with joint non-integrity

Figure 6.5: Example of position error of a joint excluded from articulation

the real-life robot, which was made available to me by Panza Robotics. With the direct kinematic model of the rear four-link mechanism, I can compare what should be the joint location of the pantograph and lower link joint given the angle of the middle knee joint. The comparison of the time-series data corresponding to the photographs from the simulation in Fig. 6.5 is shown in Fig. 6.6. The graphs in orange represent joint positions in Isaac Sim, and in blue, the positions that should take place if the kinematics were to be fully respected. The lower left graph is particularly concerning as the orange graph fails to follow the blue line almost completely, signalizing the limping behavior shown in the simulation photographs.

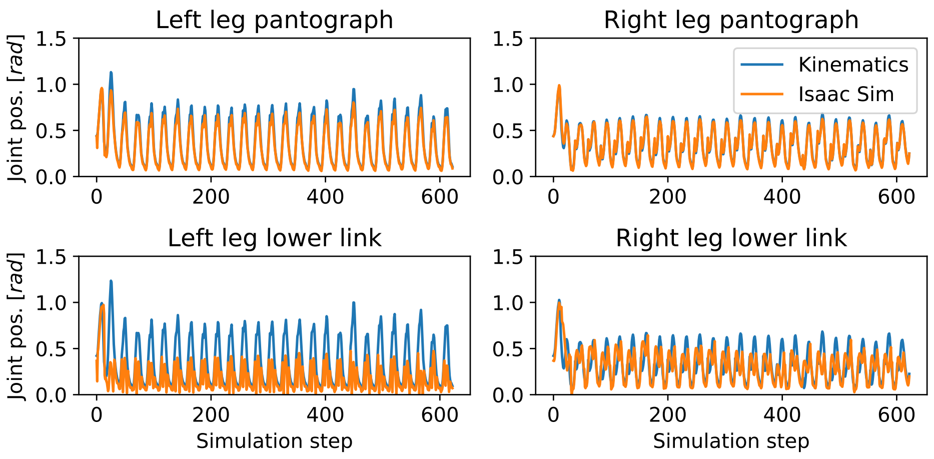


Figure 6.6: Time-series data comparison for both pantographs and lower links on both rear legs

Tweaking various simulation parameters followed to address this problem, although with no satisfactory result. The first approach was to run the simulation with more simulation steps per second. However, this parameter did not and apparently should not help in this scenario. That is because the length of the simulation step only dictates how frequently the policy is able to act per perceived time. According to the Nvidia forums and documentation, the simulation position iteration is the key factor in the computing of forces and positions of rigid bodies in the simulation. The position iteration parameter dictates what is the maximum of iterations per simulation step of the physics engine to find a suitable solution to kinematic constraints. Increasing the position iteration parameter from 4 to 8 or even to 16 seemingly helped, although on closer inspection of the collected rewards and resulting gait, it was obvious that unstable explosive behavior occurred during some epochs of training. That is because the gait turned out to be very conservative (barely moving the rear legs from a straightened pose) and barely putting weight on the rear legs. The policy seems to rather put weight on the front legs instead of risking the explosion and, therefore, a large penalty. The large penalty occurs because of restrictions on the height of the base link of the body. The error in the body height is squared and used as a scaled negative penalty. When the instability occurs, the robot flies away at great speeds into great heights and collects great penalties. Therefore, the policy learns to avoid those motions that result in great penalties, ergo avoiding using the rear legs with full weight. See the videos in appendix F.

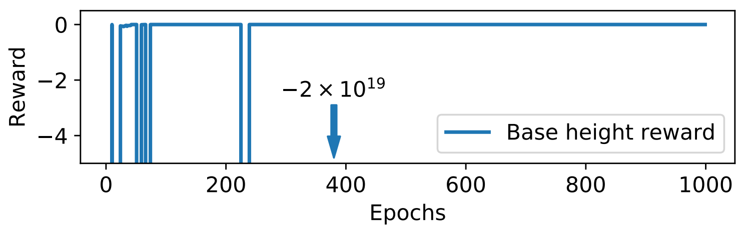


Figure 6.7: The time-series of collected rewards showing rapid oscillation to large negative numbers in the early training, the graph is cropped for better clarity

A similar fate met configurations with a spherical or distance joint instead of the revolute joint when excluding from the articulation. The joint nomenclature was described in detail in Section 5.4.1. For the distance joint, the pantograph had to be removed, and the distance joint supplemented its role. The implementation visualization is pictured in Fig. 6.8.

6.4.2 Evaluation of different training runs - rear-legged

None of the actors with rear-legged configurations were able to discover a gait that would be deemed satisfactory enough. Even though the trained gaits were technically valid, some of them showed non-integrity in the joint excluded from the articulation. When focusing the reward function implementation, its parameters, and the parameters of the simulation environment on addressing this problem, the agent still discovered gaits that show traits of so-called

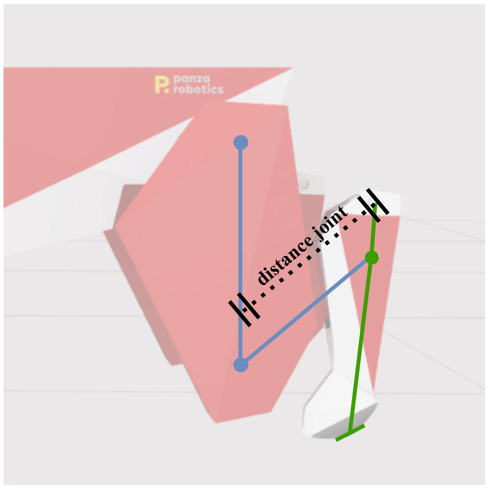


Figure 6.8: Illustration of the distance joint implementation, overlapped on top of a simulation render of the rear left leg

Goal misgeneralization. Goal misgeneralization is a reinforcement learning phenomenon that describes the difference between what the author meant to achieve with the reward function definition compared to what the agent achieved while still formally adhering to the definition of the reward function [47]. This behavior is witnessed so often in reinforcement learning settings that it should actually be considered to be the default.

Baseline gait

The first successfully obtained gait is named the **Baseline gait**. This gait produces small velocities and moves in a motion resembling gallop, i.e., the motion that horses or dogs use when running fast: jump synchronously from rear legs to front legs and then again. The gait can be minimally observed in the photograph strip in Fig. 6.9, in a longer format in the Appendix C.1 or in the video part of the Appendix F.

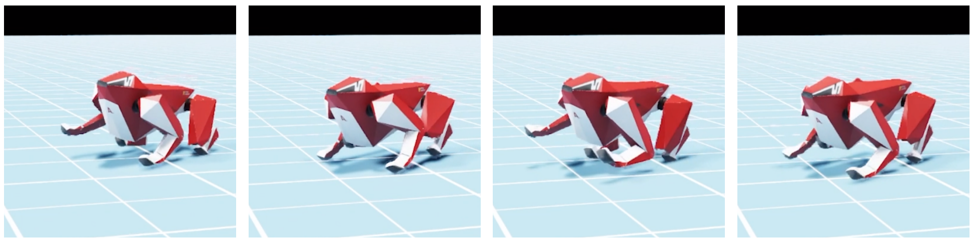


Figure 6.9: The baseline gait for the original kinematic configuration of Artaban

Knees reward gait

Since the previous **Baseline gait** was landing on the whole body of the rear lower link instead of only on its foot, a greater penalty was introduced that penalized the squared distance of the knee from the reference height. This penalty worked as intended and produced the **Knees reward gait**. However, the problems with joint integrity occurred, as described in Section 6.4.1 and shown in Fig. 6.5 and in the time-series graph Fig. 6.6. The resulting gait with emphasis on knee height penalty can be seen in the photograph strip in Fig. 6.10, in a longer format in the Appendix C.2 or in the video part of the Appendix F.

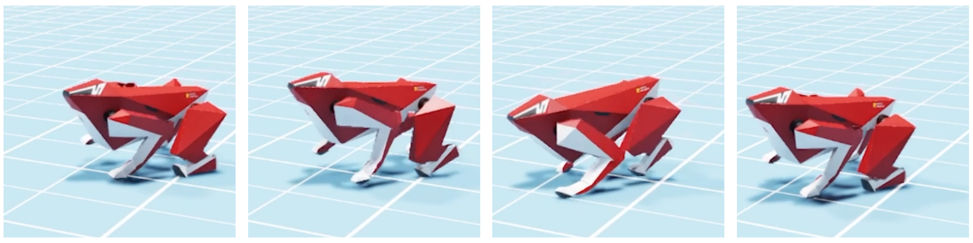


Figure 6.10: Training run with stronger penalties on knee height

Joint limits gait

The limping shown in the previous subsection in **Knees reward gait** occurs when the rear leg falls all the way toward the limit of the joint. In order to penalize this behavior, a soft limit penalty was introduced. This penalty penalized the robot when turning the joints to the edge 20% of the range on each side, creating the **Joint limits gait**. This, however, produced yet another instance of Reward misspecification / Goal misgeneralization where the agent found a gait that led to minimizing the soft limit penalty, leaning mostly on the front legs and stretching the rear legs while still being below the penalty region of the joint space. The motion captured on the photograph strip in Fig. 6.11 is a backward motion because the forward motion was insignificant, perhaps impossible with this reward setting. A longer format in the Appendix C.3 or in the video part of the Appendix F.

Distance and Spherical joint gaits

Those two gaits are mentioned together because they look visually very similar, mostly in the forward motion. The agent, again, learned that the

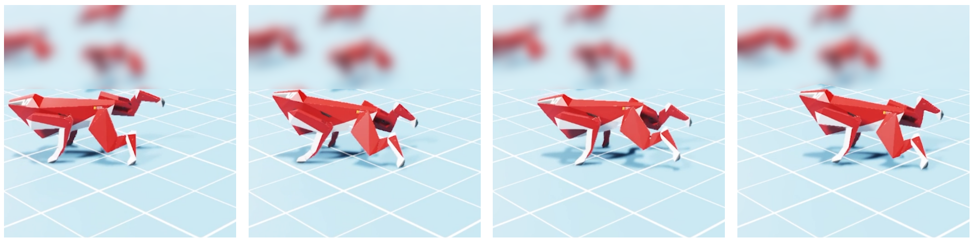


Figure 6.11: The training run with penalties for being close to joint limits

gait is more stable when jumping on the whole lower links of the lower legs, instead of only standing on the end of the feet. This is mostly because the large forces start to create parasitic forces as described in Section 6.4.1 and shown the typical evidence of this behavior in the body height rewards in Fig. 6.7. The **Distance joint gait** and **Spherical joint gait** are shown together in 6.12. A longer format can be seen in the Appendix C.4 or in the video part of the Appendix F.

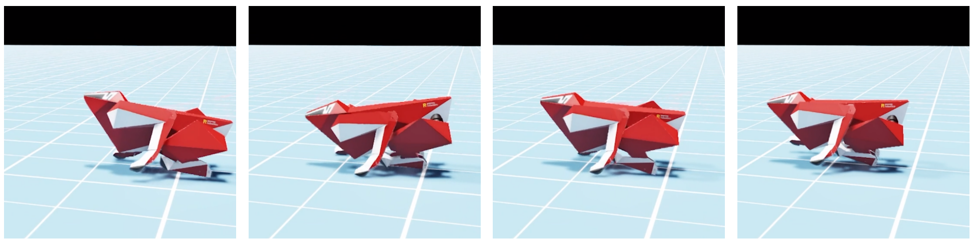
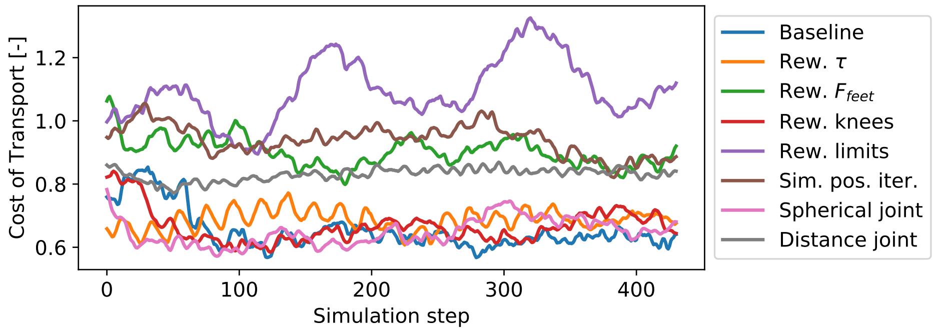


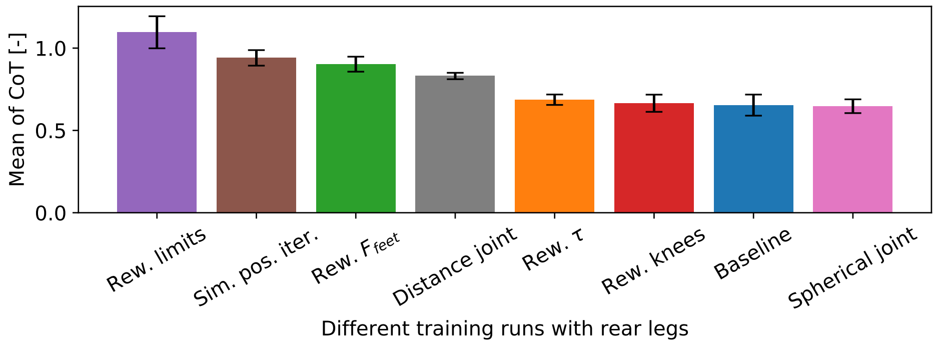
Figure 6.12: The gait for the original kinematic configuration of Artaban with distance joints

Results for the chosen training runs in the CoT metric

In order to compare the gaits between themselves and also with gaits of different kinematic configurations, a Cost of Transport metric was chosen since it is agnostic to the number of joints previously defined in eq. 6.1. The baseline gait without the knee rewards or soft limits did fairly well compared to the gaits with the reward implemented. It ranked among the top three in the CoT metric ranking. The first in the ranking is the **Spherical joint gait**, however, it is almost identical to the **Baseline gait** and **Knee rewards gait**. The time-series of the CoT throughout simulation steps and runs with the mean values of CoT ranked from the largest to the smallest can be seen in Fig. 6.13. The graphs also contain some training runs that were not mentioned in this section for brevity; however, they can be accessed in the video appendix.



(a) : CoT time-series for chosen rear-legged gaits



(b) : Mean ± standard deviation of CoT for chosen rear-legged gaits

Figure 6.13: Comparison of chosen rear-legged runs

Note on bugs in Isaac Sim

Yet another problem that made the implementation of the closed-loop articulation difficult is a confirmed bug in the implementation of Isaac Sim importer that incorrectly assigned the articulation loop to a random rigid body instead of to the base link. This hidden problem resulted in strange behavior, causing unwanted rotations on spawning. The problem was that the articulation root was set to the left pantograph primitive instead of the base link, which led to the rotation of the whole body when setting the nominal position of all degrees of freedom at the beginning of the simulation. After a long back-and-forth with the developers, they recorded and fixed this bug for the next release, which was sadly not yet available during the writing of this thesis. This problem could be partially resolved by creating a custom ‘ArticulationView‘ for the base link and reading the rotations and positions directly, not through the articulation root. However, the spawning routine is not accessible and needs to be fixed from within. Therefore, the robot needed to spawn at a slight angle because that is the default pose of the pantograph. The GitHub issue is in the reference [48].

6.4.3 Front-legged kinematic configuration

The rear-legged configuration proved unreliable in the simulation environment, and therefore the simpler, front-legged configuration is trained. The goal of this thesis is to implement the novel patented kinematics solution, which means modeling the Cardan mechanism, spinning the motor in one direction, and then benchmarking whether the overall effort-to-velocity ratio (CoT) is better with the novel kinematics. Because the original kinematic configuration often provides gaits that take advantage of the lack of integrity of the joint excluded from articulation, the benchmarking with and without Cardan mechanism modeling could become unclear. In order to eliminate those unknowns, first, a gait on the front-legged configuration without the Cardan mechanism is trained. The front-legged configuration can be clearly seen in Fig. 6.14.

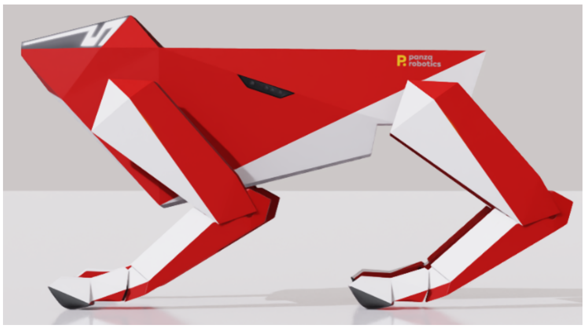


Figure 6.14: Side view of the front-legged configuration

6.4.4 Front-legged kinematic configuration - Cardan mechanism

The mathematical modeling of the Cardan mechanism was already mentioned in Section 5.1.3. The policy has access to the action for the motor output shaft of the Cardan mechanism directly. That is what is possible to observe and control by the real robot. The advantage of this mechanism is that the motor can spin in one direction while the actuator (the Cardan mechanism) follows a sinusoidal trajectory. This embodiment-specific motion generator can be theoretically used to help the robot discover efficient gaits more quickly. The way the motor shaft of the Cardan mechanism spins can be influenced either indirectly via reward shaping or directly by forcing the motors to a constant angular velocity. When setting an offset of π $[rad]$ to the motors of the Cardan mechanism on front-right and rear-left legs, a simple *"forced trot"* can be achieved. An illustration of how such a trot looks like can be seen in Fig. 6.15

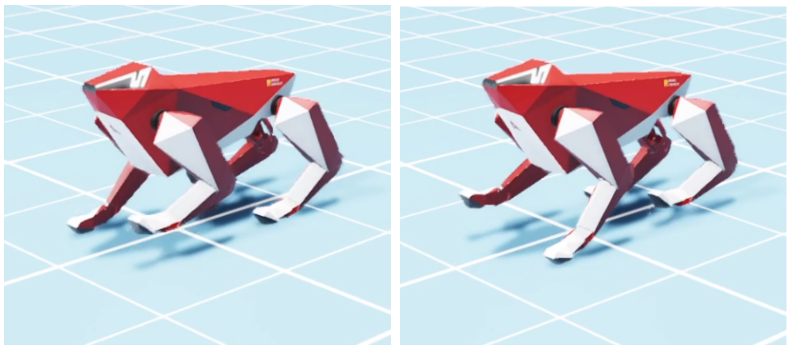


Figure 6.15: Example of forced trot with the Cardan mechanism

6.4.5 Evaluation of different training runs - front-legged

Since this kinematic configuration is less complex than the original one, the training was expected to be more straightforward.

Baseline gait

The rewards described in the column named *Front-legged* in Tab. 6.1 produced the **Baseline gait** for the front-legged configuration. Larger dynamic friction had to be used for the legs because the robot lacked grip compared to the original configuration. The training produced a decent gait that is valid and looks as expected. The gait can be seen in the photography strip in Fig. 6.16 and in the appendix in the form of a video.

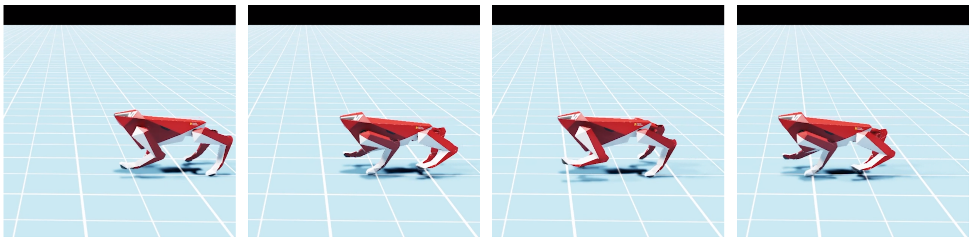


Figure 6.16: The baseline gait for the front-legged kinematic configuration of Artaban

Cosmetic gait

In the **Baseline gait**, the robot helps its movement by moving the rocker joints up and down. The rocker joints are the first joints connecting the legs to the body. Those joints are responsible for moving the legs to and from the body. Increasing the penalty on those joints resulted in a stiffer motion, and the robot dragged the rear legs mostly behind. This signals that perhaps the knee/elbow limits are not large enough for the robot to close and open the leg comfortably when the rockers are penalized for movement. Nevertheless, this approach was not successful, and the gait was less visually appealing while also being less cost-effective. The gait may be observed in Fig. 6.17, in a longer format in the Appendix D.2 or in the video part of the Appendix F.

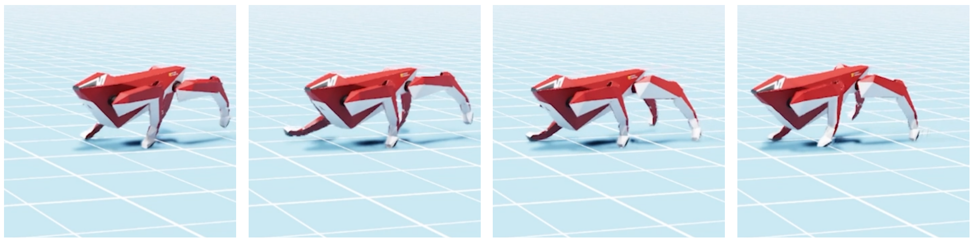


Figure 6.17: The gait for the front-legged kinematic configuration of Artaban with rewards incentivizing placement of rocker joints close to nominal position

Cardan: Baseline gait

This training run produced a **Cardan: Baseline gait**. The name suggests that the rewards did not reflect the Cardan mechanism and were left as was the case for the front-legged **Baseline gait**. I.e., there were no rewards that would penalize or incentivize any motion of the motor or cardan mechanism. The actions were directly remapped from the knee joints to the positions of the motor shafts of the Cardan mechanisms, and a gait emerged. This learned gait did not provide a good-looking or effective motion. The agent had, similar to the previous gait, trouble with going forward but managed the sideways and backward motions. The backward motion is visible in the photography strip in Fig. 6.18, in a longer format in the Appendix E.1 or in the video part of the Appendix F.

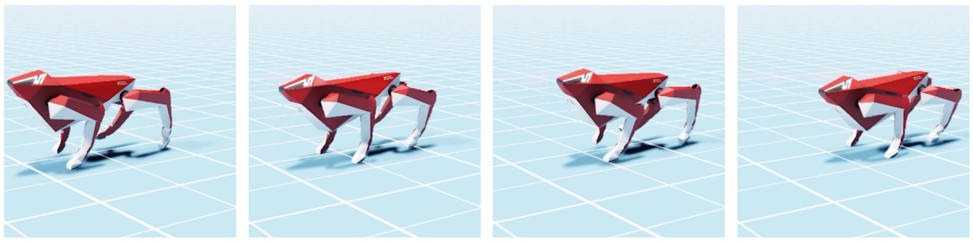


Figure 6.18: The gait for the front-legged kinematic configuration of Artaban with Cardan mechanism in place, which is backwards because the forward gait was not clearly visible in pictures

Cardan: Velocity reward gait

As mentioned in Section 6.4.4, the goal is to spin the motor in one direction and let the agent figure out the rest. Therefore, rewarding the velocity of the motor shaft of the Cardan mechanism is an obvious choice for a reward. The resulting **Cardan: Velocity reward gait** is shown in Fig. 6.19, or in a longer format in the Appendix E.2 or in the video part of the Appendix F. The forward motion was, again, problematic, and therefore, the backward motion is shown. The video in the appendix provides more insight. Mainly the fact that the period of the oscillation is too low to produce meaningful gait. A most probable explanation for this fact is that the action that would constantly increase the speed of the shaft motor had to be consistent for too long. My explanation is that the sole purpose of Proximal Policy Optimization restricted the agent from trying out actions that are too large and too different, therefore staying in a worse local maximum. Reinforcement learning is employed in settings where the solution is often complex. The goal of this approach is clear - make the motors rotate. Therefore, directing the motors to spin might reach this goal in an easier way.

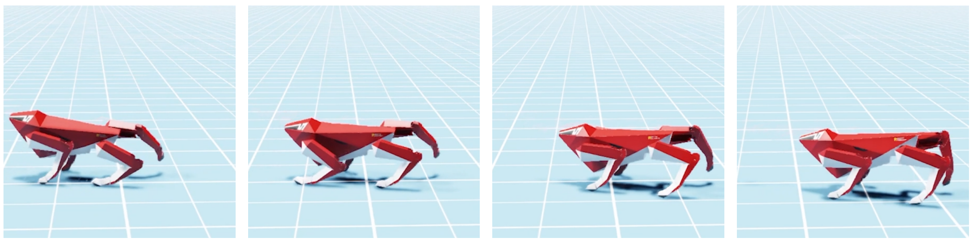


Figure 6.19: The gait for the front-legged kinematic configuration of Artaban with Cardan mechanism in place with rewards incentivizing higher velocity of the motor shaft on the mechanism

Cardan: Forced trot gait

Reaching a trot gait purely by reward shaping has shown to be more difficult than expected. Therefore, a training experiment was conducted with the motors spinning at a constant rate and their starting positions alternatively offset as described previously in Section 6.4.4, by π $[rad]$. This **Cardan: Forced trot gait** has shown great results. The gait is both visually appealing and cost-effective. The gait is showcased in Fig. 6.20, or can be seen in a longer format in the Appendix E.3 or in the video part of the Appendix F.

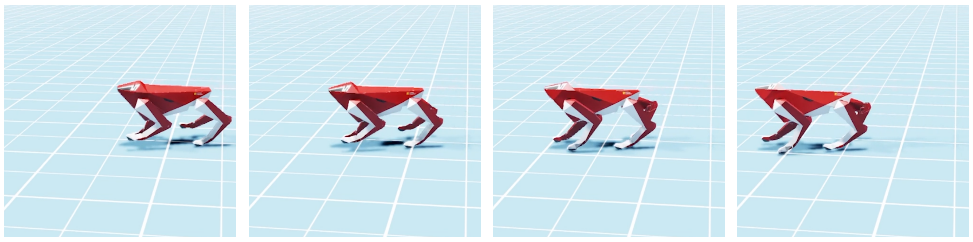


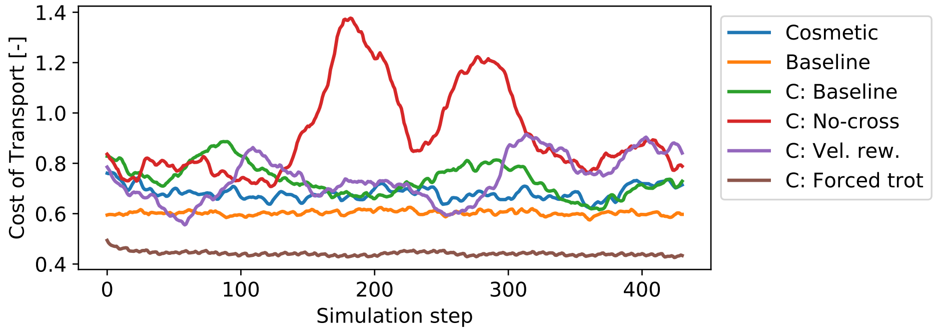
Figure 6.20: The gait for the front-legged kinematic configuration of Artaban with forced Cardan mechanism trot

Results for the chosen training runs in the CoT metric - Cardan

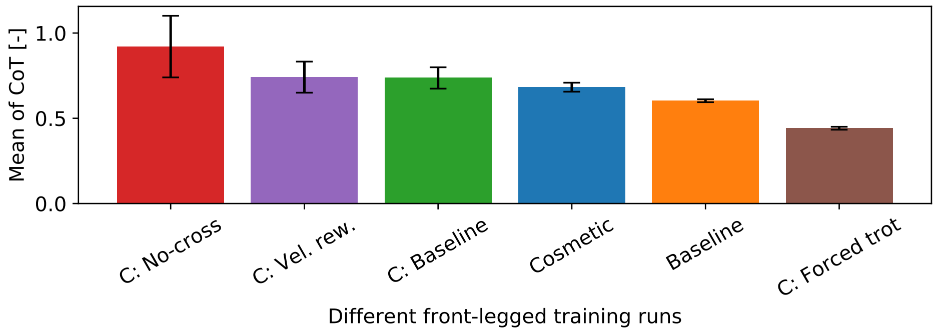
The front-legged **Baseline gait** does already very well in the CoT metric even without any Cardan mechanism implementation. The straightforward implementation of Cardan as **Cardan: Baseline** did worse than the baseline without Cardan. That is because the resulting gait was not synchronized with the walking rhythm. Forcing the motors to spin and, therefore, enforcing a trot covered the gap between the asynchrony of the walking frequency and the Cardan mechanism oscillations, which was probably too large for PPO to cross. The experiment labeled **C: No-cross** in the 6.21 shows another training run that penalized the Cardan motor shaft for changing the direction of motion (crossing zero velocity) in order to minimize oscillations. This was not enough and the gait ranks last in the chosen training runs. The **C: Forced trot gait** ranks the best in CoT and seems to provide a good result for this thesis, suggesting that utilizing the Cardan mechanism fully leads to efficiency.

6.5 Overall training results

To finish up the results, an overview of CoT of all training runs is provided in 6.22. Generally, the original kinematic configuration has 4 out of 7 runs in the best half of the CoT score order. However, the front-legged configuration and front-legged configuration with Cardan and forced trot occupy the second and the first place in the CoT order, respectively. This shows that the configuration using the full potential of Cardan for trotting brings great results in cost efficiency and overall visual form of the gait. However, it is important to note that the Cardan mechanism is simplified in this work, and its detailed implementation is left for future work.



(a) : CoT time-series for chosen front-legged gaits, the "C" stands for Cardan



(b) : Mean ± standard deviation of CoT for chosen front-legged gaits, the "C" stands for Cardan

Figure 6.21: Comparison of chosen front-legged runs

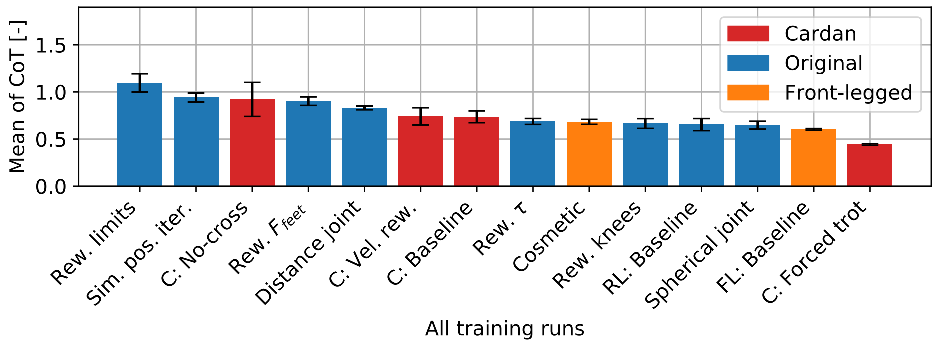


Figure 6.22: Overview of mean CoT of all training runs together

6.5.1 Payload stability assessment

Even though the Cost of Transport is a physically valid and understandable metric, it does not include information about the smoothness of the gait, although it is not completely unrelated. Roll and pitch in the movement of the body do not matter much for the overall gait, but they matter when mounting payload and/or sensors onto the robot. Therefore, an additional metric to the CoT is proposed: **Payload stability quotient** (PSQ), described as:

$$\mathrm{PSQ} = \frac{v^z_{\mathrm{base}} + \omega^x_{\mathrm{base}} + \omega^y_{\mathrm{base}}}{|v_{\mathrm{XY}}|} \tag{6.2}$$

This metric takes into consideration the linear velocity in the Z axis v^z_{base} (movement up and down), roll and pitch angular velocities (movement around the X and Y axis, ω^x_{base} and ω^y_{base} respectively), and the overall linear velocity of the robot in the XY plane $|v_{\mathrm{XY}}|$. The angular and linear velocities are not in the same units ([rad/s] vs. [m/s]). This theoretical detail is neglected as the metric is used as a quotient, the scales of the described velocities are in a similar range, and the units are not critical to the usefulness of the metric. The best-ranking gait in the CoT metric was overtaken by other gaits, and the best gait according to PSQ metric is the **FL: Baseline**. This provides another point of view on the previously favorable **C: Forced trot** gait. For future work, it is important to investigate further reward shaping to disincentivize the agent from learning a gait that is effective but cannot transport payload safely or perform other activities such as scanning of surroundings without the need for substantial postprocessing of the gathered data. The ordering of the gaits according to PSQ can be seen in Fig. 6.23. The lower the PSQ value, the more stable the potential payload.

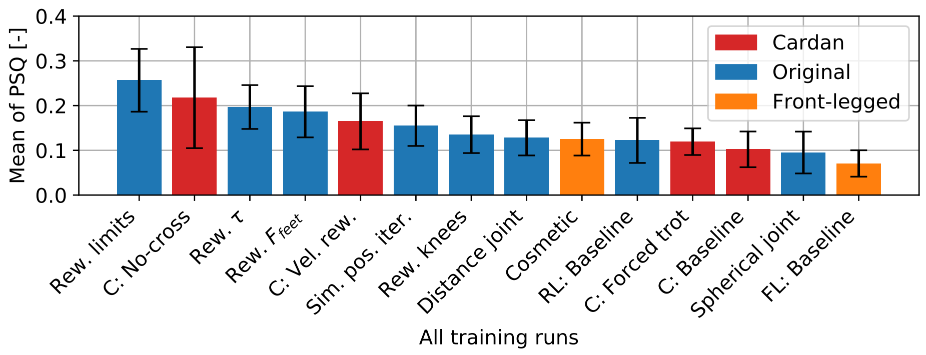


Figure 6.23: Overview of mean PSQ of all training runs together

Chapter 7

Conclusion

The goal of this work was to implement special kinematics that are specific to the robotic platform Artaban and compare it to conventional kinematics. The special kinematics include the Cardan mechanism containing one four-link mechanism that transfers torque and the second four-link mechanism on the rear legs. In order to benchmark each part of the mechanism, three kinematic configurations were tested: the original one with front and rear legs different (including only the second four-link mechanism for the rear legs), the front-legged configuration with both pairs of legs the same as in front, and the front-legged cardan configuration that models the torque transmission from motor shaft to the lower link of the legs (i.e., the first four-link mechanism). It was possible to produce some form of a gait with all three configurations; however, the original kinematic configuration, sometimes also referred to as rear-legged, met simulation difficulties. The simulation difficulties manifested in improper loop-closure implementation that was often too elastic. This should not have happened and resulted in gaits that were not valid, even though the robot was able to move around. Different measures were tried to stop this behavior from occurring. However, instead, the robot learned not to use the rear legs properly in order to avoid gaits that caused the inconsistency. Trying out different simulation step lengths and position iteration counts of the physics solver did not bring positive results. The closed articulation loop is still problematic for the simulation pipeline. Adding strength to the argument, I uncovered a bug relating to the closed-loop articulation during the implementation of this thesis. The reported bug has been confirmed and allegedly fixed. The fix will, however, be only implemented in the next release of Isaac Sim. The front-legged configuration did not cause implementation problems, and a valid gait was discovered successfully. After the front-legged baseline gait was discovered, the implementation of the Cardan mechanism followed. Using the mechanism while spinning the last motors in the legs' actuation chains in one direction, an even more efficient and visually appealing gait was produced. All of the described gaits were qualitatively compared on the basis of the Cost of Transport metric and produced a favorable outcome: the gait using the patented Cardan mechanism for walking was the most efficient of all of the discovered gaits. This gives hope for future work with this mechanism in taking advantage of its hinted efficiency. To assess the future trajectories of the robot's gait, an additional metric, named the Payload stability quotient (PSQ), was proposed. This metric measured how stable is the center of gravity of the robot relative to the z-axis linear movement and x and y-axis angular movement, normalized by the linear speed in the XY-plane. The lower this metric, the more stable per velocity unit the payload would be. The gait produced by the Cardan mechanism brought additional angular movements of the body around the x-axis, and therefore, it did not rank as well in this metric; however, stayed in the top five.

With this section, all of the goals set in the thesis assignment were achieved, starting with the review of common approaches to DRL for quadrupedal gait, an overview of different approaches to building the robot was shown in Chapter 3, Cardan kinematics were described in multiple sections and the training progression and comparison in terms of load cycles (CoT metric) was provided in the Chapter 6.

Chapter 8

Discussion & Future work

8.1 Discussion

This work explored implementations of different kinematic configurations of the Artaban robotic platform. The original kinematic configuration faced simulation difficulties that should be investigated in greater depth than was possible in the scope of this thesis. Simulating the rear legs one way or the other will be important in crossing the sim-to-real gap, which is notoriously hard to cross. Any infidelities in the simulation of the rear legs will make it hard for the agent to control its motion reliably when the actual behavior of the hind legs is different. An assumption is therefore made that the findings in this work, summarized in Chapter 7, will be similarly effective on the original kinematic configuration as they were on the front-legged configuration when the rear legs are stably simulated.

8.2 Future work

This thesis is only the beginning of the reinforcement learning journey of the Artaban robotic platform. In this thesis, only a flat terrain was included in the training, and the importance of including different terrains and further domain randomization was shown to be great for the robustness and overall quality of the produced gait. Therefore, curricular environments, training with vision, and/or laser scanning of the environment are left for further work. The horizon of possibilities does not necessarily stop at locomotion only. The trends in robotics show that robots that only walk are already the baseline and are expected to be able to traverse difficult terrains with ease and agility, able to perform simultaneous mapping and localization, perform smart tasks using large language models, and interact with the surrounding with either its legs (opening doors, kicking ball) or an additional robotic arm that often needs to be implemented completely into the robots sensing pipeline.

Bibliography

[1] Marco Hutter. Anymal - robotic systems lab. `https://rsl.ethz.ch/robots-media/anymal.html`, 2023. Accessed: 2023-12-29.

[2] Isaac gym: End-to-end gpu-accelerated reinforcement learning. `https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s32037/`, 2023. Accessed: 2023-12-29.

[3] Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Netw*, 21(4):642–653, May 2008.

[4] A. Agrawal. *Model-Based Design for Legged Robots: Predictive Control and Reinforcement Learning*. PhD thesis, UC Berkeley, 2022. ProQuest ID: Agrawal_berkeley_0028E_21920. Merritt ID: ark:/13030/m557908b. Retrieved from `https://escholarship.org/uc/item/9qc8d6kz`.

[5] Joseph Norby, Yanhao Yang, Ardalan Tajbakhsh, Jiming Ren, Justin K. Yim, Alexandra Stutt, Qishun Yu, Nikolai Flowers, and Aaron M. Johnson. Quad-SDK: Full stack software framework for agile quadrupedal locomotion. In *ICRA Workshop on Legged Robots*, May 2022.

[6] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *CoRR*, abs/2010.11251, 2020.

[7] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *CoRR*, abs/2201.08117, 2022.

[8] Princeton University. A quiet revolution in robotics - vladlen koltun, sept 12, 2023, September 2023.

[9] T.S.Eliot. The hollow men, 1925.

[10] Teymur Azayev and Karel Zimmerman. Blind hexapod locomotion in complex terrain with gait adaptation using deep reinforcement learning and classification. *Journal of Intelligent Robotic Systems*, 99, 09 2020.

[11] Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning, 2022.

[12] Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning. *CoRR*, abs/1909.08399, 2019.

[13] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. *CoRR*, abs/2109.11978, 2021.

[14] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2023.

[15] Miloš Prágr, Petr Čížek, and Jan Faigl. Cost of transport estimation for legged robot based on terrain features inference from aerial scan. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1745–1750, 2018.

[16] Kenneth Waldron and James Schmiedeler. *Kinematics*, pages 9–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[17] Ligang Yao, Hao Yu, and Zongxing Lu. Design and driving model for the quadruped robot: An elucidating draft. *Advances in Mechanical Engineering*, 13(4):16878140211009035, 2021.

[18] Steven D. Potter, Zachary John Jackowski, and Adam Young. Screw actuator for a legged robot, 6 2018. Application number US15/380,561.

[19] Ben Katz. Building all the robots. `https://build-its-inprogress.blogspot.com/2019/11/building-all-robots.html`, November 2019. Accessed: 2023-12-29.

[20] Cardan joints din 808 form e normal version with sliding fit g. `https://www.mbo-osswald.de/en/shop/cardan-joints-din-808-form-e-normal-version-with-sliding-fit-g`, 2023. Accessed: 2023-12-29.

[21] Ing. Radoslav Balajka. Robot leg, 9 2021. Application number SK9321Y1, filed on 2021-02-12 and published on 2021-09-29. Priority to PCT/SK2022/050001 on 2022-02-14.

[22] [Author's Name]. Minitaur. Master's thesis, Florida State University, Tallahassee, FL, Year of Publication. Accessed: 2023-12-29.

[23] Evan Ackerman. Ghost robotics' minitaur quadruped conquers stairs, doors, and fences and is somehow affordable. *IEEE Spectrum*, Sep 2016. Accessed: 2023-12-29.

[24] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[25] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[26] Sergey Levine. Deep reinforcement learning: Actor-critic algorithms, October 2017.

[27] Pumperla Max and Ferguson Kevin. *Deep Learning and the Game of Go*. Manning Publications Co., 2019.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[29] Papers With Code. An overview of policy gradient methods.

[30] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[31] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning, 2002.

[32] Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Simon Bøgh, and Nestor Arana-Arexolaleiba. skrl: Modular and flexible library for reinforcement learning. *Journal of Machine Learning Research*, 24(254):1–9, 2023.

[33] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[34] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[35] Denys Makoviichuk and Viktor Makoviychuk. rl-games: A high-performance framework for reinforcement learning. `https://github.com/Denys88/rl_games`, May 2021.

[36] Taewoo Kim, Minsu Jang, and Jaehong Kim. A survey on simulation environments for reinforcement learning. *2021 18th International Conference on Ubiquitous Robots (UR)*, pages 63–67, 2021.

[37] Pixar Animation Studios. Universal scene description. `https://openusd.org/release/index.html`, 2021. Accessed: 2023-12-29.

[38] NVIDIA Corporation. Nvidia physx sdk 4.1 documentation. `https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html`, April 2021. Accessed: 2023-12-29.

[39] NVIDIA Corporation. Isaac sim. `https://docs.omniverse.nvidia.com/isaacsim/latest/overview.html`, 2023. Accessed: 2023-12-29.

[40] Oyndamola. Ros discourse forum, nvidia isaac sim latest release update with gazebo bridge and new features. `https://shorturl.at/dgxA1`, June 2022. Accessed: 2023-12-29.

[41] Gymnasium documentation. `https://gymnasium.farama.org/`, 2023. Accessed: 2023-12-29.

[42] Openai gym beta. `https://openai.com/research/openai-gym-beta`, April 2016. Accessed: 2023-12-29.

[43] NVIDIA Corporation. Reinforcement learning environments for omniverse isaac gym. `https://github.com/NVIDIA-Omniverse/OmniIsaacGymEnvs`, 2023. Accessed: 2023-12-29.

[44] NVIDIA Corporation. Articulations - omniverse extensions. `https://docs.omniverse.nvidia.com/extensions/latest/ext_physics/articulations.html`, 2023. Accessed: 2023-12-29.

[45] NVIDIA Corporation. Articulations - nvidia physx sdk 4.0 documentation. `https://gameworksdocs.nvidia.com/PhysX/4.0/documentation/PhysXGuide/Manual/Articulations.html`, 2018. Accessed: 2023-12-29.

[46] J. L. McGaugh. Searching for memory in the brain: Confronting the collusion of cells and systems. In F. Bermúdez-Rattoni, editor, *Neural Plasticity and Memory: From Genes to Brain Imaging*, chapter 1. CRC Press/Taylor & Francis, Boca Raton (FL), 2007.

[47] Jack Koch, Lauro Langosco, Jacob Pfau, James Le, and Lee Sharkey. Objective robustness in deep reinforcement learning. *CoRR*, abs/2105.14111, 2021.

[48] dhajnes. Github issue, `get_world_poses()` and `set_world_poses()` getting/setting orientation outside of the `base_link`. `https://github.com/NVIDIA-Omniverse/OmniIsaacGymEnvs/issues/110`, 2023. Issue #110 opened on November 16, 2023.

[49] Zuzana Mačicová. Implementácia aproximácie priamej a inverznej kinematiky paralelného mechanizmu nôh robota artaban. Bakalárska práca, Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, 2021.

https://opac.crzp.sk/?fn=detailBiblioFormChildUM4PS&sid=A770A18E340C6018B58DE7BDD5C2&seo=CRZP-detail-kniha.

[50] Chin Pei Tang. Lagrangian dynamic formulation of a four-bar mechanism with minimal coordinates. 2010.

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kružliak Andrej**	Personal ID number:	**483496**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Reinforcement Learning for Quadrupedal Robot Control with Novel Kinematics

Master’s thesis title in Czech:

Posilované učení pro ovládání čtyřnohého robota s inovativní kinematikou

Guidelines:

1) Study the conventional solutions for kinematics of quadrupedal robots.
2) Study the conventional approaches to solve quadrupedal gait using reinforcement learning.
3) Implement the special properties of novel kinematics* into the simulation environment.
4) Employ reinforcement learning algorithm of choice to solve the quadrupedal gait in the simulation.
5) Verify the implemented control system in the simulation environment in the terms of load cycles.
* novel kinematics relates to a patented joint

Bibliography / sources:

[1] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter (2022). Learning robust perceptive locomotion for quadrupedal robots in the wild. Science Robotics, 62(7), 317- 328.
[2] T. Azayev & K. Zimmerman (2022). Blind Hexapod Locomotion in Complex Terrain with Gait Adaptation Using Deep Reinforcement Learning and Classification. Journal of Intelligent & Robotic Systems, 3-4(99), 659-671.
[3] N. Rudin, D. Hoeller, P. Reist & M. Hutter (2022). Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning, ArXiv arXiv:2109.11978v3.
[4] V. Tsounis, M. Alge, J. Lee, F. Farshidian & M. Hutter (2020). Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning. IEEE Robotics and Automation Letters, 2(5) (3699-3706).

Name and workplace of master’s thesis supervisor:

Rastislav Marko, MSc. Panza Robotics, s.r.o.,Trnava, Slovakia

Name and workplace of second master’s thesis supervisor or consultant:

doc. Ing. Karel Zimmermann, Ph.D. Vision for Robotics and Autonomous Systems FEE

Date of master’s thesis assignment: **21.06.2023** Deadline for master's thesis submission: **09.01.2024**

Assignment valid until: **19.02.2025**

Rastislav Marko, MSc.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt   Student’s signature

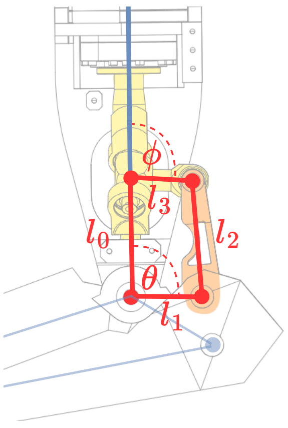
Appendix A

Cardan mechanism to knee angle computation

The following set of equations A.1a describes the computation of the angle of the knee θ from the angle of the clevis fork of the Cardan mechanism ϕ. The lengths of the four-link l_i correspond to the l_i noted in the Fig. A.1b. The complete methodology behind the given equations can be found in [49] and in [50].

$$
\begin{aligned}
k_1 &= -2l_1l_3\sin(\phi) \\
k_2 &= -2l_1(l_0 + l_3\cos(\phi)) \\
k_3 &= l_0^2 + l_1^2 - l_2^2 + l_3^2 + 2l_0l_3\cos(\phi) \\
\theta &= 2\arctan 2\left(-k_1 + \sqrt{k_1^2 + k_2^2 - k_3^2}, \quad k_3 - k_2\right)
\end{aligned}
$$

(a) : Cardan mechanism to knee angle computation



(b) : Four-bar Cardan mechanism, shown on the front leg

Figure A.1: Computation of the knee angle from the Cardan mechanism

Appendix B

Detailed reward function definition

The given Table B.1 describes the mathematical definition of the reward signals. The capital letter **R** stands for the given reward scale adjusted for the simulation time step as $R = r \times dt$ where r is the reward scale relevant for the given reward as seen, for example, in Table 6.1 and dt is the simulation step size. If the simulation runs at 100 FPS, that means that $dt = 1/100 = 0.01$. Those reward signal definitions were built on top of the `Anymal` environment of OIGE by Nvidia [43], which provided a meaningful starting point for the reward shaping that followed.

Reward name	Reward definition
Linear XY vel.	$\exp(-(v^{\text{XY}}_{\text{cmd}} - v^{\text{XY}}_{\text{base}})^2/0.25) \times R$
Angular Z vel.	$\exp(-(\omega^{\text{XY}}_{\text{cmd}} - \omega^{\text{XY}}_{\text{base}})^2/0.25) \times R$
Linear Z vel.	$(v^{Z}_{base})^2 \times R$
Joint acc.	$\sum_{j\in\mathcal{J}}[a_j^2] \times R$
Joint vel.	$\sum_{j\in\mathcal{J}}[\lVert v_j \rVert] \times R$
Action rate	$\sum_{a^\pi\in\mathcal{A}^\pi}[(a^\pi - a^\pi_{\text{prev}})^2] \times R$
Cosmetic	$\sum_{j\in\mathcal{J}_{\text{rockers}}}[\lVert\theta_j - \theta_j^{\text{default}}\rVert] \times R$
Soft limit	$\sum_{j\in\mathcal{J}}[\ 1\ \texttt{if}\ \lVert\theta_j - \theta_j^{\text{limit}}\rVert < \epsilon,\ \texttt{else}\ 0\ ] \times R$
Base height	$(x^{\text{Z}}_{\text{base}} - z^{\text{default}}_{\text{base}})^2 \times R$
Knee height	$\sum_{j\in\mathcal{J}_{\text{knees}}}[(x_j^{\text{Z}} - z_j^{\text{default}})^2] \times R$
Fallen over	$(\texttt{if fallenOver==True 1, else 0}) \times R$
Feet force	$\sum_{j\in\mathcal{J}_{\text{feet}}}[\lVert F_j - F_{\max}\rVert] \times R$
Collision	$\sum_{j\in\mathcal{J}}[\texttt{if forbiddenCollision(}j\texttt{)==True 1, else 0}] \times R$
Motor C zero-cross	$\sum_{m\in\mathcal{M}_C}[\texttt{if}\ \lVert v_m\rVert < \epsilon\ \texttt{1, else 0}] \times R$
Motor C vel.	$\sum_{m\in\mathcal{M}_C}[v_m^2] \times R$

Table B.1: Computation of reward signals for reinforcement learning. $\mathcal{J}$ represents set of all joints, A^π represents number of actions available to the agent, e.g., 12 in the case of 12 motors, x, v and ω represent the robots position state, linear and angular velocity respectively, θ represents a motor or joint angular position, F_j represents force measured in the link forward from the $j - th$ joint, $\mathcal{M}_C$ represents the set of C motors which are responsible for spinning the Cardan mechanism for the Cardan kinematic configuration.

Appendix C

Original kinematic configuration — photography strips of gaits

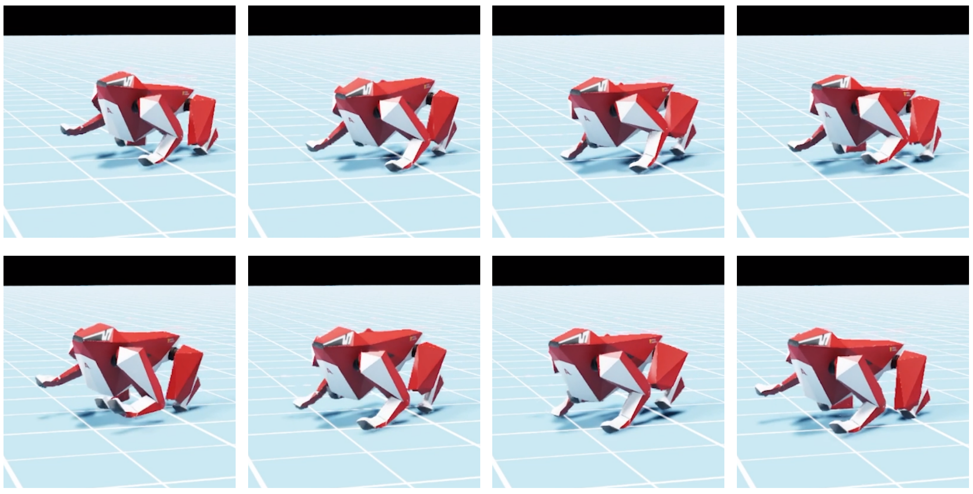


Figure C.1: The baseline gait for the original kinematic configuration of Artaban

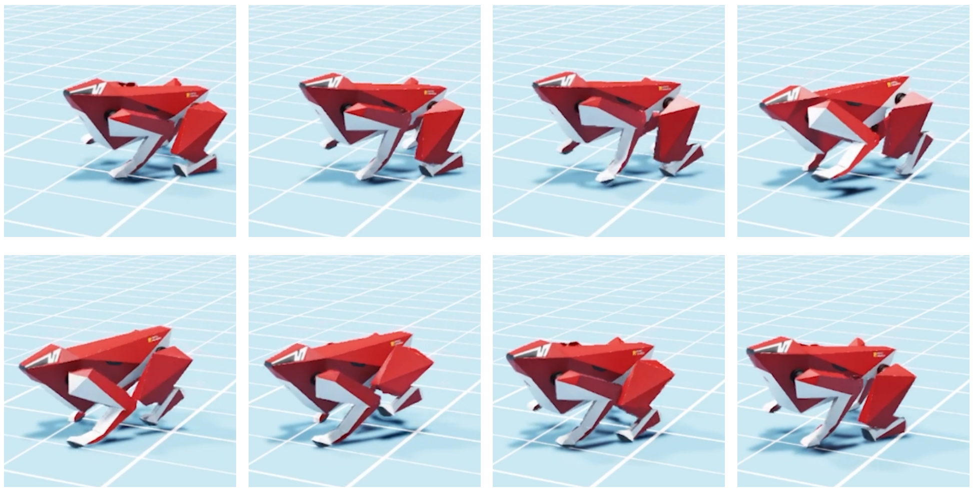


Figure C.2: The gait for the original kinematic configuration of Artaban with penalties incentivizing lower knee height error

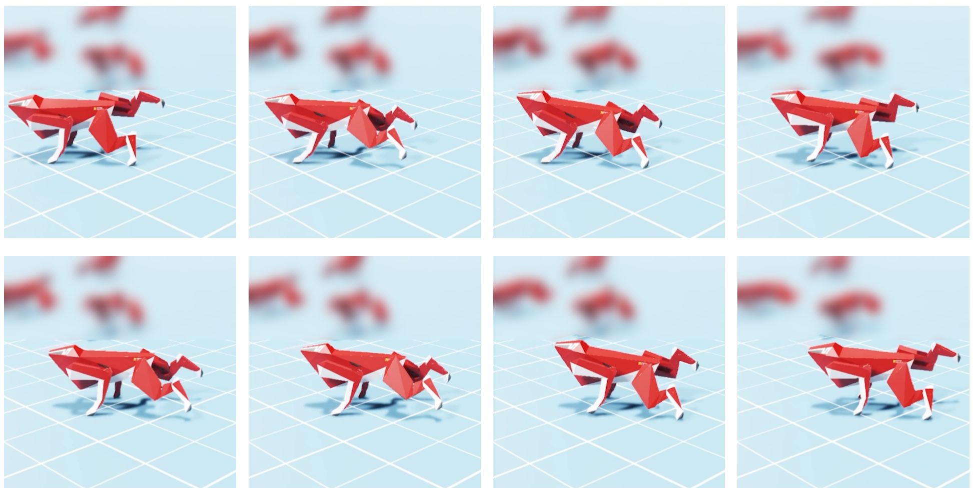


Figure C.3: The training run with penalties on being close to joint limits

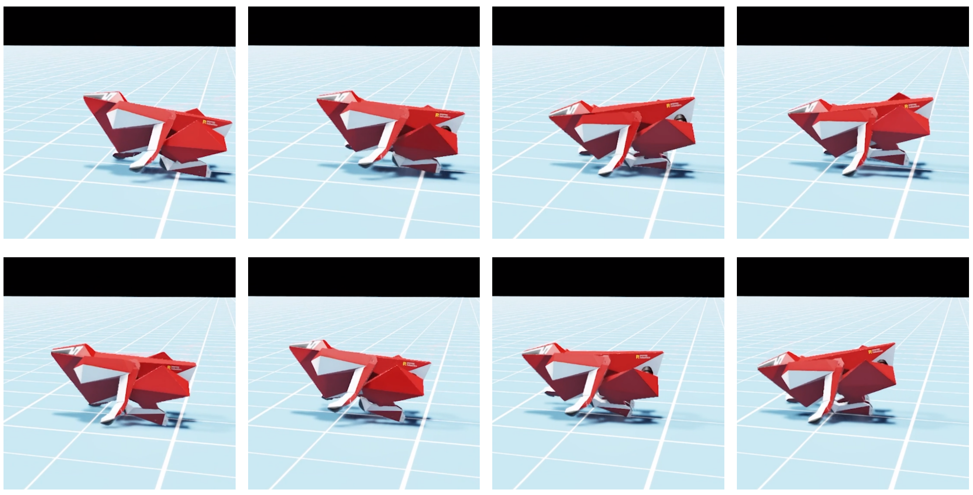


Figure C.4: The gait for the original kinematic configuration of Artaban with distance joints

Appendix D

Front-legged kinematic configuration — photography strips of gaits

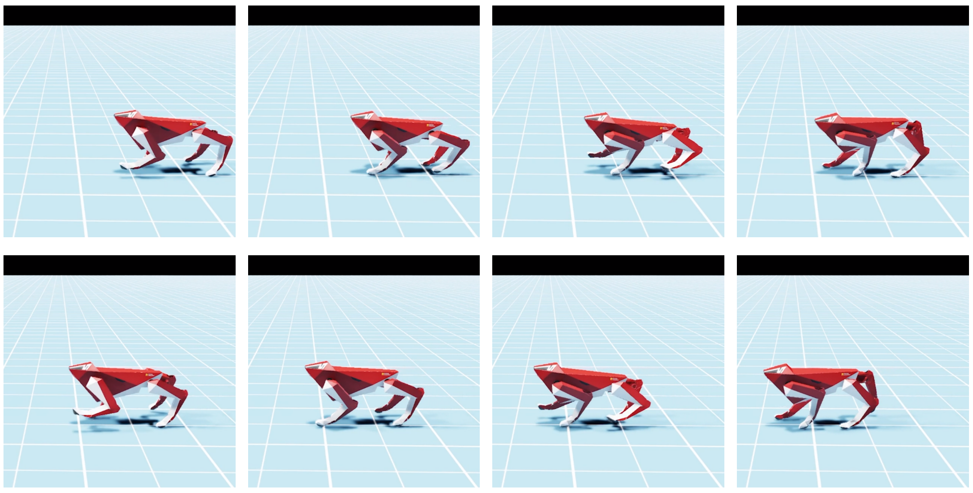


Figure D.1: The baseline gait for the front-legged kinematic configuration of Artaban

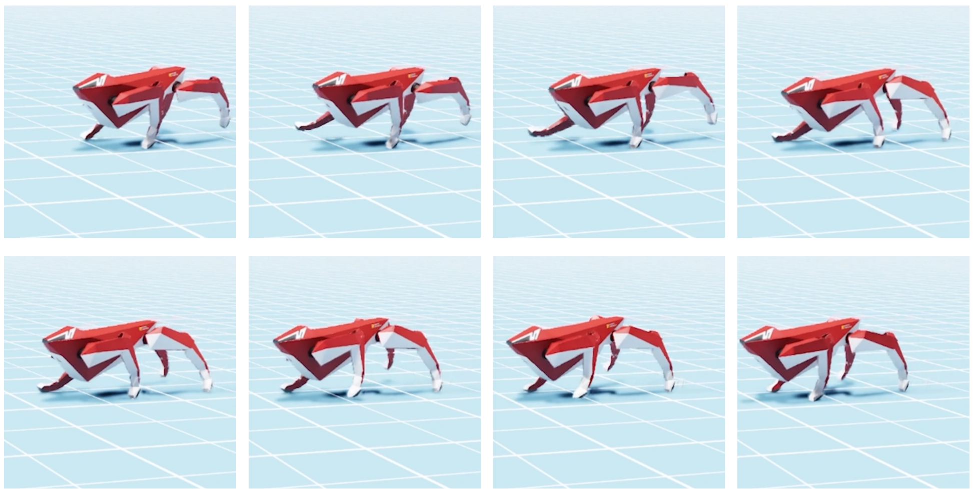


Figure D.2: The gait for the front-legged kinematic configuration of Artaban with rewards incentivizing placement of rocker joints close to nominal position

Appendix E

Cardan: front-legged kinematic configuration — photography strips of gaits

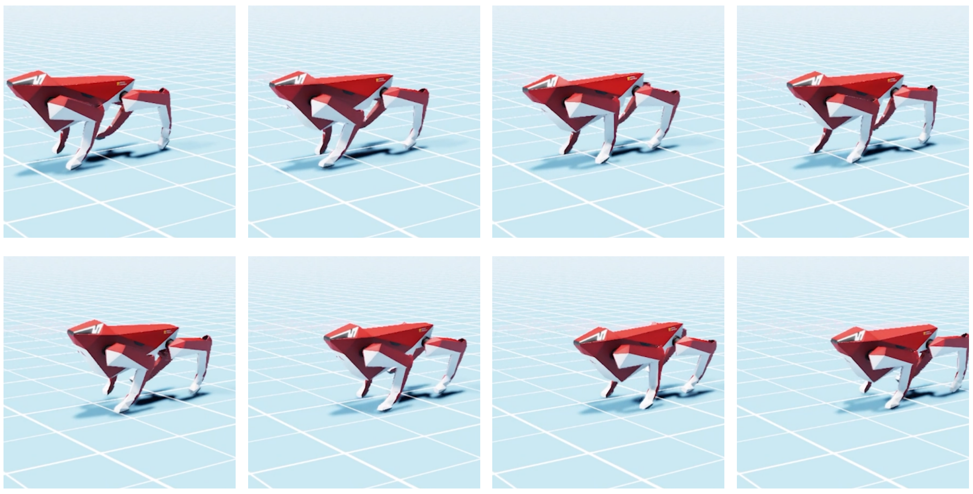


Figure E.1: The gait for the front-legged kinematic configuration of Artaban with Cardan mechanism in place

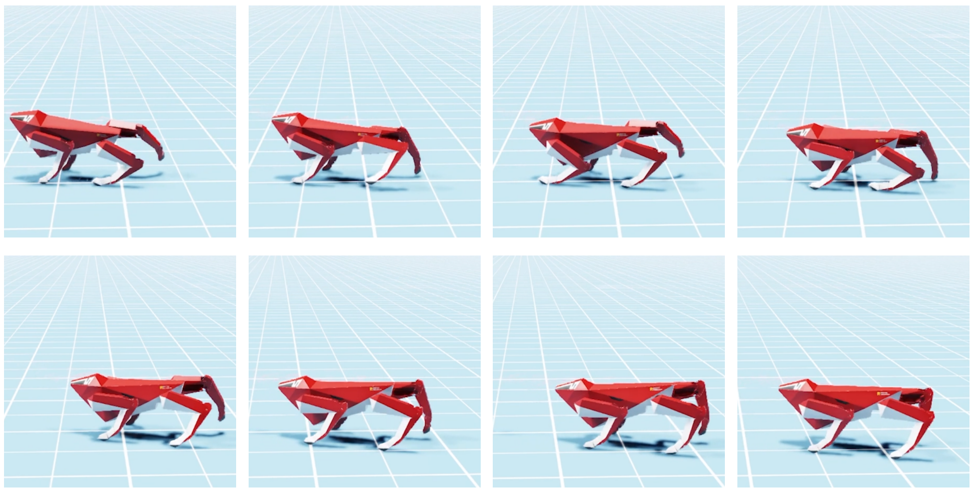


Figure E.2: The gait for the front-legged kinematic configuration of Artaban with Cardan mechanism in place with rewards incentivizing higher velocity of the motor shaft on the mechanism

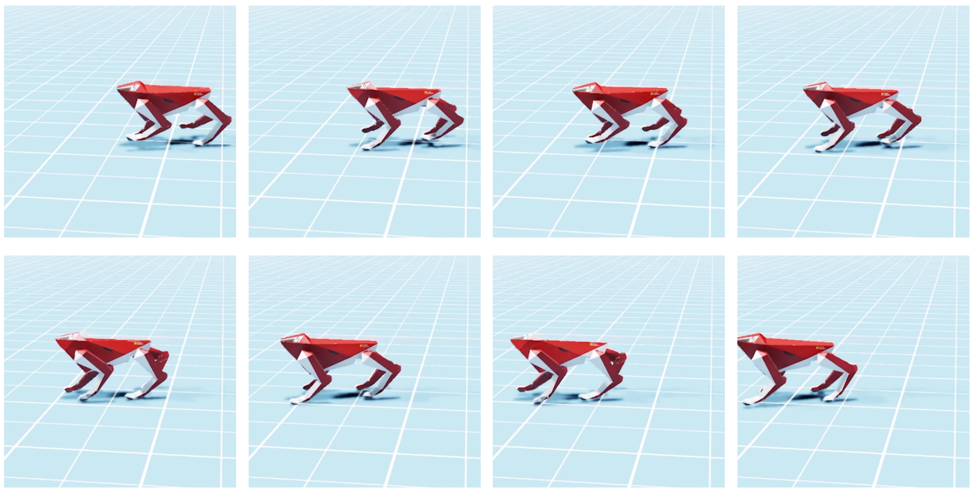


Figure E.3: The gait for the front-legged kinematic configuration of Artaban with forced Cardan mechanism trot

Appendix F

Video documentation of the provided gaits

A file `VIDEO_APPENDIX_SUBSET.zip` is uploaded to the appendix section of the thesis. It contains folders `video_subset_front_legs` and `video_subset_rear_legs` carrying videos of the most important results, and a text file `README.txt` that includes further instructions on how to watch the videos in the most convenient manner. The videos are only a bare subset in order to respect the file capacity cap of 100 MB per work (thesis + appendix). The rest of the videos documenting the results of different training runs are uploaded to two unlisted `YouTube` playlists. Links to those playlists are attached in the `README.txt` file.