Reinforcement learning of a navigation method for contact planning on humanoid robots

Jason Chemin

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *28 novembre 2022* **par :**

Jason CHEMIN

Reinforcement learning of a navigation method for contact planning on humanoid robots

---

JURY

Julien PETTRÉ	Directeur de recherche	Rapporteur
Sylvain CALINON	Senior Research Scientist	Rapporteur
Serena IVALDI	Chargée de recherche	Examinatrice
Christine CHEVALLEREAU	Directeur de recherche	Examinatrice
Florent LAMIRAUX	Directeur de recherche	Examinateur
Nicolas MANSARD	Directeur de recherche	Examinateur
Steve TONNEAU	Lecturer (associate professor)	Examinateur

École doctorale et spécialité :
EDSYS : Robotique 4200046

Unité de Recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes (UPR 8001)

Directeur(s) de Thèse :
Nicolas MANSARD et *Steve TONNEAU*

Rapporteurs :
Julien PETTRÉ et *Sylvain CALINON*

Remerciements

“En vérité, le chemin importe peu, la volonté d’arriver suffit à tout."

— Albert Camus*, Le Mythe de Sisyphe*

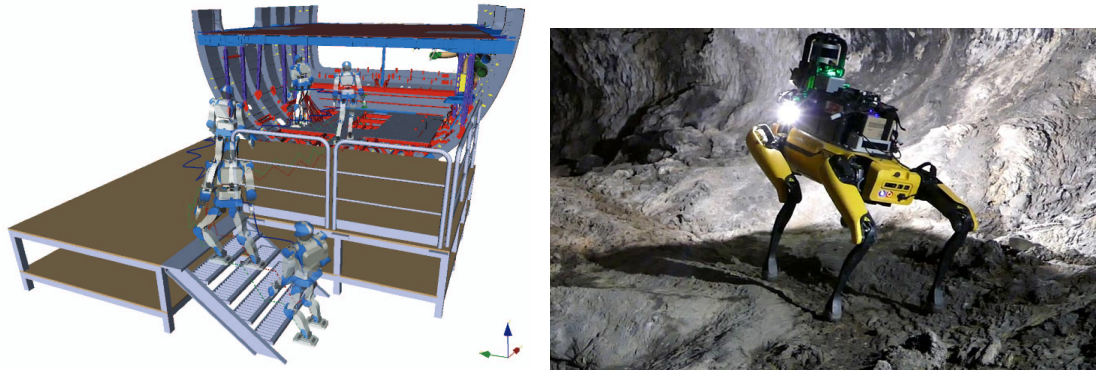
Ces années de thèse n’étaient sûrement pas les plus faciles, entre le covid et les problèmes qui ont pu survenir, mais elles sont passées. Merci au soutien de mes directeurs de thèse pour leur aide et guidance tout au long de cette période, surtout sur la fin qui a été intense. Merci bien sûr à ma famille pour qui a été là quand j’en avais besoin. Et enfin, je ne remercierai jamais assez tous mes collègues de l’équipe Gepetto. Heureusement que vous étiez là !

Contents

Chapter 1

Introduction

Contents



(a) (b)

Figure 1.1: Legged robots are currently opening a new area of robotics capabilities, beyond what wheeled manipulators are able to reach, in many domains such as aerospace manufacturing or underground exploration... and later in our homes? Sources: (a) Caron et al. [CK16] and (b) ©NASA/JPL-Caltech.

Robots are already essential tools in the industry and will play a part in our daily life in the near future. However, most of them still require specifically designed environments to perform their task. In recent years, research on legged robots has opened a whole new range of possibilities. These robots could operate in less structured industrial areas to perform various tasks just like us (Figure 1.1a). They could also explore in our stead risky environments as demonstrated during the DARPA subterranean challenge [Agh+21; Tra+22], where the robots have to map, navigate, and search for casualties in complex underground environments (Figure 1.1b).

Nevertheless, to achieve these tasks, they have yet to perform the most basic but challenging skill that is to *locomote through the environment*.

Figure 1.2: Planning the path is crucial for locomotion on complex terrains. While the blue trajectory is likely an easy output of our supercomputer human brain, asserting its feasibility can yet only be achieved by actually walking the path with our body.

1.1 Legged locomotion in complex environments

In the large collection of works on legged robots, several strategies emerged to solve this problem. In this thesis, we are interested in the robot locomotion problem in complex environments, that can be solved with a similar decision process to their creators.

We humans can achieve this task in real-time. As shown in Figure 1.2, the objective is as follows: "how to reach the other side of this terrain?" Here, this task is particularly difficult, hence thorough planning is required. We can typically decompose this task into two sub-problems:

1. *What path do we take?* This decision is based on an estimation of our capabilities. First, the path is subject to conditions of *reachability*, as we need to be able to touch the ground, and obviously of *collision avoidance* as we can not go through obstacles. Second, we need to evaluate the terrain *traversability* to plan a feasible path. Based on these criteria, we decide to plan the blue path in our example.

2. *How do we move our body to follow the path?* Walking without thinking about where to place my foot could be sufficient for most scenarios. However, difficult terrains such as this one require a careful *contact planning* to avoid taking a wrong step and falling.

Following our human intuition, we approach these questions in two stages with (1) a navigation task to plan a feasible path, and (2) walking along this path while carefully planning our contact on the terrain.

The key difficulty is that the two stages are intricate, as the only complete way to solve (1) is to also solve (2). Again, we can have the intuition that our brain works with simplified models representing feasible solutions of (2) when exploring the candidate paths in (1).

However, reproducing the masterful human reasoning for locomotion remains yet a difficult problem. How to program robots to achieve this decision process? Furthermore, can we automatically deduce navigation models handling the first stage (1), from the empirical capabilities of the contact planner (2), with the hope to extend them to other robot morphologies or capabilities?

1.2 Thesis Statement and Summary

This thesis builds upon the so-called Loco3D framework [Car+17; Ton15] whose goal is to achieve a fast computing and safe solution for legged robot locomotion in complex environments. At the core of the locomotion workflow is a clever model of the robot locomotion capabilities, the reachability model [Ton+15], used to simplify the legged navigation planning. We will discuss the importance of this model in locomotion planning, but also its limitations in the next chapter.

Our research topic is to provide a better alternative to this model by replacing the human intuition used to design it with a systematic evaluation of the robot locomotion feasibility based on data generation and machine learning.

Our main contribution is a local navigation method learned by reinforcement. Our method, named LEAS, can locally navigate under reachability and collision-avoidance constraints using a local observation of its environment.

The organization of this thesis is as follows:

Chapter 2 presents a review of the works on legged robot locomotion. We explore different solutions to obtain a safe and robust locomotion, leading us to our choice of a navigation method prior to contact planning (also known as the motion-before-contact approach), from which we formulate our main contribution.

Chapter 3 presents our steering method LEAS, that can locally navigate complex terrains. We describe our method to learn by reinforcement how to generate paths under reachability and collision-avoidance constraints. The capabilities of LEAS are then empirically explored, exemplified, and characterized using a simple feasibility oracle.

Chapter 4 presents the results of LEAS using the acyclic sampling-based contact planner [Ton+18a] as a feasibility oracle. Our steering method learns how to generate paths fitting this contact planner. With this setting, LEAS can improve the performance of the planner using the reachability condition, without requiring the tuning of the intuition-based model.

Chapter 5 generalizes the training of LEAS with the more advanced contact planners, MIP and SL1M [Son+20]. We explain their formulation and their limitations relative to the path. Finally, we present the current results as well as the different experiments we conducted.

Chapter 6 discusses the advantages and limitations of our steering method, finally concluding with the perspective of our work.

Chapter 2

Background

Contents

Our contribution in this thesis aims at addressing the very complexity of the locomotion problem, where the decision on continuous motion variables and discrete contact locations have to be taken from an intricate process. The study of this problem started more than 50 years now, with very quickly the ambition of getting a complete solution, i.e. a complete motion controller driving the locomotion of real robots, and later on, dynamic avatars evolving in physical simulation subject to realistic constraints.

In this chapter, we will first browse the various approaches that have been historically proposed, until recent locomotion controllers able to optimize or learn all degrees of freedom at once (Section 2.1). This will help us understanding the key aspect of locomotion: the intricated decision of motion and contact. We will then focus on one subpart of the locomotion problem: contact planning (Section 2.2). In this thesis, we will be interested in one particular approach to tackle this problem called motion-before-contact, that we will argue to be promising for the main realistic locomotion problems, that we will connect to the navigation problem. In Section 2.3, we will list the main existing solution to navigate in a 3D environment with legged system. Finally, we will conclude the chapter by defining our thesis propositions and explaining how they contribute to the state of the art.

2.1 Synthesizing Locomotion

Research on modeling legged robot movements for locomotion is a long searched topic [FDY17]. The task of determining stable and feasible motions is performed by the *whole body controllers* (i.e. controller able to take real-time decisions about all the robot degrees of freedom whole body). They are among the most difficult controllers to engineer due to their numerous stability criteria [Wes+07; Kaj+14].

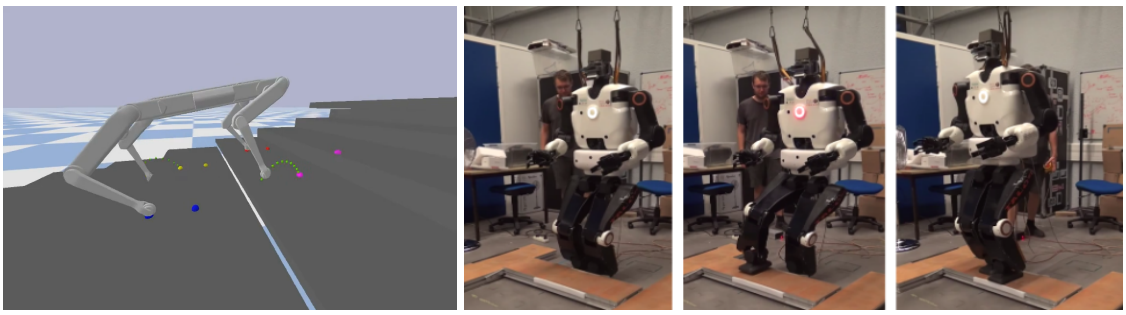
We propose here to explore these methods by first obtaining a complete movement on the real robot in a realistic simulator. This exploration will help us understanding the importance of planning the future contact of the system in its environment. As we will see, while contact planning can initially be considered aside from the whole body controller, recent advances tend to more effectively consider both problems together

We will browse through the panel of existing locomotion controllers, that we can explore following 2 criteria:

(1) Contact decision, that can be predefined, internal, or free.

(2) Model complexity, if the model is templated (e.g. Linear inverted pendulum model), reduced, or complete.

2.1.1 Predefined Contacts



(a) Risbourg et al. [Ris+22] (b) Dantec et al. [Dan+22]

Figure 2.1: Whole body controllers performing predefined contacts.

Given a predefined contact sequence, efficient methods exist to compute the corresponding stable whole-body motion. Knowing the dynamic state of the robot (e.g. position and velocity of its basis and all its joints) as well as its current and future contacts, approximation models can be used to plan the whole-body locomotion of legged characters.

Model-based approaches. Due to the high complexity of the locomotion problem, classical whole-body controllers of this category often rely on simplified dynamic robot models. In the seminal work, Kajita et al. [Kaj+03] introduced some key methods of the robot locomotion field: the linear inverse pendulum and the zero moment point. This contribution is the basis of numerous works based mainly on the study of the centroidal dynamics for walking robots [Ton+18b; Car+18]. Given a contact sequence, some Model-Predictive Control (MPC) methods permit to compute a trajectory for the robot center of mass, while ensuring its dynamics consistency [Car+16; Léz+]. From the simplified model estimations, a whole body motion can then be generated, following it as a reference [Gri+19b]. Carpentier et al. [Car+17] use a second-order inverse kinematic to follow a reference centroidal trajectory, previously generated by their optimization method. Also, the inverse kinematics can be used to compute collision-free robot motion, such as Risbourg et al. [Ris+22] who adapt the end-effector trajectory of the quadruped robot SOLO to avoid its collision

with the environment (Figure 2.1a). Alternatively, other approaches exist to achieve all in one whole body MPC. Dantec et al. [Dan+22] propose such an MPC taking into account the whole body model to achieve dynamic locomotion on the torque-controlled humanoid robot Talos over predefined footsteps (2.1b). Their control directly computes the optimal torque to be applied on the robot without any additional trajectory or estimation.

Learning whole-body controllers. Such whole-body controllers can also be learned by reinforcement in simulation [Xie+20]. Peng et al. [Pen+17b] train a biped character in simulation to follow predefined footsteps, and achieve walking in mostly flat environments. Following this idea, Tsounis et al. [Tso+20] learn a whole-body controller in simulation for quadruped locomotion on more complex terrains. Gangapurwala et al. [Gan+22] learn whole-body motion tracking and recovery controllers for improved robustness, accounting for changes in the dynamics of the robot and perturbations. Their policy is then performed on a real quadruped robot. Combining model-based and RL approaches, Xie et al. [Xie+21] learn by reinforcement how to control the accelerations of a centroidal model, then used to compute ground reaction forces translated to joint torques applied on the robot. Their method, combined with simple heuristics for footstep placement, demonstrates robust walking in complex scenes on the quadruped robot Laikago.

To this day, such controllers are mostly applied in the real-world to quadruped robots and have yet to be performed on humanoid robots [Sin+22].

Conclusion on predefined contacts. Most of the works in the literature focus on whole-body controllers generating motions over predefined contact sequences. Model-based approaches have shown impressive locomotion skills on complex environments [Vai+14]. Reinforcement learning is another approach to obtain such controllers. The trained controllers can cope with the system dynamics along with stability criteria. However, they require up to several days of training, and are yet to achieve the results of model-based controllers on real humanoid robots.

In this thesis, we will be using Task-Space Inverse Dynamics (TSID)-based controllers [Pre+16], following the methodology introduced in the context of the Loco3D project [Car+17], yet expecting the coming generation of robot controllers to be rather base on whole body MPC [Dan+22]. In the meantime, we explored RL-based controllers. While we can hope that future discoveries will lead to the unification of these different concepts, all are indeed compatible with the approach discussed in this thesis. We will explore in Section 2.2 that what really matters is to characterize their feasibility domain.

2.1.2 Internal Contact Decision

(a) Gong et al. [Gon+18] (b) Lee et al. [Lee+20]

Figure 2.2: Legged robots walking on small variation terrains.

Flat floor. Traditional whole-body controllers for flat ground locomotion make the assumption that contacts will always occur at the same height in global space. Achieving stable, periodic walking on flat ground is already a challenging problem in itself to comprehend the nature of dynamic models for locomotion, which is especially difficult for biped robots [Gri+14]. The study of the center of mass trajectory, with the linear inverted pendulum model and zero moment point as well as their variations, has been at the core of the legged locomotion problem with model-based approaches [VB04; WTK16; Car17].

A common strategy to solve this problem is to use a reference walking motion, which is then modified by the whole-body controller to ensure its stability at runtime. Using such a controller, Chevallereau et al. [CDG08] modify the reference joint motion to obtain the desired zero moment point evolution, and thus a stable walking motion. Tsujita et al. [TTO01] propose a gait pattern controller adapting the reference motion in function of touch sensor signals on the foot of a quadruped robot.

Simplified models of robot dynamics are an effective strategy. Using a linear inverted pendulum as a reduced model of the robot, Kajita et al. [Kaj+02] propose a real-time walking pattern generator that adapts footsteps during the motion to follow a desired walking speed and direction. In this line of work, Herdt et al. [Her+10; HPW10] introduce the notion of "walking without thinking", where a model predictive control scheme takes as input a given direction to follow, and outputs safe foot placements and motion to walk seamlessly while reacting to disturbances on the humanoid robots HRP2. Such a strategy permits fast online planning [Apg+] as well as the optimization of different tasks simultaneously such as locomotion and manipulation [DLL12] or obstacle-avoidance in real-time [Nav+17].

Small variation terrains. This strategy naturally extends when the motion on flat ground is robust enough to be performed on complex terrains. It can be done by simplifying the robot dynamics model with templated models to generate a nominal reference motion considering a flat ground. This motion is then performed by the whole body controller that adapts it for complex and rough terrains. Rezazadeh et al. [RH20] include a reflex-based control scheme to walk blindly on uneven terrain with the biped robot Atrias. Building upon this work, Gong et al. [Gon+18] adapt motions from a gait library and demonstrate various locomotion tasks on the biped robot Cassie including balancing on uneven moving surfaces or walking in the sand (Figure 2.2a).

In computer graphics, learning a controller from data can produce natural and plausible walking motion [Hol+20; HKS17]. However, data-driven strategies often require a large amount of motion capture data, and cannot cope with the dynamics variations in the robot (or character) states inherent to physics-based simulations and the real world.

Reinforcement learning, and more specifically imitation learning, can overcome this limitation and bridge the gap between simulation and reality (sim-to-real). Li et al. [Li+21] learn to adapt motion from a gait library. They then achieve the sim-to-real by randomizing the system dynamics in simulation, and achieve robust locomotion on the biped robot Cassie. Using another strategy, Lee et al. [Lee+20] learn a teacher policy for walking on the quadruped robot ANYmal, that is fully aware of its surrounding environment. The safe motion generated by the teacher is then imitated by another policy, that learns how to blindly walk in complex terrains. They then demonstrate walking in the real world in very rough scenarios (Figure 2.2b). Learning residual control is another method to bridge the reality gap. Duan et al. directly learn how to modify the reference motion with residual actions to obtain a more robust bipedal walking [Dua+21].

While these works demonstrated some capabilities to walk in uneven terrains, they remain limited in more complex scenarios. Indeed, such "blind" controllers are performing a control only in reaction to impacts on the environment. They are on the opposite anticipating proper contact creation, or briefly taking advantage of more advanced knowledge about their environment. As a consequence, they are irremediably prone to collisions (and falls) on terrains such as stairs.

Simultaneous motion and contact optimization. Optimizing simultaneously contact and motion permits the consideration of the robot dynamics in the contact choice, considering a known terrain model.

Approximating the environment as a continuous function and computing contact and motion permits the generation of whole-body motion in complex terrains [DVT14; PCT14]. In simulation, Mordatch et al [MTP12] present a contact invariant method optimizing contacts and motion trajectory simultaneously to perform a wide variety of locomotion tasks. In the same line of work, Winkler et al [Win+18] present a trajectory optimization formulation generating highly dynamic motion plans for a variety of legged characters on complex terrains. Whole-body trajectory optimization was successfully applied to real quadruped robots, achieving planning on flat ground and execution of walking movements in near real-time [Win+17].

The key aspect of these approaches is to consider some piecewise-smooth representation of the contact variables. The contact sequence can then evolve continuously on a piecewise-smooth world while the motion solver explores various locomotion patterns, in a continuous (optimization-based) manner. Of course, this prevents some world representations such as Mario-style floating platforms, yet not reducing much the applicative scope. The main limitation comes from the inability of the (convex) optimization solver to handle the non-convex nature of the contact location in a non-flat world. Moreover, this also prevents the solver to discover non-regular gaits, or with some clever but non generic reformulation [Win+18].

On the other hand, it has been proposed to explicitly model the discrete nature of the contact sequence by integer variables, hence leading to an optimization problem deciding of a mix of continuous and discrete variables, called Mixed-Integer Programming (MIP) [Gurb]. With this approach, [Kui+15; Ace+19] simultaneously optimize contacts, gait and motion on their legged robot to locomote through complex terrains. However, MIP does not scale well to these large or non-linear problems, hence reducing up to now the impact of these approaches to whole body problems. On the other hand, this limitation can be solved if we can keep a linear formulation [Ton+20]. This approach will be further covered in this thesis.

Conclusion on internal contact decision. A classical approach for locomotion is to synthesize walking motion while adapting footstep placement on flat ground. Generalizing these approaches to uneven terrains is feasible but limited, as such controllers consider the terrain variations as errors during the motion that needs to be fixed. For more complex scenarios, numerical solvers fail to globalize the exploration and discover interesting locomotion patterns. We better understand the complexity of the search when explicitly formulating it as a MIP problem, then boiling down to a combinatorial exploration.

2.1.3 Contact Agnostic

Whole-body controllers previously presented implicitly or explicitly reason about contact placement on the terrain. Another strategy is to employ a contact agnostic approach, where the controllers are not given any reference motion or contact sequence. The contact sequence is then not an explicit variable, but rather a consequence of the whole-body actions, often locally based on its surrounding environment knowledge.

Optimal control with differentiable simulation. When optimizing the robot trajectory with fixed contact sequence, the reason why the solver cannot decide to change the sequence is that the contact dynamics are set as non-differentiable. Indeed, in robotics and computer graphics, we often consider rigid (stiff) contact solvers, which offer the best trade-off between algorithmic efficiency and realism [CB16]. Yet this leads to non-differentiable models that gradient-based algorithms cannot manipulate. In [HRL15] a gradient-free solver is used on top of the rigid (ODE) simulator, leading to contact exploration even if with a local exploration range.

In order to use gradient-based algorithms, finite-differencing can typically be used to approximate the gradients such as the physics engine MuJoCo [TET12]. However, it tends to introduce round-off and discretization errors. Recently, some differentiable physics engines have emerged to solve these issues, such as Nimble from Werling et al. supporting complex contact geometry and gradients approximating continuous-time elastic collision [Wer+21]. So far, this direction has not been pushed to a realistic robot setup. Current works are mostly on the simulator formulation [Hu+19a; Hu+19b] or on local smoothing using randomization [Suh+22; Le +22].

Learning how to locomote. Similarly to trajectory optimization, Reinforcement learning can explore movements to optimize an objective function. Yet they offer unprecedented flexibility in environments and the objective they can tackle. Furthermore, they only require optimization at training time, while the run-time only consists of the execution of the resulting policy. As they do not require the evaluation of the simulator gradients, they are also less prone to the limitations discussed with trajectory-based solvers related to contact differentiability, and accept classical (stiff) contact formulation.

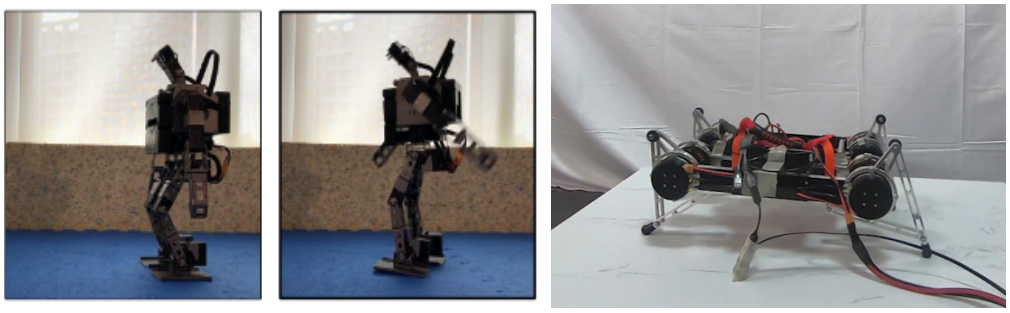
(a) Peng et al. [PBP16] (b) Heess et al. [Hee+17] (c) Won et al. [WGH22]

Figure 2.3: Works on reinforcement learning for legged characters locomotion.

Reinforcement Learning (RL) in physics-based simulation can be used for character animation [Kwi+22]. This method permits learning controllers able to cope with the system dynamics. Peng et al. learn by reinforcement a terrain-adaptive locomotion controller, outputting target joint angles to compute the torques on 2D biped and quadruped characters [PBP16] (Figure 2.3a). Using a similar control, such RL controller can be extended to 3D locomotion [Hee+17] (Figure 2.3b). These seminal works are the basis of many others further improving the naturalness and robustness of the motions as well as the diversity of skills performed [Ber+19; Luo+20; Lee+19]. More recently, Won et al. [WGH22] use conditional variational autoencoders to learn how to imitate motion from a database and achieve various tasks in a physics-based simulation. One of their results shows a biped locomotion task using a low-resolution local height map as terrain observation to walk and run on rough terrains (Figure 2.3c).

Evolutionary algorithms can be a suitable alternative to RL, even to learn locomotion skills from scratch [Sal+17]. While both approaches present their pros and cons [Maj+21], they also present similarities as they both learn from interactions with the environment. Covariance matrix adaptation is one evolution strategy that can be used to learn walking by controlling torques [YLP07; WFH09] or musculo-tendon units on humanoid characters [Wan+12] and various biped creatures [GPS13].

The recent breakthroughs in RL algorithms such as PPO [Sch+17] and others [FHM18; Haa+18a] have permitted the learning of highly adaptive whole-body controllers on rough terrains for legged character locomotion in simulation. These methods are promising to compensate for the weaknesses of model-based approaches, which can be difficult to develop and demonstrate fewer generalization capabilities. However, these controllers are often long to train as they require millions of interactions with the environment. Moreover, the motions generated are usually jerky visually, and the search for a more stable bipedal walk with RL remains [Par+20].



(a) Bloesch et al. [Blo+22] (b) Haarnoja et al. [Haa+18b]

Figure 2.4: Robots learning how to walk in the real world.

Robot learning and sim-to-real. Several works using RL demonstrate how to learn to walk on legged characters in simulation. However as stated in the survey of Ibarz et al. [Iba+21], a policy learned in simulation usually performs badly on the real robot due to the discrepancy between the simulation and the real world.

Diverse sim-to-real methods have appeared to bridge the reality gap such as randomizing the simulation dynamics [Pen+17a], identifying the dynamics parameters on the real robot [Yu+19] or developing an accurate actuator model and simulating latency [Tan+18]. While those methods alleviate the reality gap problem, they do not compensate totally for the environment model inaccuracy in simulation. That is why such trained controllers could require additional fine-tuning on the real robot.

Despite some recent success, the sim-to-real transfer is mostly a matter of robustness, that we can now improve at the cost of longer training and less optimality. One of the stakes in reinforcement learning is to rather adapt the learning to the real robot dynamics.

Robot learning in the real world. Learning controllers by reinforcement directly on real robots is a appealing direction to remove the dependency on imperfect simulation [KBP13]. However, such an approach is challenging due to critical real-world limitations.

The sampling collection is a tedious process on the real robot, and so the policy has to be learned from a significantly less amount of data. As a consequence, algorithms with better sample efficiency are required [Haa+18b; Cha+18].

Reinforcement learning algorithms are trial-and-error processes, thus leading to numerous failures during learning. In the real world, these failures translate into a high risk of breaking the robot and a potential threat to the safety of its surroundings. As a result, learning whole body controllers "in the wild" [Blo+22], meaning directly on the real robot, requires additional safety measures. Most works on the topic avoid such an issue by learning on relatively small-sized and harmless robots [Nav+12; Blo+22]. Back in 2005, Tedrake et al. learned how to walk by reinforcement directly on a low degree of freedom biped robot [TZS+05]. Later on, as the RL algorithms and robot designs improved, learning locomotion from scratch has been performed on quadruped [Haa+18b] (Figure 2.4b), hexapod [Haf+20] and more complex biped robot [Blo+22] (Figure 2.4a). Another limitation of learning in the real world is that during the training (up to several hours or days), the human will have to manually reset the robot position as it falls over, bumps into a wall, or reaches the edge of the terrain. Yang et al. [Yan+22] alleviate this limitation by lowering the number of falls. They propose a safe recovery policy to take over the control when the learning agent violates some safety constraints, thus decreasing the need for human intervention while improving the safety of the robot. Finally, some strategies can use knowledge from a simulation to quickly learn how to adapt to the real world. With a small hexapod robot, Mouret et al. [CCM14] learn in simulation a behavior map using evolutionary algorithms, then composed of thousands of walking behaviors. At runtime on the real robot, this map can be

efficiently searched using optimization algorithms. Their results demonstrate that the robot can learn to select the best walking behavior among this map in a few trials, to compensate for its body damage and to adapt to its environment.

Applying RL in the wild is an exciting research direction. But it is for now limited in the context of locomotion to small robots that require constant human intervention during training, and which has only been applied to flat ground so far. Designing bigger-sized robust robots or strategies to make the robot automatically recover [Sch+10] are promising directions.

Conclusion on contact agnostic approaches. The recent surge in the use of reinforcement learning has proven promising to develop controllers with a contact agnostic approach. Works on this topic demonstrated impressive results to generate robust walking motion, while adapting to the rough terrains in simulation [WGH22] and in the real world [Yan+22].

Nowadays, learning robot locomotion (and other skills) in the wild is yet limited to small robots for safety and cost reasons. Learning from simulation remains the best way to ensure the safety of the robot and its surroundings. It also permits to easily reset the robot state, and bypass the poor data efficiency of learning on real systems [Li+18] (i.e. tedious collection process of data). However, learning safe and robust walking on biped robots without specifying a reference motion or contacts is still difficult. Finally, the search for more accurate simulations or efficient sim-to-real methods remains.

2.1.4 Conclusion

In summary, we can classify the existing locomotion controllers in 3 categories, whether they cannot decide the contact variables, or explicitly optimize them as continuous or partly discrete values, or agnostically decide the contacts as a consequence of the movements.

Implicitly optimizing the contact position along with the motion is mostly limited to flat ground scenarios. While recent whole-body controllers can demonstrate good generalization to uneven terrains, they are still inherently limited due to their blind nature. Simultaneous motion and contact optimization are promising to generate dynamic and safe locomotion. However, they remain expensive to compute, thus limiting their use for online planning.

Research in locomotion has seen a lot of interest in contact agnostic approaches using reinforcement learning. However, these methods are not mature yet to be feasible on a real human-sized robot.

For now, all these methods fail to properly generalize to some kind of complexity in the locomotion scenarios (e.g. handrail grasping), to the more unstable bipeds, or to efficient robot deployment. In general, the best strategy in all these cases is to rely on an external algorithm to decide or guide the contact decision [Car+16; Léz+; Dan+22].

Now we are left with one question: *How to obtain these contacts ?* Manually defining them is a solution for fixed scenarios, but to obtain a more general locomotion planner we need to investigate the so-called *contact planning problem*.

2.2 The Contact Planning Problem

A contact planner finds a discrete sequence of contact positions that the robot has to perform to go through the terrain. It is expected that the produced contact sequence does not come with a complete whole body trajectory, although some of its elements may be computed as well depending on the underlying algorithm.

As stated by Chestnutt et al. [Che+09], planning footsteps rather than whole body motion allows the robot to reason about contact with the environment to ensure safe and stable support.

This also reduces the planning state space to a dimensionality computationally tractable for online walking. From a sequence of predefined contacts, efficient whole body controllers exist for motion planning [Car+16; Car+18; Léz+; Dan+22]. The division of whole-body control and contact planning greatly lowers the complexity of the locomotion problem. Indeed, it relies on simplifications and assumptions on the robot dynamics to efficiently compute a feasible sequence of contacts. However, it also comes at the cost of the non-guarantee of the feasibility of the contact plans.

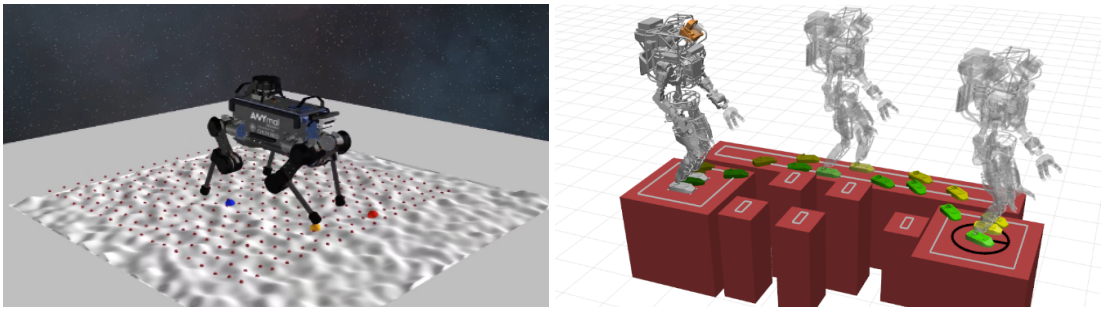
The question of feasibility is of the main importance here, as we will show in this section: given a contact sequence in a given environment, can we guarantee that there exists a corresponding whole-body trajectory that follows it? This feasibility question is irrelevant for the previously presented methods which compute motion and contacts simultaneously [Win+17; Ace+19].

In this section, we classify contact planners of the literature into two categories as described by Bretl et al. [Bre06]:

- *contact-before-motion*, that decides the future contact placements, before planning the whole body motion.
- *motion-before-contact*, that further divides the contact planning into two sub-problems. First plan a rough trajectory the robot root has to follow, which we call the *guide path*, then compute the contact placement along it, and finally the whole-body motion.

We will explore the results of both strategies, along with their pros and cons.

2.2.1 Contact-before-motion



(a) Gangapurwala et al. [Gan+22] (b) Deits et al. [DT14]

Figure 2.5: Contact-before-motion strategy with (a) short horizon and (b) long horizon planning.

Planning contact placements prior to the whole-body motion raises a key problem: *how to know if a sequence of contacts is feasible by the robot?* A naive positive proof would be to exhibit the corresponding whole body trajectory. However, we would need a cheaper decision test, which ideally should also prove a negative answer. Seminal works solve this question using simplified models of the robot dynamics in its environment. Their related concepts such as the zero momentum point [Kaj+03] or contact wrench cones [BT02] can be used to characterize the stability of contact configurations. Assuming quasi-static locomotion is also a classic assumption that simplifies the contact planning problem with approximated stability criteria. With this approach, we can plan robust contact placements that the robot can follow while staying in static equilibrium during the single and double support phases [DTM16]. However, a quasi-static approach also limits the contact planners as they consider slow enough and in constant equilibrium motions. Recently, computing the dynamic transition feasibility to a next contact has been made efficiently [Fer+20] and allows for more dynamic locomotion.

Finally, we need to answer the question: *where to place these contacts in the environment?* For that, we will explore contact planners operating at two different scales, short-horizon and long-horizon planning.

Short horizon planning. This approach focuses on computing the immediate next contact (or few contacts) to be performed by the robot, following a given direction.

As one could guess, this approach is particularly fast to compute as it is often based on simple heuristics for foot placement while respecting equilibrium and environment constraints [Rai86]. Using this strategy, Scianca et al. [Sci+19] compute the next footstep to be made on flat ground, then adapt and perform it by their whole-body controller on the humanoid robots NAO and HRP-4. Rebulla et al. [Reb+07] present a model-based controller for quadruped robots planning only the next footstep to statically walk on rough terrains. Recently in [Bra+22], a fast acyclic contact planner is learned by a multi-output regression neural network. Their network ranks discretized stepping regions and learns to select the best next feasible footstep.

Reinforcement learning has also been used to choose the next few contact positions to be performed by the robot [Tso+20; Gan+22] (Figure 2.5a). However, they often require complex reward design, long learning time (tens of hours), and the learned policies are specific to each robot model.

Short horizon contact planners are fast to compute (a few milliseconds per step) but are by definition not designed for global planning, as they cannot guarantee its completeness to reach a distant goal. Indeed, they are likely to navigate to insurmountable obstacles in complex environments, thus leading to local minima. However, those characteristics make them particularly suited for robot locomotion with joypad-like user guidance to avoid unfeasible scenarios.

Long horizon planning. This approach solves a global contact planning problem in order to reach a distant goal. Of course, as, we have no means to know in advance the number of required steps for that, it often leads to long discrete sequences of decisions with potential underlying combinatorics. As a consequence, contact planning on a long horizon is a combinatorial problem that takes longer to solve, but that should offer guarantees of optimality and completeness (i.e. it must find a solution if one exists) given enough computation time. We identify two approaches in the literature to solve this problem: discrete and continuous.

Discretization of the terrain, in position and orientation, can be used to obtain a graph of robot footstep candidates, then searched by an algorithm (typically A*). Using a user-defined contact set is a possibility to build a graph, such as Kumagai et al. [Kum+20] that then plan multi-contact locomotion on a real humanoid robot. However, automatizing this processs is required to generalize to more environments. One strategy is to uniformly discretize the terrain into a grid, describing where the foot contacts and transitions could potentially be made [Abd12]. Following this strategy, some works [NCK12; Gri+19a] define each grid node as a potential contact placement, and rely on an A* algorithm to plan a sequence of footsteps on complex terrains. A disadvantage of such uniform discretization is that many possible solutions could be ignored depending on the resolution of the grid. While a high resolution solves this problem, it also irremediably leads to a curse of dimensionality, thus requiring more efficient graph-search algorithms [Ver+09; Zuc10; Hor+12; CS19]. Probabilistic sampling-based algorithms are an alternative to uniformly discretized search space. In particular, Probabilistic Road Map (PRM) [Kav+96] and Rapidly-exploring Random Tree (RRT) [LaV98] algorithms can be used to plan contacts for climbing robots [Bre06] and humanoid robot locomotion [Hau+08; Per+12; Fer21]. The potential field is another sampling method that can be used for contact planning, such as Escande et al. [EKM06] that use it to incrementally build a contact tree up to a goal. The sampling efficiency and feasibility of transition can also be improved using precomputed footstep displacements [Che+03; Bau+11], however limiting the number of possibilities.

Continuous approaches for contact planning deal with the *discrete* problem of selecting the contact surfaces and the *continuous* problem of finding footstep placement on these surfaces [Son+20]. Deits et al. [DT14] decompose the terrain into convex regions of potential contact surfaces, then formulate the problem as a Mixed Integer Programming problem (MIP) with discrete variables to decide on which convex region to step on, and continuous variables for footstep

positions and orientations. The resulting contact planner can find contact plans up to a goal on complex scenes composed of tens of surfaces (Figure 2.5b). However, it requires predefined convex contact surfaces and still presents high computation time due to the formulation (10-30steps computed in up to a few minutes depending on the terrain). Using a relaxation of the MIP, Tonneau et al. [Ton+20] reformulate a feasibility problem and present a much faster computation time than MIP when planning a smaller number of steps. As discussed in Section 2.1.2, this work also connects with MIP-based locomotion optimization, where the continuous motion and the discrete contacts are optimized together. As it reasons with a simplified model inspired from contact planning, it also introduce the notion of feasibility, although with complete guarantee. Continuous approaches for contact planning, depending on the formulation of the problem, are promising to alleviate the limitation of discrete approaches using graphs. However, their formulation requires predefined convex regions for contact surfaces and can still be computationally expensive (or even fail) in the presence of highly combinatorial problems. These approaches will be covered in-depth and used later on in this thesis.

Conclusion on contact-before-motion. Short horizon footstep planning solves a local problem to move in a given direction. It is fast to compute, and thus pertinent for real-time replanning. However, it can be stuck in local minima on complex scenes.

Long-horizon planning solves a global planning problem up to a distant target. This approach presents higher computation times in function of the terrain complexity and the desired number of footsteps (e.g. many choices of potential contact surfaces), but can offer guarantees of completeness or optimality if given enough time.

2.2.2 Motion-before-contact

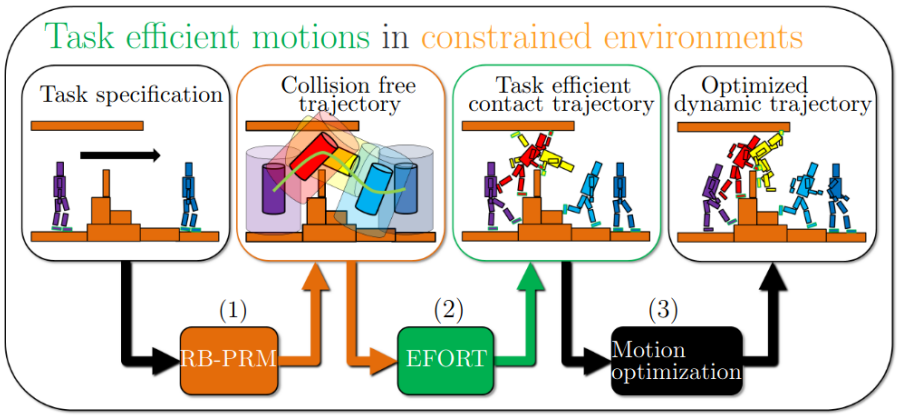


Figure 2.6: Motion-before-contact strategy, planning a rough robot trajectory to guide the contact planning. Source: Tonneau [Ton15].

Planning only a few steps ahead requires the careful guidance of the robot to avoid falling into local minima. On the other hand, planning several steps is subject to combinatorics, making the problem exponentially expensive to compute.

Inspired by some previous works on character animation [Kuf99; PLS03], the motion-before-contact strategy alleviates these limitations by adding a higher-level planning layer, thus further dividing the contact planning problem into two modules (See (1) and (2) in Figure 2.6):

1. A guide path planner, to generate a rough trajectory the robot has to follow to go through the environment (e.g. a robot base collision-free trajectory).

2. A contact planner, to compute the contacts along this trajectory.

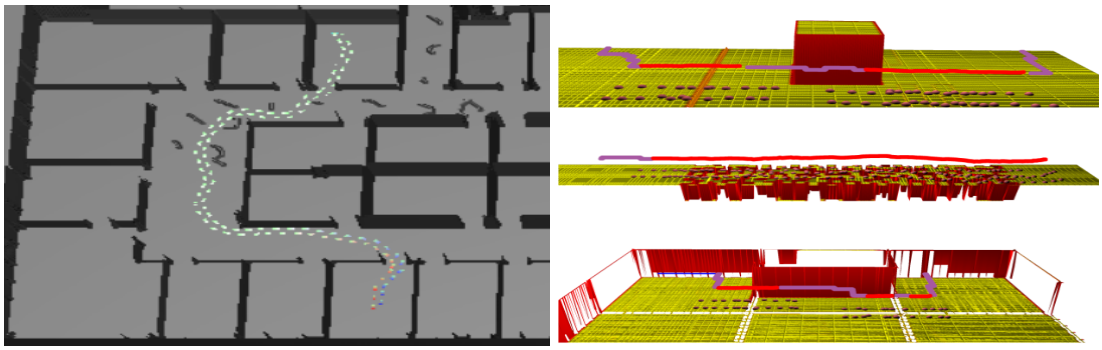
Previously presented contact planners can thus be adapted to use a guide path to constrain the search for feasible contacts in the environment, and to obtain additional information about the path to traverse.

Guiding a contact planner. Planning a robot trajectory is a problem of lower complexity that can be used for both horizons of contact planning. Using a guide to control the input direction of short horizon footstep planners, can potentially solve their local minima issue [Nor+22]. On long horizon planners, the guide can constrain the search for footsteps around the guide, thus alleviating the combinatorics of the problem [Hil+17].

Using this architecture, Chestnutt et al. first manually define a graph searched for a 2D collision-free path [CK04; Che07] (Figure 2.7a) or interactively drawing it on a user interface [Che+09]. This path then guides an A*-based contact planner. Similarly, Yoshida et al. [Yos+05] introduce a guide planner using a bounding box to model the robot moving while manipulating a large object. Their method generates collision-free trajectories to go through the environment, then followed by a pattern generator.

These works have been extended to more complex terrains with a reachability condition to plan 3D guide paths, before computing a sequence of contacts along it [Ton+15; Ton+18a; WH21]. Using this condition, guide paths can also help in solving the surface selection problem of continuous contact planning methods [Son+20]. In a model predictive control fashion, Risbourg et al. [Ris+22] use a guide to obtain candidate contact surfaces and allow real-time short horizon footstep planning with a MIP method on the quadruped Solo.

Motion-before-contact approaches have proven successful in efficiently planning contacts for legged character locomotion [EV10; Bou+09a; Bou+18]. However, it also presents some limitations as explained in [KE08], where the guide should be rough enough to be quickly planned while being constrained enough to generate feasible contacts along.



(a) Chestnutt et al. [CK04] (b) Brandao et al. [BFH19]

Figure 2.7: Examples of work on motion-before-contact planning.

Estimating the traversability. Planning a path according to an estimation of the difficulty to traverse the terrain can generate guide paths more likely to be feasible by contact planners. This strategy has been used to move some quadruped robots through difficult rough environments [KRN08; Kal+11; Win+14; Win+15; Wer+16].

Additionally, the guide paths can give information about the sections of the terrain traversed, permitting different walking strategies in function of its difficulty, also known as the multi-modal planning problem. To move a humanoid robot in cluttered environments, Lin et al. [LB18] plan a torso guide path using an A* searching algorithm on a cost map, and accounting for the difficulty of traversing the environment. They then decompose the guide path into segments, selecting for each one a locomotion mode. They switch between simple biped walking on flat ground or multi-contact locomotion using the robot's hand for increased stability in cluttered environments. In

the same line of work, Brandao et al. [BFH19] use a quadruped robot body path to plan the different modes, such as a walking or trotting gait on flat ground, and contact planning on terrains requiring careful footstep placements (Figure 2.7b). Their results demonstrate that multi-modal planning locomotion can potentially achieve faster computation speed than pure footstep planning methods.

Conclusion on motion-before-contact. The motion-before-contact strategy is promising toward faster contact planning for legged robot locomotion. A prior guide path can be used in different ways by contact planners. Some works use it to accelerate the search for footsteps around the guide path, while others use it to get information about the traversed terrain to also solve the multi-modal planning problem.

The key aspect of motion-before-contact is the question of feasibility. If the feasibility is properly captured, then we can hope to quickly plan good guide paths before selecting contacts. Otherwise, the trade-off between planning infeasible paths or missing valid sequences will certainly lead to an over-conservative planner, with strong practical limitations.

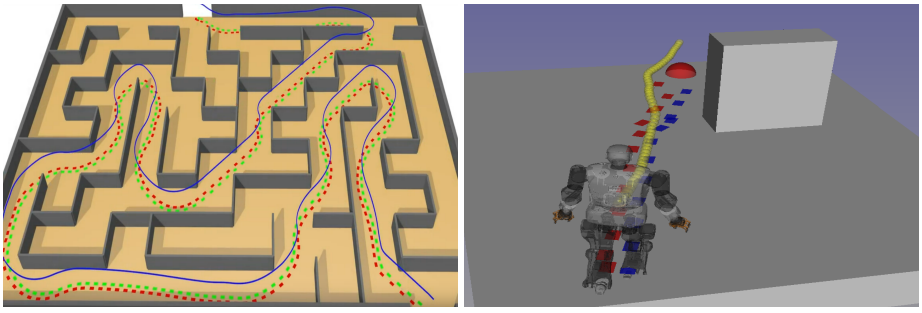
So far, properly describing the feasibility with generic models has been the main bottleneck to these approaches, as we will see in Section 2.3.2.

2.2.3 Conclusion

Contact planning is yet a needed stage to generate, in tractable time, complex movement on legged robots. This task relies on specific model reduction, to ensure footsteps feasibility, that is combined with dedicated algorithms to tackle the discrete nature of finding contact sequences.

In this section, we categorized contact planners into two categories. The *contact-before-motion* approach can lead to local minima on short horizon planning, and become computationally intractable on a long horizon. That is why our research will focus on *motion-before-contact*, that is promising to alleviate these limitations. However, it raises the problem of the non-guaranteed feasibility of the guide path by a given contact planner. The motion-before-contact approach in turn implies guide planning, that can be separately handled as a legged navigation task.

2.3 Navigation Task



(a) Global planner (b) Local planner

Figure 2.8: Navigation task at two different scales. Sources: (a) Tonneau et al. [Ton+20], and (b) a simulation in HPP [Mir+16].

Navigation can be formulated as the problem of planning and following a conflict-free path from some initial to goal positions, where conflict-free refers to the satisfaction of validity criteria (e.g. collision-free). This challenging task has been extensively studied in the context of legged

system or various autonomous vehicle navigation. In particular, the environments they move in are often partially known or dynamic, thus requiring fast planning while satisfying optimality criteria.

To solve this problem, the navigation task can be performed with different architectures operating at different scales [Nak+11]. Recent state-of-the-art implementations for autonomous vehicles, like the ROS navigation stack for 2D navigation [Zhe17], or for crowd simulation [TP21] employ an architecture composed of two modules:

- **Global planner**, that plans a rough feasible path through the environment to a goal (Figure 2.8a). This is usually performed by *path planning* algorithms that search for sub-goals to reach sequentially with a local planner.

- **Local planner**, that follows the rough path under specified rules. These rules can for example be based on the current sensory information of the robot for collision avoidance (Figure 2.8b). This is usually performed by *motion planning* (or *trajectory planning*) algorithms that consider the robot or vehicle dynamics. Here, we will call such a local planner a *steering method*, which is a widely used term in navigation.

The contribution specific to legged navigation is mostly in the definition of the steering method, whose main challenge is to capture the feasibility of the robot whole-body, hence leading to acceptable robust or even natural movements. Yet, the properties that one must aim for when designing a steering method cannot be understood without, first, the algorithms underlying the navigation problem. While we expect most of our readers to be already familiar with these algorithms, we decided to dedicate the next Section 2.3.1 to a tutorial about them. Expert readers can skip it. Then we explore the existing contributions specifically addressing the legged navigation problem in Section 2.3.2. This will enable us to finally give my thesis contribution as a conclusion to this chapter.

2.3.1 Overview: Navigation Methods

We propose to explore the algorithms used in the broad context of navigation. These algorithms can be categorized depending on the scale they operate in.

Global navigation in known environments.

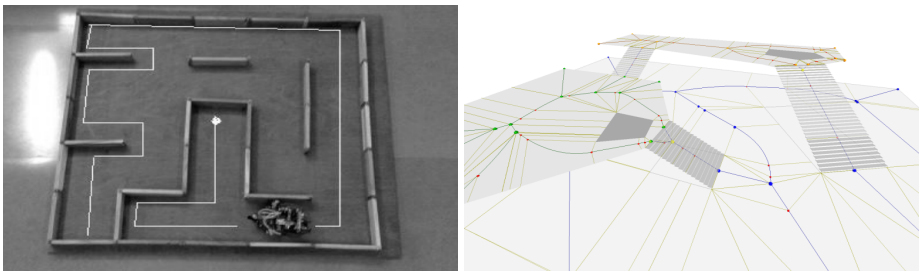
Consider a complex environment in which the robot has to move from an initial to a goal configuration. By knowing the full model of the terrain in 2D or 3D, various strategies have emerged to solve the path planning problem.

We will explore the two main strategies for path planning that are *graph-based* and *sampling-based*. These methods were previously introduced to solve the discrete contact planning problem (Section 2.2.1). Here we will present these path planning algorithms for a wide range of navigation applications. Due to the global scale they operate on, most path planning algorithms do not consider the kinetics and kinodynamics of the system (e.g. maximum acceleration, velocity, and steering angle). Indeed it would result in burdensome computation time. That is why these constraints are often considered by the local planners that will be presented in the next section.

Graph-based searching. A graph is a structure composed of nodes and links, that are the discretized robot or character positions and their connections to other nodes respectively. This graph is then processed by a search algorithm that solves a combinatorial problem to find the shortest path from the initial to the goal configurations. These search algorithms have a long history in the path planning field with Depth-First Search, Breadth-First and the well-known Dijkstra algorithm [Dij59]. Numerous search algorithms emerged to more efficiently explore such graphs, among which A* [HNR68], biasing the exploration toward the objective, along with its variants

D* [Ste94], for an efficient replanning on unknown or partially known environments, or Anytime Repairing A* (ARA*) [LGT03] finding a suboptimal solution quickly then optimized toward optimality.

Now that we know efficient algorithms exist to search a graph, we are left with the question: *How to build such a graph?* Manually specifying each node and link (waypoint graph) is a possibility to ensure the construction of a graph with conflict-free links [Lid02].However, more generic methods requiring less cumbersome manual labor are preferred, such as Voronoi diagrams [BG07] or visibility graphs [HC04].



(a) Mishra et al. [MB08] (b) Toll et al. [TCG11]

Figure 2.9: Examples of global navigation in 2D and 3D: (a) Micromouse challenge, (b) Navigation Meshes

Grid discretization. Grids are used to uniformly decompose the environment into cells. Each cell corresponds to a node in the graph that is linked to its adjacent neighbors in free space (i.e. not occupied by any obstacle), thus forming a graph covering the full environment on which we use a search algorithm. ROS navigation stack [Zhe17] models the environment as a grid, which is then used for global path planning on mobile robots. The grid can represent an occupancy map indicating the presence of obstacles, or a cost map indicating the difficulty to traverse each cell. Several searching algorithms have been introduced for navigation in grid environments, mostly developing variants of A* algorithms to improve the computation time and optimality of the paths [PS05; Duc+14]. In the Micromouse challenge, where a mouse robot has to explore a maze and then search an optimal path up to the goal (Figure 2.9a), modeling the maze as a grid and planning the shortest path with a flood fill algorithms has been one of the most successful strategy [MB08; BM18].

Representing the environment as a grid is a simple yet efficient strategy to compute global 2D paths for various vehicles [FL08; 18; SRR20]. In the same manner, grid decomposition can be extended to 3D using voxels [CFS06; Per+17]. However, it raises the dimension of the problem, which can lead to intractable complexity.

Navigation meshes. These methods also called NavMesh [Sno00] have long been used in the video game industry [Bre19] and crowd simulation [TCG12] to move some simulated characters. They use the terrain geometry, grids or voxels to decompose it into convex polygons traversable by a character. From the polygons, we can then generate a graph of conflict-free links (Figure 2.9b). Toll et al. [TCG11; Tol+18] compute navigation meshes in the order of milliseconds using their method, then search it to efficiently plan paths for thousands of virtual characters in real-time. However, it requires a full model of the environment as well as an offline preprocessing phase to build the mesh. Navigation meshes can be a computationally efficient solution for interactive global navigation, depending on the model of the scene and the NavMesh method used [Tol+16].

Sampling-based algorithms. Building graphs for navigation in a higher dimension or very large

terrains can become intractable, as the combinatorics exponentially increases with the size of the search space [Hauss]. In these cases, sampling-based strategies such as Probabilistic Road Map (PRM) [Kav+96] and Rapidly-exploring Random Tree (RRT) [LaV98] are particularly efficient to build a graph or a tree respectively.

Sampling-based algorithms have been widely used for navigation, using PRM [SLL01; SC12; MK19] and RRT [ZK18] as well as their variants. The system dynamics can also be considered in the sampling, thus permitting motion planning. However, it increases the problem dimension, and so the number of samples required and the planning complexity. As a result, the efficiency of these algorithms boils down to their sampling strategy, and their local planner capabilities to generate link trajectories [Fer+17; Fra+20].

Conclusion on global navigation task. A large panel of algorithms solves the global navigation problem [Lat91; LaV06; Hauss]. As it can be difficult to find which algorithm suits our application, several surveys exist to guide our choice [Yan+16; ZK18; Pat+19].

Overall, global planners have to demonstrate efficient exploration capabilities of the search space. Due to the scale they operate in, they often require an expensive offline phase to build the graph or searching time. As a consequence, those are not suited for replanning in uncertain environments, where the global perception is rarely realistic and unexcepted events can occur.

That is why, it is preferred to combine these global algorithms with a simple linear interpolation to generate collision-free trajectories between graph nodes at a minimal cost, without considering the system dynamics. The graph is then searched to get a rough path (i.e. waypoints to reach sequentially), that is then followed by a local planner to handle environment uncertainty.

Local Navigation for environment uncertainty

A local planner computes a trajectory between two points generated by a global planner. In the context of navigation, we call such a local planner a *steering method* (Figure 2.10). These local planners have been extensively used in navigation tasks for fast and local replanning. Contrary to global planners, steering methods often do not require a guarantee of completeness or optimality but need to produce local paths under desired criteria, such as fast computation, collision-avoidance behavior, or kinetic and kinodynamic constraints.

Definition of a steering method. The definition of the word *steering* is "the control of a vehicle to make it follow a route or direction". However, we identify in previous works different definitions for the term "steering method".

The first definition states that a steering method "computes an open-loop trajectory that brings a nonholonomic system from an initial state to a goal state without the presence of obstacles" [LJ01]. This definition has been used in other works [KS14; SLL01] specifying that such steering methods exactly connect two states.

A broader definition is a local planner that "follows the rough path while adhering to certain local rules" (e.g. collision-avoidance) [TCG12]. Adherence to these rules can imply some reactive behaviors from observations of the terrain such as the presence of obstacles. This definition is used in navigation [Ond+10; Fra+20] where the characters or vehicles are often steered toward waypoints obtained by global path planning algorithms, but may not reach them exactly. In this thesis, we will use the broader definition that we deem more relevant in the context of legged robot navigation.

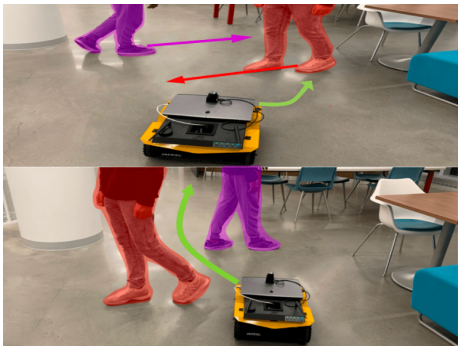
Terrain-free steering methods. Computing a trajectory connecting exactly two points without considering the environment can be done efficiently with model-based approaches.

Interpolation methods consist in computing a curve passing exactly through all the given subgoals [Bou99]. Linear interpolation is the simplest and fastest way to link two configurations in

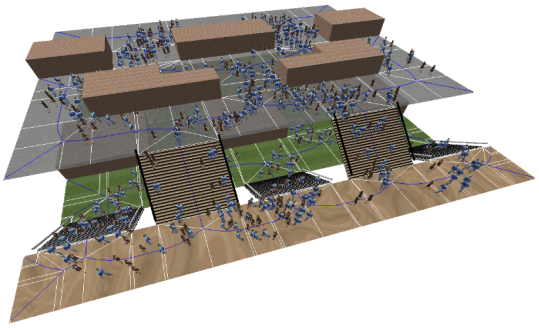
space, making it the default steering method of most graph-based and sampling-based algorithms,

Other interpolation methods such as bezier, spline, and hermite curves exist to obtain smooth and continuous trajectories. Those curves can be a solution to perform motion planning, which regroups different terms as kinodynamic [LJ01; KS14] or nonholonomic planning, that must account for constraints on the system dynamics (i.e. velocity and acceleration) or whose state depends on the path taken respectively.

These steering methods exactly connect two sub-goals and are fast to compute as they ignore obstacles. However, they heavily rely on conflict checking by the global planner when applied to complex environments. This can often be burdensome in algorithms such as PRM and limit their use in applications with environmental uncertainty which requires fast local replanning.



(a) Patel et al. [UM21]



(b) Toll et al. [TCG12]

Figure 2.10: Examples of terrain-aware local navigation: (a) a mobile robot and (b) characters in a crowd simulation locally avoiding collisions with the terrain and other people.

Terrain-aware model-based steering methods. Computing a conflict-free trajectory considering the environment is another steering strategy. Those additional conflict-free constraints permit a local replanning in function of the environment observations, at a higher computation cost compared to terrain-free steering methods.

Local planning can be posed as an optimal control problem [MSE01]. Liu et al. [Liu+17] use a model predictive control to enforce collision avoidance with other vehicles in the context of autonomous driving. Depending on the formulation of the problem and the constraints modeling the environment, these optimization methods can be an efficient solution for locally terrain-aware planning.

Potential field [HA92] is a common technique for path planning using local information about the environment [Ras+17]. This method sees the terrain as an energy potential landscape where obstacles will repulse the robot, while the goal attracts it. While this method could be used for global planning, it is by nature prone to local minima and therefore more suited for local and reactive terrain-aware planning. Potential field has seen some success in crowd simulation with the corridor map method [GO07; Ger10], computing first a global path to follow, then locally planning some collision-free paths according to a potential field.

Some of the most widely used local motion planner for collision-avoidance in vehicle navigation are the Dynamic-Window Approach (DWA) [FBT97] and the Time-elastic-band [Rös+13], both presenting their advantages [Rös; Lec21], and implemented in the ROS navigation stack [Zhe17]. Dynamic-Window Approach (DWA) [FBT97] discretizes the control space of the robot to generate a set of candidate trajectories. The best trajectory is then selected based on a user-defined score (e.g. collision avoidance, clearance to obstacles, distance to goal). Those steps are performed continuously during the navigation and permit a fast replanning. However, this method can be computationally burdensome depending on the discretization of the control space, and the complexity of the conflict checking along the candidate trajectories. Time-elastic-band [Rös+13]

optimizes a robot trajectory generated by a global planner. This trajectory can be seen as an elastic band that will be modified considering some objectives such as minimizing its path length or time, or obstacle avoidance.

The crowd simulation field has also developed methods for local collision-avoidance behaviors between autonomous agents and their environment. One of them is the Optimal Reciprocal Collision Avoidance (ORCA) algorithm [Ber+11] selecting the best action to compute collision-free motions for thousands of characters in real-time (Figure 2.10b). These methods solve a particular problem that is the collision-avoidance in highly dynamic environments, subject to different criteria (e.g. social distances or group behaviors). Those are not applied in the context of this thesis, but we refer the reader to the survey of Toll et al. [TP21] for further details.

Terrain-aware RL steering methods. Reinforcement learning has seen a lot of success in locally terrain-aware navigation due to its perception and generalization capabilities, and robustness to uncertainty.

Controlling an agent in velocity and orientation, RL controllers have been used for steering in crowd simulation [Kwi+22] and autonomous robot navigation [Che+16; Kir+20] in uncertain and dynamic environments. Lee et al. [LWL18] present a method to teach an agent collision avoidance behavior, using rays to detect obstacles and other agents. Francis et al. [Fra+20] learn an RL steering control over linear and angular velocity on an autonomous mobile robot for local navigation. They also use this steering method inside a PRM to evaluate the links feasibility and build a graph. Combining RL with model-based approaches Patel et al. [UM21] generate trajectory candidates using the Dynamic Window Approach (DWA), then learn a policy to select the best candidate (Figure 2.10a). Finally, Muller et al. [MTG17] use an evolutionary algorithm to generate candidate trajectories, thus potentially generating better motion plans than a classical DWA approach.

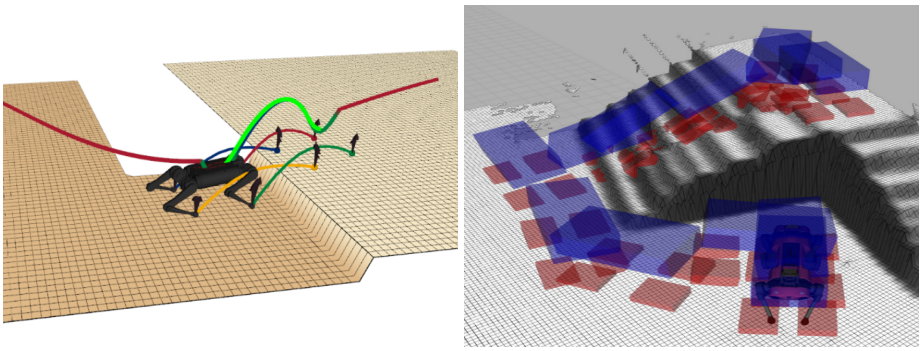
Different representations of the environment can be used to learn local planners such as velocity fields [Haw+20], depth images [Wu+18], lidar sensors information [Fra+20; Chi+19b] or occupancy grids [Alo+20].

Reinforcement learning is a powerful tool to achieve behaviors that are complex to engineer with model-based approaches. In the context of local navigation, those methods tend to adapt well to uncertain or partially known environments. They are a pertinent choice over model-based approaches to learn how to solve complex navigation tasks while respecting some desired criteria.

Conclusion on local navigation. Terrain-free steering methods are fast to compute, but have to rely on expensive global planners for conflict checking and replanning when facing environmental uncertainty.

Terrain-aware methods can remove the need for replanning on simple scenarios by generating conflict-free paths. While model-based approaches can efficiently compute trajectories with collision-avoidance behaviors (e.g. ORCA, DWA), reinforcement learning is particularly promising to learn complex navigation tasks. It is used in the context of autonomous mobile robot navigation to learn policies collision avoidance with humans, or crowd simulation to learn social distancing and adapting to other agents steering behavior. Furthermore, they can generalize well to unexpected events and partially known environments by using rich perceptual observations.

2.3.2 Navigation Task for Legged Robot Locomotion



(a) Norby et al. [Nor+22] (b) Wellhausen et al. [WH21]

Figure 2.11: Legged robot navigation for motion-before-contact planning.

Historically, navigation methods have been used on large near-flat terrains [Che07] or legged robots and avatars [PLS03], paying mostly attention to naturalness [MSW20]. Yet, as soon as the environment is more constrained, legged navigation becomes a dedicated topic, with the central feasibility question: *How can we plan a guide path while ensuring that a corresponding feasible whole-body trajectory exists?* While no definitive answer yet exists, several advances have been made on a part of the problem main difficulties.

Collision-free path. Planning collision-free whole-body trajectories is required for legged robots to locomote through terrains with non-traversable obstacles. Trivially, we have no access to these trajectories during navigation, but we can expect the robot base to be collision-free.

Following [PLS03], previous works typically rely on simplified system walking models for collision detection such as collision boxes. In [Hil+17], the robot is represented as a point with safe margins, simple yet efficient to generate 2D paths avoiding obstacles on flat terrains. As a result, their footstep planner is guaranteed collision-free footstep trajectories, thus demonstrating clear computation gains over their previous contact-before-motion approach. In [WH21; LB18; Ton+18a], the robot torso is approximated by a simple box volume to significantly improve 3D collision checking speed, hence reducing guide planning time.

Robot kinematic reachability. Naturally, the extension of legged robot navigation to 3D also raises the question of reachability, where the robot should be kinematically able to reach the ground with its end effectors. The reachability condition introduced by Tonneau et al. [Ton+15] use an approximation of the robot range of motion, that ensures the robot can reach the terrain all along the path. It has proven successful to generate guide paths on complex terrains with a high success rate for producing a sequence of whole-body posture. However, it turns out to be less successful when trying to connect these postures with feasible dynamic trajectories.

An approach is to incorporate these conditions in sampling-based path planning algorithms [Son+20]. Some works use a PRM with a linear interpolation method [Ton+18a] for navigation and contact planning in tractable computation time, or RRT with model-based kinodynamic steering methods [Fer+17] for dynamic locomotion. In this line of work, [NJ20] use a kinodynamic body path planner to guide their short-horizon contact planner. Their extended results show that their quadruped robot can walk and jump to locomote over complex terrains [Nor+22] (Figure 2.11a).

The reachability condition is the cornerstone to build legged robot contact planners. Yet, it has limited performance as it implies hard-tuned heuristics models that are difficult to generalize, and often fail to capture the connectivity between successive key whole-body postures.

Cost map for terrain traversability. As a refinement to these two required conditions, the fine details of the terrain map can be approximated by a so-called *traversability* score, which evaluates how easy it is for the robot to traverse a region. This technic has been employed in particular during the DARPA challenge to locomote quadruped robots on rough terrains [Tra+22].

In [KRN08; Kal+11; Wer+16], some hand-tuned heuristics are implemented to score footholds, hence evaluating how desirable a foot placement is at a given location. Following this approach, Winkler et al. [Win+14] compute sequentially from a terrain height map: the footholds score map, then the corresponding body score map which evaluates the likelihood of a body position to find desirable and feasible contacts. The body map is then searched to plan guide paths more likely to be feasible by their robot according to their heuristics. However, these heuristics often require complex tuning to sufficiently approximate the robot capabilities, and result in expensive computation to build the map over the search space. In [LB17], this last limitation is alleviated by learning regressors to estimate terrain traversability, faster to querry during planning. Their results demonstrate a higher contact planning success rate with their traversability estimation over a classical collision-free guide, but still at the cost of extra computation.

Combining terrain traversability with the reachability conditions, [WH21] learn a foothold score predictor in a supervised fashion from a few manually labelled data. This predictor enables removing unsuitable geometry from the reachability space. It is then proposed to employ the reachability condition with a PRM and linear interpolation to efficiently plan guides in complex terrains (Figure 2.11b). By doing so, they ensure the presence of suitable surfaces to step on along the guide, hence potentially facilitating the robot locomotion with their blind controller [Lee+20].

Supervised learning seems a promising direction to learn how to approximate guide path feasibility by a contact planner and by extension the whole-body motion. Yet, they mostly rely on heuristics that are either too expensive to compute or can fail to sufficiently capture their capabilities. Learning directly from these planners could solve these limitations, however, the evident lack of dataset makes it difficult to generalize.

2.3.3 Conclusion on Legged Navigation

Legged navigation is strongly related to motion-before-contact planning strategies. Solving it properly then turns the problem of finding a contact sequence to a much simpler one. We follow the argument from Norby et al. [Nor+22] to consider it among the most promising directions to solve the contact planning problem.

A panel of strategies emerged to increase the success rate (or compatibility) of the guide with contact planners. However, they either rely on often necessary but not sufficient conditions (e.g. collision-avoidance, reachability) or on heuristics to improve the likelihood of success that are computationally untractable.

2.4 What are the next steps in artificial locomotion?

Despite some recent promising approaches for legged character locomotion, there exists no locomotion solvers able to generate complex contact platterns without explicitely considering contact decisions on real humand-sized legged robots. It seems evident to us that we must consider the whole problem of deciding contacts and motion simultaneously, both to formulate the theoritical objectives and in the perspective of a promising solution to achieve this task in the future. However, tackling the whole problem in a unique motion solver is yet computationally untractable.

Dividing the locomotion problem in two parts, contact planning and whole-body control, is still required to plan safe robot locomotion in real time on complex terrains. We explored the contact planning problem to automatically generate the contacts to be performed by the whole-body controllers. We have seen contact planning at two different scales. Short-horizon planning

is fast to compute, but prone to local minima. Long-horizon planning can offer some guarantee of completeness and potential optimality at the cost of a higher computation time.

Motion-before-contact strategy is a promising solution to alleviate the limitations at both planning scales. This hierarchical approach contains the following modules:

1. A **guide path planner** to plan a rough path the robot has to follow.
2. A **contact planner** to plan the contacts along this path.
3. A **whole-body controller** to compute the whole-body motion performing these contacts.

We argued that this hierarchichal division offers the most promising directions. Yet, as in any divide-and-conquer approach, it raises the question of consistency in the hierarchy that we formulated as the **feasibility** question: *How to generate feasible guide paths?*

The feasibility has been studied in several research fields. Yet we feel these tentatives lack of systematic approaches. In particular, there is no proposed guarantee either theoritical or empirical that the propose feasibility conditions of one hierarchical level makes the resulting movement a valid guide for the following stage. We propose to explore this problem with a reinforcement learning approach.

The topic of this thesis is the integration of **legged navigation** in the decision flow of locomotion. Particularly, our first proposition is that the **feasibility condition** should be formulated in a way that intrinsically takes into account the limitations of the next task in the hierarchy, the **contact planning**. We formulated our problem as follows: *Can we plan feasible guide paths that are more likely to be extended into valid contact sequences?*

Previous works approach the problem by approximating the contact planner capabilities, such as the **reachability** condition [Ton+15] or other user-designed heuristics. Then, they plan paths considering these approximations. Arguably, they can often be insufficient or too complex to engineer depending on the contact planning strategy [Ton+18a; Win+14]. But any such conditions so far has demonstrated strong practical limitations when scaling it to real scenarios (e.g. nicely producing valid contact sequences but failing to extend to complete dynamic trajectories). Our second proposition is to define dedicated feasibility conditions directly from data collected on past executions of the contact planner. However, there is no evident existing dataset to start learning a feasibility condition, nor any method to compute sufficiently diverse paths and generate such data. We then finally propose to approach the training of such a feasibility condition by *Reinforcement Learning*. We learn the legged navigation task using the contact planner as an external oracle rewarding the navigation system for feasible guidance.

Our work find some inspiration in some recent works on reinforcement learning navigation and animation [Fra+20; Pen+17b]. This thesis further explores these approaches in the context of legged robot locomotion. We answer the question: *How to learn by reinforcement a navigation task for a better contact planning feasibility?*

Chapter 3

Guide path: LEAS

Contents

In this chapter, we present a general approach to learn to steer a locomotion contact planner "LEAS", the core module of this thesis. Our steering method answers the question: *How to locally navigate complex and unknown terrains subject to validity and collision constraints?* LEAS takes as input a desired direction and a local height map of its surrounding terrain, to generate a robot root trajectory. As many different approaches have been tried to solve such a task, we provide an insight into our design choices and particularly why we use Reinforcement Learning.

Here, we focus on the formulation basis of LEAS, which will be reused and adapted for different contact planners through Chapters 4 and 5.

This chapter is organized as follows: In Section 3.1 we describe the context and the challenges we want to solve. In Section 3.2 we present the specification of the problem and our solution LEAS [Che+21]. We describe it as an RL agent and explain our design choices for its observation, control, and reward to achieve the desired navigation behavior. In section 3.3, we present our implementation regarding the feasibility approximations, the random terrain generation, and the learning architecture. In Section 3.4, we evaluate how LEAS can navigate unknown terrains under reachability and collision avoidance constraints, and compare its results to some other model-based steering methods. Finally, we discuss the advantages, limitations, and potential improvements of our local navigation method.

3.1 Motivation

3.1.1 Context

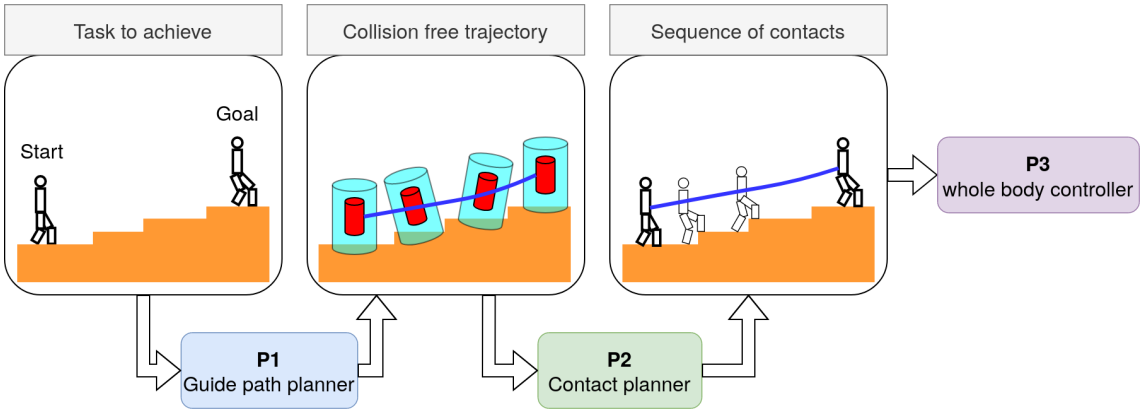


Figure 3.1: Pipeline of the Loco3D framework addressing the locomotion with a multi-stage approach.

As discussed in the previous chapter, most of the recent locomotion planners are commonly structured in several hierarchical stages. Let us now present in detail how we structured our planner, based on the seminal work that drives the locomotion methodology in our team [Car+17]. The general organization is shown in Figure 3.1.

The first stage generates a *guide path* ($P1$), i.e. a desired trajectory of the main robot body, referred to as the *root* (i.e. the basis of the torso for Talos). In the second stage, a contact planner ($P2$) computes the contacts along the guide path. In the last stage, a whole-body controller ($P3$) computes the control sent to the robot to perform these contacts.

The decoupling of $P1$ and $P2$ has proven successful in [EK08; Bou+09b] and later on in [Car+17; Ton+15], where each sub-task can be solved independently from each other, thus reducing the complexity of the problem and allowing us to experiment and compare different module implementations.

The first module $P1$ generates a guide path, defined in this thesis as a discrete sequence of configurations for the robot root in $SE(3)$ (i.e. position and orientation). This navigation module can be further divided into two parts: a Steering Method (SM) and a path planning algorithm. The SM locally navigates the terrain following a given direction, while the path planning uses the SM to sequentially reach sub-goals (also called waypoints) up to a distant objective. On the resulting guide path, all configurations must respect two constraints as expressed in [Ton+15] and shown in figure 3.2.

The first constraint $\mathcal{R}$ is referred to as the *reachability* and imposes that the robot, for a given root configuration, has a non-null contact reachable space (i.e must be able to touch the ground).

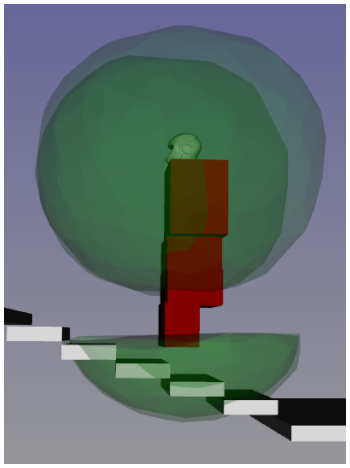


Figure 3.2: Validity constraints of the Talos Robot, (green) ranges of motion of each end effector, and (red) the robot trunk. The configuration above is valid as the robot can potentially reach the ground ($\mathcal{R}$) and the trunk is not in collision with the terrain ($\mathcal{C}$).

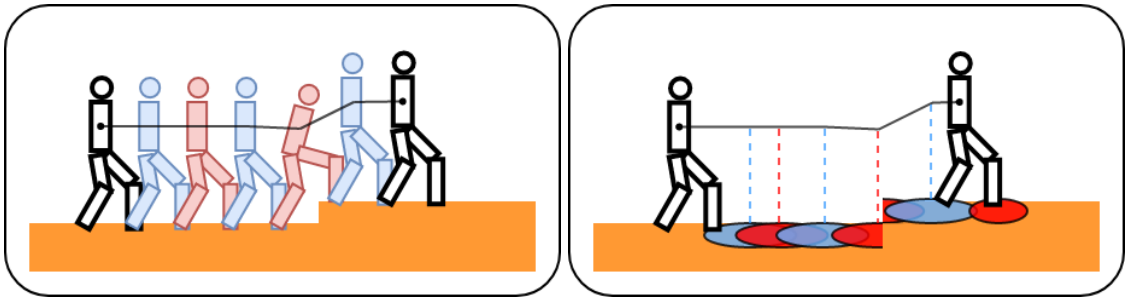
This constraint is approximated as a reachability volume, which is a polytope representing the range of motion of one effector of the robot. As long as an intersection exists between the terrain and the reachability volume for a given configuration, we consider that this constraint is respected. For simplicity's sake, we consider in this thesis only one reachability volume as the union of both legs polytopes. The second constraint $\mathcal{C}$ is referred to as *collision avoidance* and imposes that the robot must not be in contact with the terrain other than with its end effectors (foot and hands). This constraint is approximated as a polytope representing the robot trunk. If an intersection exists between this polytope and the terrain, the configuration is considered in collision. In this manuscript, we will consider a configuration as valid if it respects both constraints $\mathcal{R}$ and $\mathcal{C}$. They both approximate the feasibility of the problem by the next module (the contact planner). We have two Steering Methods at our disposal in [Car+17]:

- RB-Lin, a linear interpolation between two configurations in $SE(3)$. In all our scenarios, we program RB-Lin to first interpolate on the orientation, i.e. to rotate the robot toward the goal with the angular velocity ω_{max}, then to interpolate on the position by moving the robot at $v_{desired}$ each timestep T until reaching the goal;

- RB-Kino [Fer+17], which uses a Double-Integrator Minimum-Time control [KS14], is a kinodynamic SM connecting exactly two configurations in position, velocity, acceleration, and orientation.

The main limitation of these SM is that they do not consider collisions and reachability with the terrain, therefore relying on path planning to validate these constraints along the trajectory. In this work, we will either manually give the sequence of waypoints to reach, or use a reachability-based probabilistic roadmap [Ton+15] as a path planning algorithm with these model-based methods.

The second module $P2$ receives a guide path as input and populates it with a contact sequence. Such a contact planner can have different strategies on how to use the guide. In this thesis, we will explore two strategies for quasi-static contact planning with very distinct problem formulations. The first uses the guide path directly as an exact root trajectory to follow and computes the contacts along it (Figure 3.3a). The second uses the guide to get some candidate surfaces to step on, then solves a surface selection problem to have exactly one surface selected for each contact (Figure 3.3b). The decomposition of the contact planning problem in P1 and P2 (motion-before-contact) breaks the complexity of the contact planning problem, by constraining the search for contacts in the guide path vicinity [Ton+18a; Son+20].



(a) Find key contact configurations along the root trajectory

(b) Solve the surface selection problem for all steps

Figure 3.3: Contact planner strategies, (a) generates key configurations contact following the root trajectory in input, (b) discretizes the guide to find potential surfaces to step on.

Lastly, the module $P3$ performs the contact sequence on the robot with a whole body controller [Car+17; Pre+16; Pen+17b]. For future work, we also implemented an environment to learn such a whole-body controller via deep RL [CLD22]. In this thesis, we will not use the module $P3$ and focus on the contact planning problem (P_1 and P_2).

3.1.2 Problem Statement

In this context of division into two independent sub-tasks $P1$ and $P2$, the price to pay for the simplification of the contact planning problem is the absence of guarantees of feasibility between the different modules. In such a sequential framework, the success of a module is a necessary condition for the success of the next in the pipeline but not sufficient.

That is why, to address this issue, some approximations have been made to improve the feasibility of the problem ($P1$) by the next module ($P2$), such as the reachability condition $\mathcal{R}$. The guide path planner assumes that if the ground is reachable at all times along the path, then it is possible to generate a contact sequence. This strategy has been proven effective in many scenarios [Ton+18a], but can still fail as we will see in the next chapters, weakly approximating the capabilities of the contact planner plugged in. Each contact planner behaves differently as seen in Figure 3.3 and we do not know what makes a feasible guide path for it. As a consequence, it is difficult to define new additional constraints or heuristics in our navigation module ($P1$) to better approximate the contact planning feasibility ($P2$).

We want to fix this issue with a high-level approach, on the very first module of the pipeline $P1$. The problem we are trying to solve can be formulated as the question: *What is a feasible guide path for a given contact planner ?* In other words, how can we build a guide path planner that will better approximate the feasibility space of the contact planner. Such a planner could improve the success rate of $P2$ and sequentially, the success of the whole pipeline.

As explained previously, $P1$ can be decomposed into two components: a path planning algorithm generating some waypoints to reach, and the SM that sequentially connects these waypoints. In this thesis, we focus on the steering method that locally generates a guide path between two waypoints. We want to build an SM able to locally navigate while observing the surrounding terrain to avoid obstacles and validate the reachability condition.

The reason why we do not further investigate path planning in this work is that in most simple scenarios, such as walking on flat ground or climbing some stairs, a steering method should be enough to generate a valid trajectory. Doing so also removes the need for a path planning algorithm (RRT, PRM) expensive to compute.

We build such a terrain-aware steering method with deep Reinforcement Learning (RL). The choice of using RL is motivated by its capabilities to learn a representation of its environment and to act in it, while maximizing the future cumulative reward and keeping a valid state. An

SM learned by reinforcement could learn how to navigate locally with a vision of its surrounding terrain, while respecting the validity constraints ($\mathcal{R}$ and $\mathcal{C}$) and increasing the success rate of the contact planner.

In this chapter, we first train a pure steering method and provide the results. In the next chapters, we use a contact planner as an oracle to validate the trajectories during the learning, then we observe what behavior changes on the SM to increase its success rate.

3.2 LEAS: an RL Steering Method

Given initial and goal configurations, we learn by reinforcement a steering method generating a guide path in a given direction while respecting both reachability and collision constraints ($\mathcal{R}$ and $\mathcal{C}$). To that end, we designed LEAS [Che+21], an RL agent that learns by trial and error how to perform this task and to generalize to unknown environments. An overview of our method is presented in Figure 3.4. The main idea behind LEAS is that for each action taken in the environment, the configuration in $SE(3)$ of the robot will be modified and stored in a list: the guide path.

In this section, we will take a look at the result of LEAS without plugging it to a contact planner. Explanations on how LEAS can improve the success rate of a given contact planner will be left for Chapters 4 and 5.

3.2.1 Specifications

We need to specify the task we want to perform with LEAS, with what behavior, and how to achieve it with Reinforcement Learning.

We choose to input a goal direction in LEAS instead of a goal configuration. We defined a steering method as a planner following a rough path, hence generating a trajectory ending closer to the goal than initially, and that is exactly what LEAS does by moving in the goal direction. This input also allows for more flexible user controls that can set a fixed goal (reachable or not) in the environment or steer in a direction with a joypad.

Following this design, we desire LEAS to have the following list of behaviours:

(A) Move in the goal direction at a fixed desired velocity.

(B) Respect the reachability and collision constraints, $\mathcal{R}$ and $\mathcal{C}$.

(C) Stop if it can not go further (i.e keep a valid state).

(D) Navigate locally without the help of path planning, meaning that LEAS should only act based on a very short visual range.

(E) Orientate its root in the desired goal direction. This design choice can be limiting in cluttered environments where side walking is required, but was necessary for our experiments to obtain a straight walk (sidewalking was a predominant locomotion strategy without this specification, as we will discuss in the next chapters).

The design of an RL agent for our navigation task requires four main components: the agent *state* that represents what the agent sees from the environment, the *actions* it performs according to its actual state, the *reward* that evaluates how well the agent acts to achieve the desired behavior, and finally the *done* condition that verifies if an episode is over.

In our work, the *done* condition is straightforward as it verifies the reachability and collision constraints ($\mathcal{R}$ and $\mathcal{C}$), i.e. the episode is over if the actual robot configuration cannot reach the ground or is in collision with the environment. Consequently, to maximize the future cumulative rewards, the agent learns by reinforcement how to avoid such cases and to fulfill the behavior (B). We can add two optional conditions to stop the episode, depending on the scenario played: when reaching a maximum number of steps, or when being close enough to an objective.

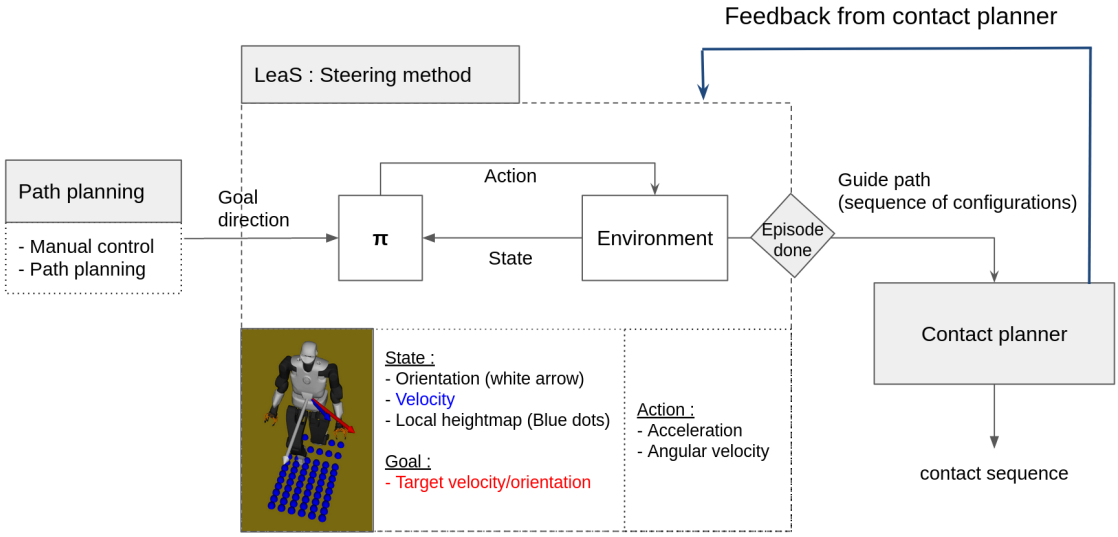


Figure 3.4: Overview of LEAS: the user or a path planning algorithm inputs a target direction to our steering method. LEAS is an RL agent that moves the robot in the environment and sequentially saves all its states. Once the episode is over, this state sequence is the guide path given to the contact planner $P2$. Finally, $P2$ returns an evaluation of the contact plan success that makes LEAS learn how to generate feasible guide paths for it.

3.2.2 States

The robot configuration can be seen in Figure 3.5, from which we can get the observable states driving the actions of LEAS. The desired velocity and orientation are represented by the same vector and expressed implicitly in our states.

The observable state is a set:

$$\mathrm{s} = [v_o, o_{target}, h_o] \tag{3.1}$$

with v_0 the velocity of the robot relative to its orientation, o_{target} the angle between its actual and desired orientations (yaw), and h_o a local height map of the terrain relative to its root configuration.

The dimensions of the height map are 7 values in front of the robot, 3 in the back, and 7 values on each side with a discretization step (rounded) of 15 cm, 17 cm, and 9 cm respectively. This roughly corresponds to a short vision range of 110 cm in the front, 50 cm in the back, and 60 cm on each side of the robot (Figure 3.6a).

The observable height map is small on purpose as we desire a steering method with the behavior (D), to navigate locally. Increasing its visual field leads to an over-fitting on our training terrain and the emergence of path planning behavior. Indeed, if we give too much information about the surrounding terrain (Figure 3.6b), one could memorize its topography, guess its global position on it and plan its path through it, as done in [Alo+20]. Empirically, we found that 110 cm in front and 60 cm on each side of the robot was enough for LEAS to detect the obstacles and react in consequence. The height map resolution is 15 cm, compared to previous works as [Gan+22; Tso+20; Pen+17b] that have a resolution of 2cm, 4cm, and 34cm respectively, and is sufficient to navigate the training ground and to generalize to unknown terrains. Increasing its resolution did not improve the learning or navigation skill of LEAS for our test scenarios, but may be required in the future for more complex scenes with elements of smaller surfaces (handrails detection for example).

All z values in the local height map are bounded between $[z_{min}, z_{max}]$. The tuning of these bounds depends on the topography of the terrain where LEAS will navigate. Indeed, a wider interval means that the RL agents will better dissociate higher height variations (stairs and obstacles) at the cost of a lower resolution for small variations (rubbles and small slopes). This is illustrated in

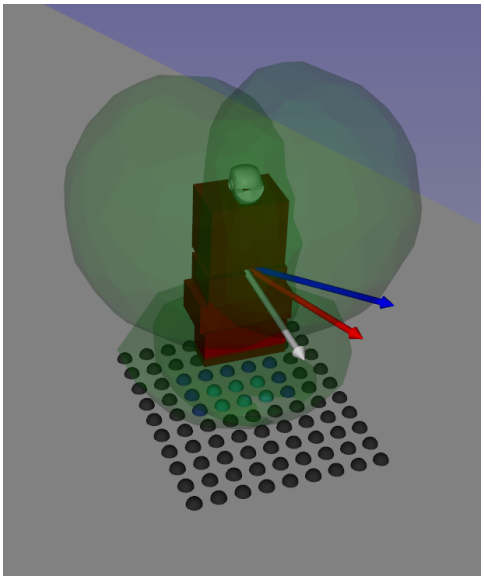


Figure 3.5: Robot states: (blue arrow) velocity of the robot, (white arrow) orientation, (red arrow) desired velocity and orientation, (dots) height map relative to the root configuration.

Figure 3.7 where wider bounds (a) are a better representation for obstacle detection, and smaller bounds (b) make the agent more sensitive to small height variations but may mistake the distant obstacle as a stair. Empirically, we find a right balance $z_{max} = -0.2$ and $z_{min} = -1.4$ meters from the robot root (usually located at 1 meter from the ground), to be sufficient for LEAS to visualize both small and high variations during a pure navigation task.

3.2.3 Actions

At each step, the RL agent takes action in the environment based on the observed state. Our policy returns a set of actions:

$$\mathrm{a} = [a_x, a_y, a_z, \omega] \tag{3.2}$$

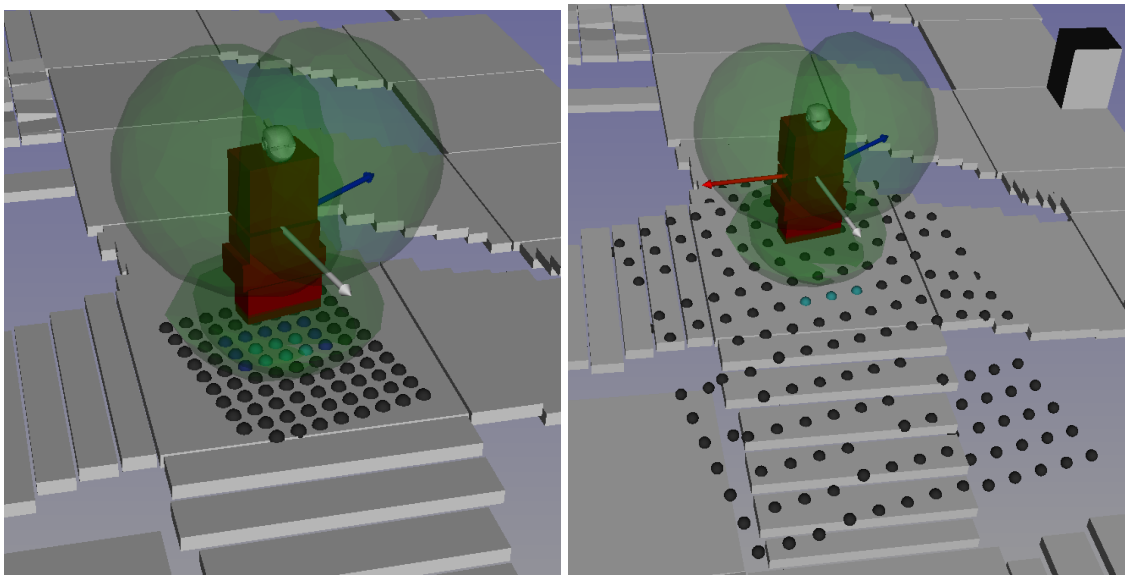
with a_x, a_y, a_z the accelerations of the robot on each axis, and ω the angular velocity of the robot on the yaw axis. At each step i, the robot position q_{pos}, velocity q_{vel} and orientation q_{ori} are modified in the following order:

$$\begin{cases} (a)\ q^i_{pos} = q^{i-1}_{pos} + q^{i-1}_{vel} * T \\ (b)\ q^i_{vel} = q^{i-1}_{vel} + [a_x, a_y, a_z] * T \\ (c)\ q^i_{ori} = q^{i-1}_{ori} + \omega * T \end{cases} \tag{3.3}$$

with T, a user-defined timestep, q_{pos}, q_{vel}, and q_{ori} the global position, velocity, and orientation of the robot respectively.

The velocity q_{vel} and timestep T impact the number of configurations along the guide path. For a constant velocity along the guide of 0.10 m/s and $T = 0.2$ seconds, the discretization step between each configuration is 2 cm. We can further bound q_{vel} in order to keep its norm $|\mathrm{q}_{vel}| \leq v_{max}$ with $v_{max} = 0.2$ m/s.

The choice of T depends on the contact planner used. Empirically, we know that the contact planner [Ton+18a] has a higher success rate for discretization steps inferior to 15 cm. For LEAS, we choose to set $T = 0.2$ seconds which results for $v_{max} = 0.2$ m/s in a maximum discretization step of 4 cm. In the next chapters, we will further prune configurations along the guide to modify this maximum discretization step when required. These parameters have been empirically selected to achieve the specified navigation behaviors, while computing enough configurations along the guide to be feasible by our contact planners (discussed in the next chapters).



(a) (b)

Figure 3.6: Vision field of LEAS for different local height map sizes.

The order in which the robot states are updated induces a delay in the action of the agent. The position of the robot is updated with the previous velocity (1), then the acceleration actions update the velocity (2). As a result, the agent can observe its action impacts directly on the velocity, but not on the robot position and so its observable local height map. One can ask if inverting the order of (1) and (2), hence removing this delay on the position could improve the learning and results of our agent. In practice with recent RL algorithms, fixed delays of one or two steps do not matter [Wal+07] as long as the agent has enough steps left to react to an event. We verify it in Figure 3.13 that shows no difference in the learning of both controls with our parameters.

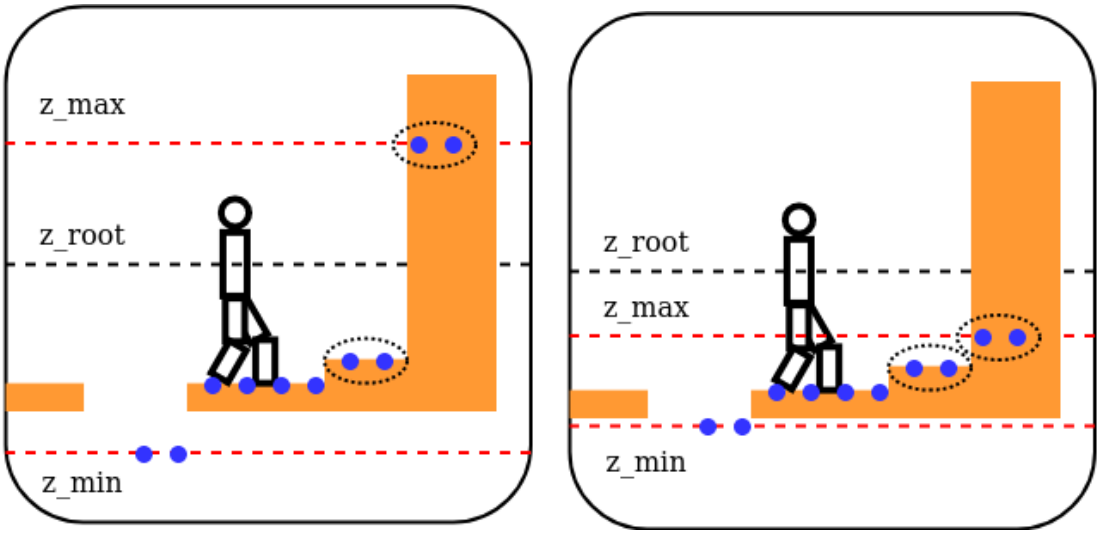
We choose an acceleration control for LEAS, where velocity changes along the trajectory are limited by a_{max}, the maximum acceleration. In our experiments, we find that small accelerations with a small timestep T help the exploration and lead to a more stable training overall, as the actions have a lower impact on the system compared to a velocity control. That is why we set a sufficiently high maximum acceleration $a_{max} = 0.08$ m/s^2 to react to new observations, as the detection of an obstacle, but low enough to help its learning and generate smooth trajectories.

Another option is position control, where the actions directly correspond to the next root position q_{pos}, or similarly velocity control with a high timestep T, to exactly compute one configuration per footstep along the guide. While this strategy is pertinent in the contact planning context, it is also more difficult to train as it requires a few numbers but critical actions to generate a guide path. In our experiment, training LEAS with such position control is inefficient as it drastically lowers the probability to end in a feasible state, hence requiring additional strategies to guide the exploration. As a consequence, we prefer an acceleration control leading to stable learning of our navigation task.

3.2.4 Rewards

As written in the specifications (section 3.2.1), the behaviors we have yet to obtain are: (A) move the robot in the goal direction at the desired velocity, (C) stop if it can not go further, (E) orientate its root in the goal direction. We design three rewards to get these behaviors:

(E) $R_{ori} = -(1 - \overrightarrow{q_{ori}} \cdot \overrightarrow{u}_{target})$
with $\overrightarrow{q_{ori}}$ the unit vector representing the root orientation and $\overrightarrow{u}_{target}$ the goal direction.



(a) Wide height map bounds (b) Small height map bounds

Figure 3.7: Observable height map with z_{min} and z_{max} relative to z_{root} and (blue dot) the bounded height map values. If these bounds are (a) too large, the agent can easily detect the distant obstacle but may not dissociate the ground and the stair in front of him, or (b) too tight, the agent can better detect it but mistake the obstacle as a second stair.

(A) $R_{dir} = -(||\overrightarrow{q_{vel}} - \overrightarrow{v*}||/(2v_{max}))^2$
with $\overrightarrow{q_{vel}}$ the root velocity vector, $\overrightarrow{v*} = \overrightarrow{u}_{target} \times v_{desired}$ the desired velocity vector and v_{max} the maximum velocity norm.

(C) $R_{alive} = 1$

The reward R_{ori} penalizes the agent for not being oriented toward the goal (E) and R_{dir} penalizes it for not moving in the desired direction at $v_{desired}$ (A). As both rewards are negative, using only these two encourages the agent to terminate the episode as soon as possible to avoid accumulating negative rewards. Therefore, we introduce another positive constant reward R_{alive} that the agent gets at each step to encourage him to keep a valid configuration and continue the episode. As a result, the agent learns that to maximize the future rewards, if it is not possible to move in the desired direction, it is better to keep the robot idle rather than terminate the episode, thus fulfilling the behavior (C).

To further smooth the trajectory, we add two rewards to punish the agent for taking large actions:

1. $R_{\omega} = -|\omega/\omega_{max}|^2$
with ω the action on orientation and ω_{max} the maximum angular velocity.

2. $R_{acc} = -(|||[a_x, a_y, a_z]|||/a_{max})^2$
with $[a_x, a_y, a_z]$ the acceleration actions and a_{max} the maximum acceleration norm.

The resulting reward is : $R = R_{dir}w_{dir} + R_{ori}w_{ori} + R_{\omega}w_{\omega} + R_{acc}w_{acc} + R_{alive}w_{alive}$
with w_{dir}, w_{ori}, w_{ω}, w_{acc} and w_{alive} some user-specified weights.

Another possible reward design is to let R_{dir} be positive and remove R_{alive} as done for the High-level controller of DeepLoco [Pen+17b]. However, we separate these two rewards for ease of use. Indeed, an advantage of this separation is that we can increase w_{alive}, relative to the other reward weights, to further encourage the agent to act carefully and stay away from dangerous situations like staying too close to an obstacle. In our experiments, keeping $w_{alive} = 1$ is sufficient

to get the desired behavior (C), but setting it too high as $w_{alive} = 2$ can make the agent act too safely and stay idle instead of crossing difficult obstacles.

3.3 Implementation Details

We train LEAS using HPP software [Mir+16] to load the terrain and the Talos robot model [Sta+]. To speed up the training process and learn a policy with good generalization capabilities, it is desirable to have several agents in parallel during the training to generate trajectories on one or several diverse scenes. While it is possible to have several clients of HPP on the same machine, this software was mainly implemented for short scenarios on small terrains, but not for intensive usage as we do in RL or to load scenes with thousands of surfaces.

To that end, we implemented a python library to extract a height map from the scene meshes, and we present two main tools to be implemented in HPP later on: an approximation of the collision and reachability constraints using the height map, and a random terrain generator. Finally, we present an asynchronous version of the RL algorithm Proximal Policy Optimization (PPO) [Sch+17] that will be used in the next chapters to externally compute the contact planning sequence with a master-workers architecture.

3.3.1 Validity Approximation

The validity of a configuration is subject to two constraints: reachability $\mathcal{R}$ and collision $\mathcal{C}$. Efficient functions are available in HPP to verify these constraints by checking if an intersection exists between either the range of motion of the robot legs or the volume representing its trunk, and the terrain surfaces. However, in our first version of LEAS [Che+21], the computation time of these validation functions was significant (more than 15% of the computation time), and as a consequence, we opted for a faster and more flexible implementation of these constraints.

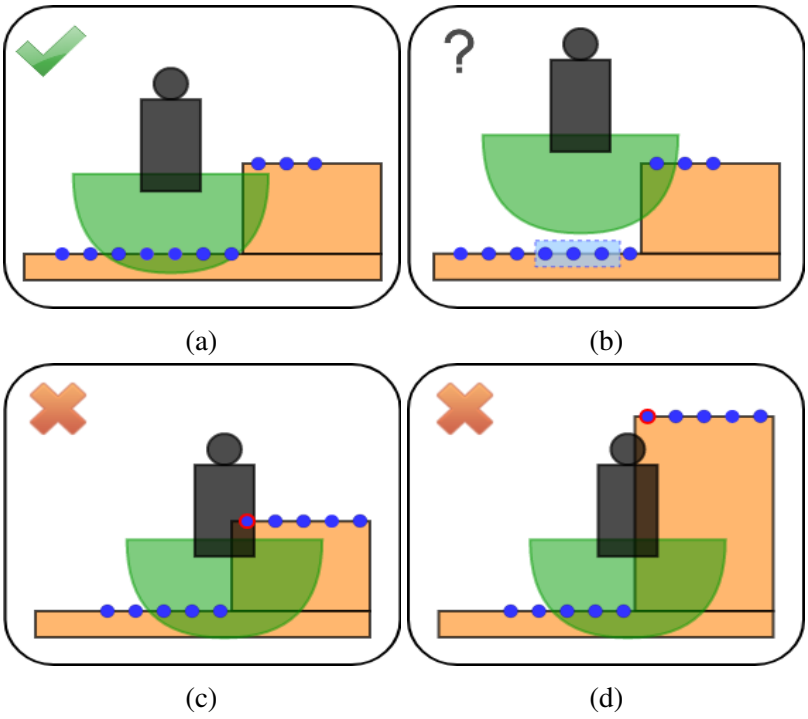


Figure 3.8: Approximated validity conditions: (Green) robot range of motion, (blue) height map. Configuration (a) is valid, (b) is valid with $\tilde{\mathcal{C}}$ and $\tilde{\mathcal{R}}$ but invalid with $\tilde{\mathcal{R}}^*$ where none of the highlighted blue dots are reachable, (c) and (d) are valid with $\tilde{\mathcal{R}}^*$ but in collision.

We approximate the collision and reachability constraints by $\tilde{\mathcal{R}} \subset \mathcal{R}$ and $\tilde{\mathcal{C}} \subset \mathcal{C}$ respectively, both using the local height map H relative to the robot root (Figure 3.8). Offline, we assign to each point $p_i \in H$, some reachability $\tilde{\mathcal{R}}_i$ and collision $\tilde{\mathcal{C}}_i$ constraints on its height z_i. A point p_i respects $\tilde{\mathcal{R}}_i$ if it lies inside the robot range of motion, and respects $\tilde{\mathcal{C}}_i$ if it lies outside and under the robot trunk volume. The approximated reachability condition is valid if:

$$H \in \tilde{\mathcal{R}} \Rightarrow \exists p_i \in H, p_i \in \tilde{\mathcal{R}}_i \tag{3.4}$$

The approximated collision condition is valid if:

$$H \in \tilde{\mathcal{C}} \Rightarrow \forall p_i \in H, p_i \in \tilde{\mathcal{C}}_i \tag{3.5}$$

These approximations require the robot root to only rotate on the yaw axis when navigating, that is a common assumption for biped locomotion [DT14].

One drawback of using a single-layer height map for this validity condition is its limitation to environments without an upper floor or ceiling. Regarding $\tilde{\mathcal{C}}$ constraint, the trivial case is that if a point p_i lies inside this volume, there is a collision (Figure 3.8c). However, we do not know if there is a collision if a p_i lies right above the robot. There are two cases: first, the robot is under an object (ceiling) and not in collision; second, the robot is inside the object (e.g. wall) and so is in collision (Figure 3.8d). From the height map, we cannot distinguish these cases and thus we choose to consider both as a collision. As all our test terrains do not contain an upper floor and we do not perform any locomotion task like passing under a gate, such an approximation is sufficient. In the future, another solution will be required to navigate such scenes like using a two-layers height map or voxels to better approximate the terrain geometry or using alternative validity conditions.

Another limitation comes from the height map resolution that can fail to approximate the constraints $\mathcal{R}$ and $\mathcal{C}$. On reachability, a low height map resolution can fail to visualize a scene composed of small and spaced surfaces. Consequently, it can wrongly consider that $H \notin \tilde{\mathcal{R}}$ (i.e. the terrain under the robot is not reachable) when in reality $H \in \mathcal{R}$. To remove such cases, we can reconfirm the invalidity of the configuration with the full constraint $\mathcal{R}$, resulting in a reasonable trade-off between computation time and completeness. As for the collisions, the opposite can happen where a collision is not detected by the approximation, $H \in \tilde{\mathcal{C}}$, when in reality $H \notin \mathcal{C}$. Consequently, only the full validation $\mathcal{C}$ can detect these cases. In practice, our steering method learns to keep a safe margin between the obstacles and the robot trunk to avoid collisions and so, implicitly avoid these collision detection failures. Therefore, we choose to rely on the approximated constraint $\tilde{\mathcal{C}}$ that is sufficient to detect most of our robot trunk collisions.

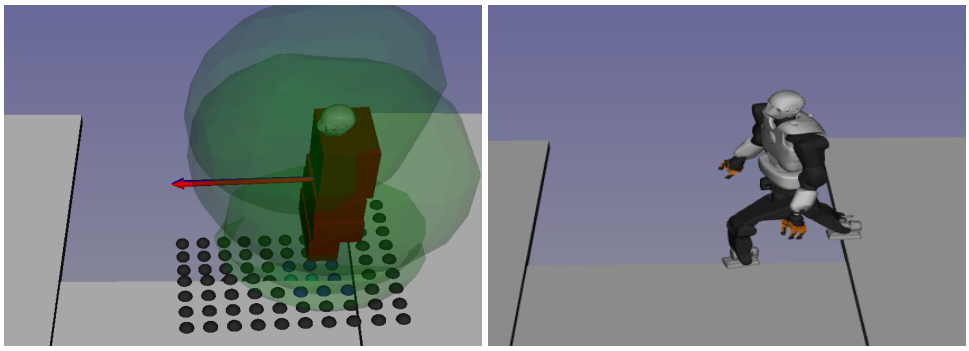


Figure 3.9: Valid configuration with $\tilde{\mathcal{C}}$ and $\tilde{\mathcal{R}}$ leading to a blocking configuration above the hole.

Finally, we add one additional constraint to further improve the guide quality for the biped locomotion task. One default of the original validity function is that configurations above the void but touching the edge of the terrain with the range of motion are considered valid (Figure 3.9),

leading to infeasible trajectories by our contact planners. To avoid these configurations, we add another validity condition on the height map H^* located directly under the root of the robot to be reachable at all times (dots under the robot in Figure 3.8b). We name the reachability constraint with the stricter condition $\tilde{\mathcal{R}}^*$ where:

$$H \in \tilde{\mathcal{R}}^* \Rightarrow \exists p_i \in H^*, p_i \in \tilde{\mathcal{R}}_i \tag{3.6}$$

With this strategy, configurations are considered invalid if the robot stands above the void. It can be limited on terrains with very spaced surfaces, but in all our scenarios this condition offers good results in terms of guide path quality. This solution was not yet implemented in our first version of LEAS [Che+21] where we learned by reinforcement such additional constraints using the contact planner as a guide path validator, hence leading to longer training.

In the context of RL, where the agents perform millions of steps, the approximations $\tilde{\mathcal{R}}^*$ and $\tilde{\mathcal{C}}$ significantly improve the training time. While the gains in computation can not be fairly compared due to their difference in programming language and optimization, our implementation resulted in an average of 20 times faster computation time compared to the full constraints $\mathcal{R}$ and $\mathcal{C}$, hence enabling us to train and test more models.

3.3.2 Terrain Generator

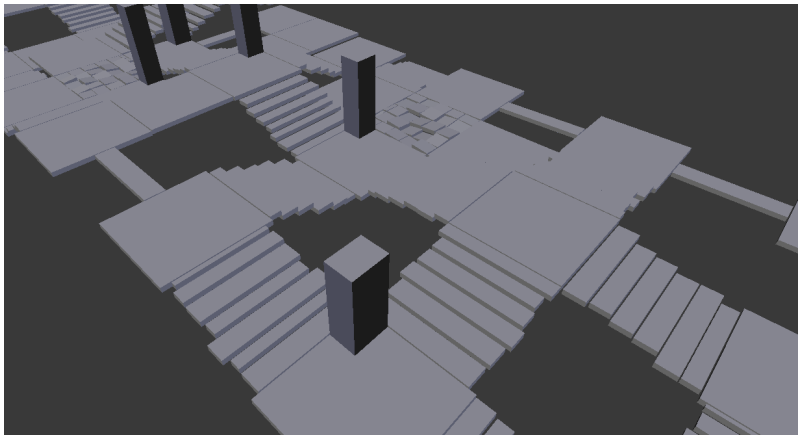


Figure 3.10: Example of a random scene generated, containing different types of tiles connected. Initial tiles: flat ground with and without obstacles. Transition tiles: rubbles, bridge, and stairs.

Our goal is to train one policy to navigate through all our test scenarios. To improve its generalization capability, our RL agent has to learn on various terrains. To that end, we extended the library used in [Son+20] to generate random terrains composed of rubbles, stairs, bridges, and obstacles. The code is available on GitHub [SC].

This library generates a terrain that we can divide into tiles. Each tile belong to two categories: *Initial* or *Transition* (Figure 3.10). The first category *Initial* contains two types of tiles: flat ground with or without obstacles. The second category *Transition* connects two *Initial* tiles and contains three types: rubbles, bridge and stairs.

We represent the terrain as a grid where each cell is a fixed-sized tile. Starting from an *Initial* tile, we build a tree filling pseudo-randomly its empty neighbors. Each *Transition* is unique as the characteristics of each surface composing it are random, but also depend on the height of the tiles it connects.

At the end of the terrain generation, we can identify all the links created. A link is defined as a sequence of *Initial* - *Transition* - *Initial* tiles in a line, and that we can further divide into several categories in function of their difficulty. Finally, we extract the possible starting positions for the robot on each link, corresponding to areas on the *Initial* tiles.

Terrains generated by this library can contain enough elements, depending on the grid dimension and the random seed, for the agent to develop its navigation skill, then generalize to our test scenes that contain the same type of transitions.

Some limitations come with the use of this library for our training. The first is that all the tiles have the same dimensions, fixed to 2x2 meters for all our training scenes. Its dimensions have to be carefully tuned according to the size of the observable height map by the agent. Indeed, a too large visual field could lead the agent to specialize in navigating scenes with this specific tile size, e.g. the agents can specialize in crossing stairs or bridges of 2 meters. In our experiments, with a visual field of 110 cm in the front (Section 3.2.2), learning on a terrain of 2x2 meters tiles is sufficient to learn local navigation behaviors. The second limitation is that even though the *Transition* and obstacle tiles are unique and randomized, they still fall into the same categories (i.e. stairs, rubbles, bridge, obstacle). Consequently, we could question the ability of the agent to navigate on very different terrains. Empirically, this was not a problem to generalize to all our test scenarios mostly containing the same type of elements, however, a higher diversity may be required in the future. The performance of the agent also depends on the size of the terrain it has been trained on. We had to generate several random arenas before finding one having enough elements and links. In our experiments, terrains of dimension 5x30 are the maximum we can have in HPP as the terrain loading and processing times exponentially increase with the number of surfaces.

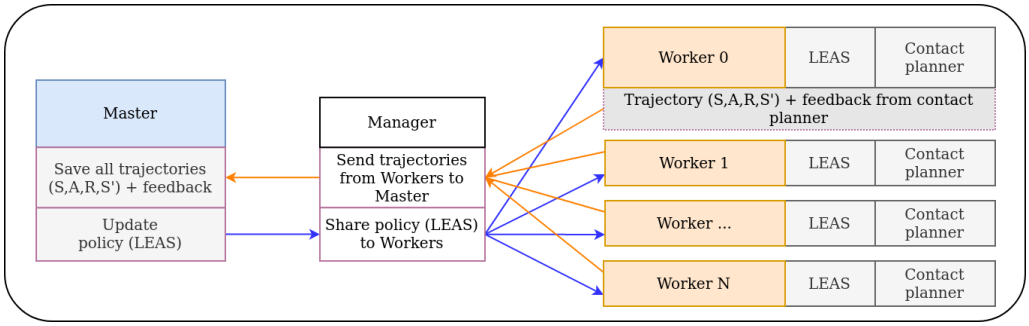
Finally, several options can be explored to learn from the random scenes generated. We can have several parallel agents learning in different scenes or modifying the scene during the training. We can also use the difficulties of each link in the scene to perform curriculum learning [Nar+20], starting from the easiest link to the most difficult one. In a previous test, we ran several agents during the training on different terrains and following such a curriculum learning, for example starting from flat ground, then switching to stairs, and finally, a terrain randomly generated. However, it did not further improve the learning time, as our agent is already able to learn quickly from a single 5x30 random scene without these methods.

3.3.3 Master-Workers Architecture: Asynchronous contact planners

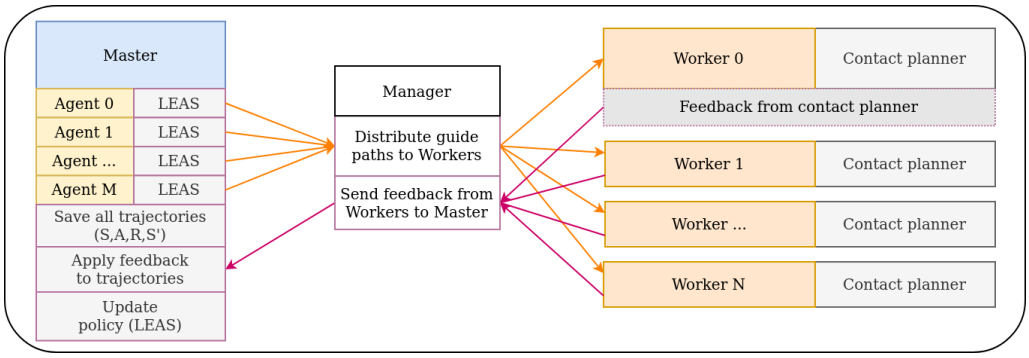
Several state-of-the-art RL algorithms are available and can be separated into different groups, each presenting some pros and cons, and that can lead to various results depending on their hyperparameter tuning and the task to perform [Hen+18; And+20]. As a consequence, it can be difficult to judge which algorithm to use for a given task other than empirically. Among the most recent and popular RL algorithms, we have Proximal Policy Optimization (PPO) [Sch+17] in the category of on-policy algorithms, and Twin Delayed DDPG (TD3) [FHM18] and Soft Actor-Critic [Haa+18a] in off-policy.

In this work, we experimented on PPO and TD3, both implemented in Stable Baseline [Hil+18] that we can easily adapt for our task. Off-policy algorithms, such as TD3 and SAC, are known to be more sample efficient but can lack stability compare to on-policy RL algorithms. That is why many recent works in RL use PPO [Kwi+22] that, to this day, can be easier to tune and more stable during the learning, even for tasks that require data-efficiency as [Ope+19]. In our first version of LEAS [Che+21], the training with PPO was slow due to its poor sample efficiency (order of hours without contact planning and tens of hours with our sample-based contact planner). However, this issue was fixed with the optimizations and approximations discussed in the previous section thus making both off- and on-policy pertinent choices. We tested TD3 and PPO for our navigation task and we observed with TD3 a slower and less stable convergence than PPO, using the default hyperparameters as defined in Stable Baseline or with manual tuning. As a result, we decided to use PPO to learn LEAS.

Given the design of LEAS, we have two components to consider during the training. First, our steering method is a policy taking actions in the environment and saving the state sequence



(a) Asynchronous guide path and contact planning



(b) Synchronous guide path and Asynchronous contact planning

Figure 3.11: Two versions of Master-Workers architecture to learn LEAS plugged to a contact planner.

as a guide path. Second, the contact planner computes the contact sequence along it and gives feedback on its result to LEAS. In this design, the main limitation comes from the contact planning validation at the end of each episode. If we simply perform this contact planning inside each agent, as all parallel agents are steps synchronous (or trajectory synchronous), if one agent computes a contact sequence, all the others have to wait for it to finish. Consequently, the computation time increases with the number of agents in parallel. We have two solutions to solve this problem: (a) Compute all trajectories asynchronously or (b) compute all guide paths synchronously and perform P2 asynchronously. We implemented both solutions (a) and (b) using a master-worker architecture (Figure 3.11).

Solution (a) dissociates the learning operated by the master and the trajectory generation performed by the workers (so-called distributed RL). The advantages of this method are its scalability, as we can add during the learning as many agents as desired that can run on different machines, and its modularity as the agents can be initialized on new terrain and stopped if needed. This method is used in IMPALA [Esp+18] running 32 CPUs and RAPID [Ope+19] with 500 CPUs, showing impressive results to learn challenging and complex tasks. However, this method also requires a lot of data exchanges between the RL algorithm and the workers as explained in [Maj+21]. The load on the network increases with the number of data (the trajectories) exchanged between the master and the workers, plus the parameters (policy network) that must be shared with all the workers after each update. As a consequence, this approach can result in some hardware and implementation limitations depending on the available resources, and a possible delay in the policy of the asynchronous agents.

The other solution (b) lets the master generate all the trajectories with one or several synchronous agents, then dispatch the guide paths generated to the workers that compute P2 and return the result to the master. It is a solution specifically adapted to our problem and simple to implement that can be added on top of any on-policy or off-policy RL algorithm. The key advantage of this method is the reduction of data exchanges between the master and the workers. Only

the guide path, a sequence of positions and orientations, and the feedback from the contact planner transit on the network. However, if the guide path generation P1 is faster than P2, we can observe some lags between the synchronous agents and the workers. That is why we need to balance their number or add more workers to follow the flow. Also, the parallel agents during the learning run on environments that can not be changed manually, making it a simple but less modular method overall compared to (a).

In our experiments, solution (a) led to a bottleneck on our hardware due to the amount of data exchanged, and going for pre-made implementation as IMPALA was not necessary for our problem that only requires a few hours of training compared to the tasks solved in [Esp+18; Ope+19]. Therefore, we opted for solution (b) which is a good compromise between synchronous and asynchronous steps. In the future, we would like to take a look at fully decentralized architectures [Wij+19] that solve the problem of network overload by sharing only the gradients for training.

With setup (b), we will now perform several tests on basic scenarios with LEAS without contact using the original PPO algorithm, then plug it into a contact planner using our modified PPO version to analyze its impact on the trained policy.

3.4 Results

Table 3.1: Parameters

State	81	Max Episode Length	800
Actions	4	Parallel agents M	6
a_{max}	0.08 m/s^2	Workers	0 or 6+
v_{max}	0.2 m/s	Batch size	$4096 * M$
$v_{desired}$	0.1 m/s	Mini-Batch size	256
ω_{max}	$\pi/9$ rad/s	Learning rate	$[5e-4, 1e-5]$
Timestep T	0.2 s	Noptepochs	10
Local height map H	10x14	Discount Factor (γ)	0.97
z_{max}	-0.2 m	Clip range	0.2
z_{min}	-1.4 m	w_{ori}	0.4
w_{dir}	1.0	w_{acc}	0.1
w_{ω}	0.1	w_{alive}	1.0

We use the HPP software [Mir+16] and the humanoid robot Talos model [Sta+]. Our algorithm is implemented in python using the PPO implementation of Stable Baselines [Hil+18] modified for our Master-Workers architecture. All parameters in the environment and hyperparameters to control the learning process can be seen in Table 3.1. We use the terrain generator described in 3.3.2 to generate the 5x30 training ground in Figure 3.12.

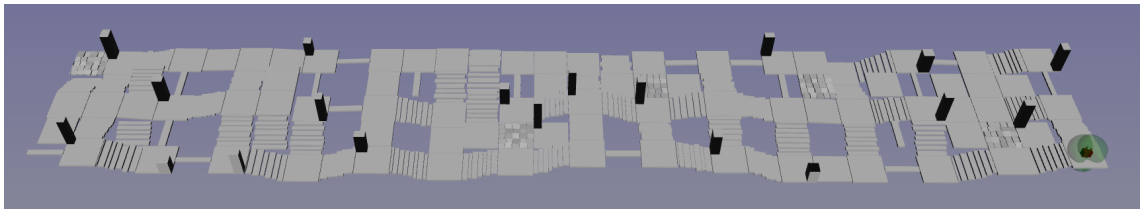


Figure 3.12: Training ground of LEAS: a 5x30 arena (corresponding to 10x60 meters) composed of 86 links: 17 bridges, 31 stairs, 8 rubbles, and 30 flat ground with obstacles. All links are two-way and have different elements characteristics and slopes.

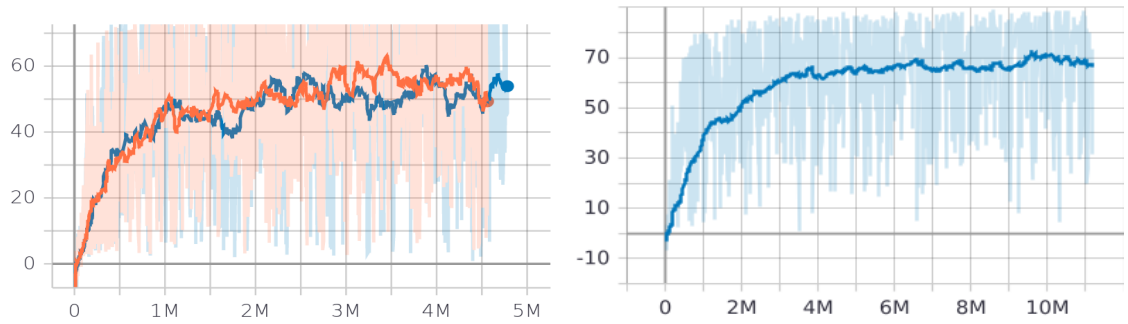
We set a fixed timestep value $T = 0.2$ seconds, which corresponds to a maximum distance between each state on the path of 4 cm for $v_{max} = 0.2$ m/s, and 2 cm for $v_{desired} = 0.1$ m/s. Each episode has a maximum length of 800 steps, meaning that the guide path can contain up to 800 states (maximum distance of 32 meters). During the training, the robot has to navigate the terrain at the desired velocity while keeping a valid state ($\tilde{\mathcal{R}}^*$ and $\tilde{\mathcal{C}}$). To further guide the learning, we initialize each episode as follows: first, the robot has to cross a link, i.e start from an initial tile and navigate through a transition tile (stairs, rubbles, or bridge), then, it has to follow a new random goal direction that is updated every $n_{rand} \in [200, 800]$ steps. The first goal prioritizes the task of crossing a link that we will perform in our basic scenarios, while the others often lead to a dead-end or a difficult path where the agent will have to learn if it has to stop or has the ability to cross it.

We linearly decay the learning rate from $5e^{-}4$ to $1e-5$ over 2 millions training steps and set a discount factor $\gamma = 0.97$. A method to tune the discount factor is to calculate the half-life $\tau = \frac{1}{1-\gamma}$ which roughly corresponds to the number of steps considered to adapt the agent behavior. For LEAS, it is equal to $\tau = \frac{1}{1-0.97} \approx 33$ steps. As a result, steps from $[0, 33]$, $[33, 66]$, $[66, 99]$, $[99, \infty]$ will roughly account for 63%, 23%, 8% and 6% of the sum of discounted rewards respectively. The distance traveled by the RL agent after 33 steps is equal to 66 cm and 132 cm for velocity equals to $v_{desired}$ and v_{max} respectively. We emphasize that we want to learn a steering method to navigate locally and that the agent only needs to think about its very near future. For example the detection of an obstacle should impact its action only when close to the robot (distance inferior to 1 meter).

In this chapter, the number of workers planning contacts is set to 0 as we train LEAS for a pure navigation task without a contact planner. In our experiment, the number of parallel agents M is limited to 6 due to hardware limitations.

The PPO actor and critic are two distinct networks with hidden layers of size 128x64x32. Tuning the number of nodes and hidden layers in the machine is empirical and depend on the task to perform. In general, deeper networks with more nodes means a better capability to solve very complex tasks and less under-fitting. However, it also means more parameters to train and can be prone to over-fitting. In our experiments on LEAS, we found no noticeable difference between 2 or 3 hidden layers and nodes number ranging from 64-256. However, we increased the network size of 64x64 from the first version of LEAS [Che+21] for more potential scalability in our tests in terms of observations and contact planners.

We train LEAS without a contact planner and evaluate the model after 12 million steps corresponding to around 2 hour of training on a PC with an Intel Core i7-8700 (12 cores, 3.20Ghz, 16GB ram). Learning curves can be seen in Figure 3.13.



(a) (b)

Figure 3.13: Learning curves of LEAS without contact planner feedback (pure navigation task): (a) Comparison between an acceleration control (Orange) with delay, (Blue) without delay on the position update, and (b) the complete learning curve with an acceleration control.

3.4.1 Comparison: Steering Method Designs

Table 3.2: Comparison of the steering method characteristics.

Steering Method	RB-Lin	RB-Kino	LEAS (ours)
Goal Connection	Exact pos,ori	Exact pos,vel,acc,ori	Near pos
Dynamic constraints	1 v_{max}	2 v_{max}, a_{max}	3 $v_{max}, a_{max}, \omega_{max}$
Terrain aware	No	No	Yes
Path planning dependant	Yes	Yes	No

We compare LEAS, a flexible alternative to our previous steering methods. Their characteristics can be seen in Table 3.4.1.

Goal connection: RB-Kino [Fer+17] and RB-Lin [Ton+18a] are two steering methods that connect exactly some initial and goal configurations in position, velocity, and orientation. RB-Kino uses the Double Integrator Minimum Time [KS14] to also add a constraint on the acceleration. LEAS does not exactly connect the initial and goal configurations as it is trained to follow a given goal direction. Consequently, it can at best end up near the goal position. The capability to connect exactly two configurations is needed in robotics for hand manipulation. Arguably, in locomotion, this accuracy is not required in most scenarios. This is especially true on long paths with key waypoints to sequentially reach up to a distant objective, where passing close enough to each waypoint is sufficient. As LEAS uses a target direction instead of a target configuration, it also enables us to use a joypad-like control, which is equivalent to a very distant moving target that the agent tries to reach. We simulate this kind of control for the training of LEAS by randomly repositioning its goal.

Dynamic constraints: RB-Lin is a modified linear interpolation that is constrained in orientation and velocity. It first rotates the robot toward the target at ω_{max}, then moves toward it at $v_{desired}$. RB-Kino requires two parameters v_{max} and a_{max} that act as strict constraints on both velocity and acceleration. However, no constraints are set on the angular velocity ω_{max} and this can be a problem as we will see in chapter 4. On the other hand, the control of LEAS constrains each configuration with v_{max}, a_{max}, ω_{max}. In this thesis, we only use quasi-static contact planners, so v_{max} and a_{max} are considered during the guide path planning but not the contact planning phase. However, using LEAS with a kinodynamic contact planner is a possibility for future work.

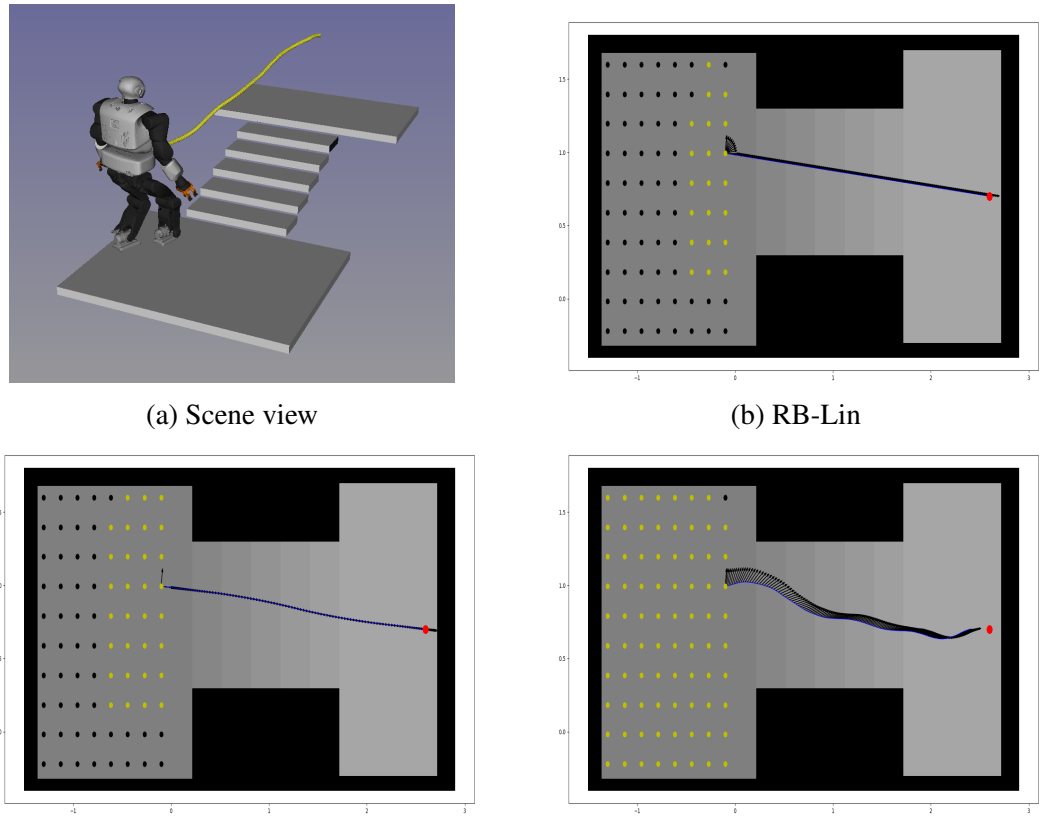
Terrain-aware and path planning: RB-Kino and RB-Lin both require a path planning algorithm to place additional waypoints to reach a distant goal. In contrast, LEAS can observe its surrounding terrain and can locally navigate it in a given direction, hence removing the need for path planning on basic scenarios (i.e. crossing a link). In complex scenarios, path planning algorithms can also be used to place waypoints followed by LEAS. In this work, LEAS can navigate the same waypoints computed by the RRT with RB-Kino. In future work, having an RRT with LEAS in the same manner as RL-RRT [Chi+19b] is a work in progress.

Finally, we recall the main advantage of LEAS over our previous steering methods which is the use of Reinforcement Learning that, through the trajectory validation by the contact planner,

can change its behavior to fit it and generate more likely feasible paths.

3.4.2 Test Scenarios

As LEAS does not connect exactly with the goal position, we consider the goal to be reached when the distance from the current state on the guide path to the goal is lower than a distance threshold ϵ set to 20 cm for all our scenarios.



(a) Scene view (b) RB-Lin

(c) RB-Kino (d) LEAS (ours)

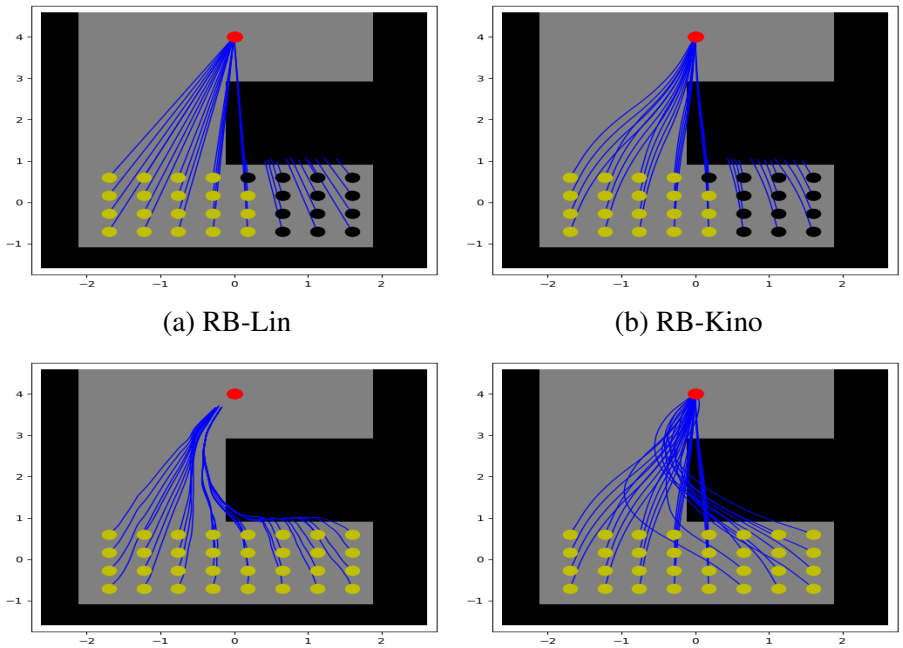
Figure 3.14: Comparison on stairs where dots represent initial states from where the SM generates a valid (yellow) or invalid (black) guide path up to the goal (red). The height map is represented by the grey shades from dark to bright, that are lower and higher heights respectively. We illustrate for each SM one trajectory where the black arrows represent the robot orientation along the guide.

Stairs. We compare the navigation skills of LEAS and our previous steering methods on the stairs scenario (Figure 3.14). We uniformly sample 80 initial root configurations at the bottom of the stairs (Figure 3.14a), from which the steering methods have to generate guide paths up to the same fixed goal location. Each initial state is oriented at 90° with respect to the stairs, to better show the rotation performed by the steering methods, and with a null velocity. Initial states from where the SM reaches the goal with a valid guide path are represented by yellow dots. Conversely, black dots represent states where either the SM fails to reach the goal or generates an invalid guide path (i.e. ground not reachable or collision) and thus requires additional waypoints to be provided by a path planning method. For RB-Kino, we need to define some constraints that are the velocity and acceleration desired on the goal configuration and that have an impact on the shape of the trajectory. In this scenario, we set both goal acceleration and velocity to 0.

RB-Lin is designed to first rotate the robot toward the goal, then generate a straight line up to the goal (Figure 3.14b). Its main limitation is that it succeeds only when directly placed in front

of the stairs, and further initial positions result in guide paths where the robot is unable to touch the ground. RB-Kino presents the same limitation as RB-Lin, plus another one on the orientation where it rotates directly from $90°$ to $0°$ in one timestep (Figure 3.14c). This is a problem inherent to RB-Kino which is the correlation between the initial velocity and the angular velocity. As a consequence starting with null velocity, as we will see later on, is critical with our contact planners as such fast rotation is not feasible kinematically. In practice, we avoid this problem by adopting the same strategy as RB-Lin by first rotating the robot to always start RB-Kino with a $0°$ orientation.

LEAS succeeds on a much broader range of initial states, hence removing the need for path planning in most cases (Figure 3.14d). Our steering method generalizes to this scenario that has never been encountered during its training: it rotates and moves toward the goal, detects the stairs on its local height map, and adapts its velocity v_z to climb it while keeping a valid state, finally reaching an area of 20 cm around the goal.



(a) RB-Lin (b) RB-Kino

(c) LEAS (ours) (d) RB-Kino + RRT

Figure 3.15: Comparison of steering methods on hole scenario where dots are initial states from where the SM generates a valid (yellow) or invalid guide path (black) invalid guide path up to the goal (red). Guide paths are represented by the blue trajectories.

Hole. We now compare our steering methods on our hole avoidance scenario (Figure 3.15). Each initial configuration has a null velocity and is oriented toward the goal located on the other side of the hole. RB-Kino and RB-Lin generate valid guide paths only when the problem is feasible in a straight line. It means that they require path planning for half of the initial configurations (Figures 3.15a and 3.15b). In contrast, LEAS always succeeds in avoiding the hole and reaching the goal. Even when starting from difficult configurations on the right, LEAS detects the hole and slowly slides along it while keeping a valid state.

Finally, Figure 3.15d illustrates RB-Kino combined with an RRT path planning algorithm to solve the hole scenario. The trajectories of (a,b,d) are generated using the full validity condition $\mathcal{R}$ without the stricter constraint (Section 3.8). Consequently, most guide paths generated lie above

the hole with a maximum distance of 40 cm from the surfaces, corresponding to the width of the range of motion of the Talos robot legs. This brings out the main problem of our previous validity condition $\mathcal{R}$ and the need to have a stricter constraint like $\tilde{\mathcal{R}}^*$ to avoid such configurations along the guide path.

Evaluation of LEAS. We evaluate the success of LEAS (Table 3.3) to navigate all the transition tiles of our evaluation terrain, never met during its training (Figure 3.16). We uniformly sample 100 configurations before each transition tile with two different initial orientations, $0°$ and $180°$, to evaluate their impact on LEAS success.

For both rubbles and stairs (down), LEAS succeeds in all 100 trajectories for both initial orientations. Resuls show that our steering method succeeds in crossing all transition tiles of our scenario with a near 100% success rate, hence demonstrating its terrain-aware navigation skill. However, it also fails in some bridge and stairs (up) scenarios. This is mainly due to initial configurations facing backward and very close to the transition tile to cross. The agent thus fails to rotate the robot, to have a clear vision of the terrain in its back, before navigating it. These difficult cases appear on the bridge where the agent stops the robot near the void to avoid falling, and on the stairs up where the agent collides with it during its rotation. In pratice, the robot will never be positioned in such extreme initial configurations, hence the results show that our steering method can successfully navigate all our practical cases.

Finally, we construct a scenario to sequentially cross different transition tiles (Figure 3.17). To do so, we place manually chosen waypoints on the terrain to traverse some stairs and a bridge. Results show that LEAS successfully reaches each waypoint, while generating valid configurations.

Initial orientation	Rubbles	Bridge	Stairs (down)	Stairs (up)
$0°$	100 %	100 %	100 %	100 %
$180°$	100 %	87 %	100 %	97 %

Table 3.3: Success rate of LEAS For two initial orientations on 100 uniformly sampled trajectories for each transition tile (Figure 3.16). Initial positions can be far or near the transition tiles. LEAS needs enough time to rotate the robot, detect the transition tiles and navigate through it while keeping a valid configuration.

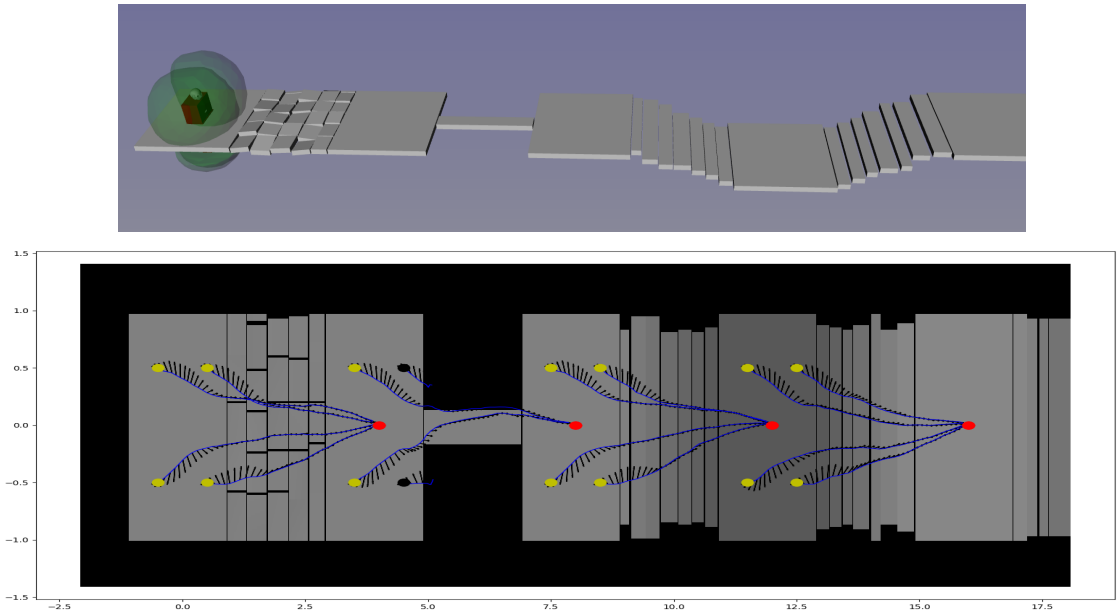


Figure 3.16: Evaluation terrrain with 4 transition tiles: rubbles, bridge, stairs (down), and stairs (up). Examples of extreme initial configurations (dots) from where LEAS generates: a valid (yellow) or invalid guide path (black) up to a goal (red). Black arrows are the root orientation along the guide (blue).

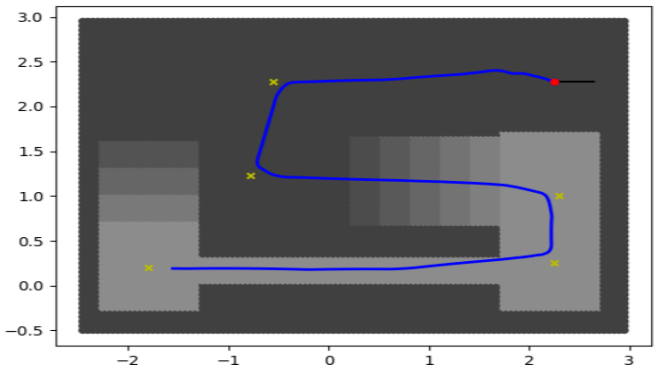


Figure 3.17: LEAS connecting manually placed waypoints (yellow).

3.5 Discussion

LEAS offers a flexible alternative to our previous steering methods. Our results show that LEAS performs better in generating valid guide paths, under collision-avoidance and reachability criteria, than RB-Kino and RB-Lin thanks to its local terrain awareness. This method learned by reinforcement succeeds in navigating all our test scenarios, while permitting a more flexible user control by providing a goal direction instead of a goal configuration. However, there is still a lot of scope for improvements in its design.

3.5.1 Limitations

Terrain visualization. Parameters $[z_{min}, z_{max}]$ as well as the height map resolution impacts the agent capabilities to detect its environment. In our experiments, the most difficult scenarios to handle were navigating through bridges and obstacle avoidance, where the tuning of these values was critical. First, a too high value z_{min} can lead LEAS to interpret the void surrounding the

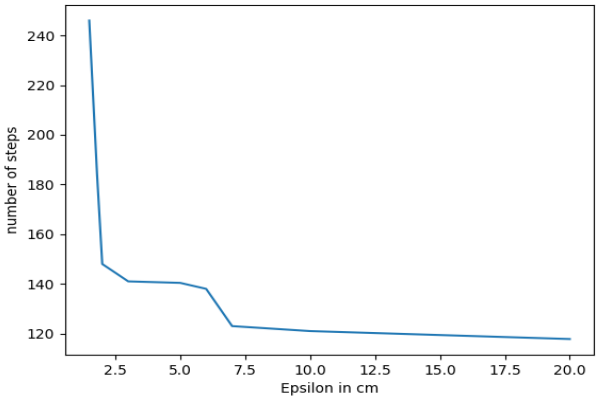


Figure 3.18: Average number of steps required to reach an area of radius ϵ cm around the goal.

bridge as a potential stair. This caused LEAS to lower the robot root, at the limit of the trunk collision with the bridge, to confirm if what it sees is a stair or a void. As a consequence, it can lead to some collisions with the bridge due to these two conflicting behaviors. A similar problem appeared for a too low value z_{max}, with the interpretation of an obstacle as the next step of some stairs and causing a collision with it. Finally, setting a too wide bound $[z_{min}, z_{max}]$ made the values of the heightmap difficult to interpret by our agent, that was unable to capture small to medium terrain variations. The values we have selected offered a balanced trade-off, however, further tests are required to improve LEAS navigation performance.

To avoid such collisions, one could also increase w_{alive} to encourage the robot to act more carefully. We tested this solution that greatly improved the LEAS capabilities for collision avoidance while keeping reachable states. However, LEAS was acting so safely that it did not dare to cross difficult transition tiles, sometimes making the robot idle in front of bridges to safely accumulate the reward R_{alive}. The final value $w_{alive} = 1$ decided for LEAS offers a compromise safety/risk but requires further investigation.

Constraint on the orientation. We observed that LEAS trained on complex terrains tended to explore its surroundings by erratically orienting its root and so its local height map to have a better vision. We limited this undesirable behavior by setting a high weight w_{ori} to enforce its orientation toward the goal. However, such a reward will probably not work to navigate cluttered environments requiring the robot to sidewalk.

Goal reaching accuracy. We recall one of the main limitations compared to our previous steering methods which is the non-exact connection to the goal position. We consider the goal as reached if the distance between the state on the guide path and the goal is inferior to a threshold of value ϵ. In all our scenarios, we fix $\epsilon = 20$ cm that we consider accurate enough. We further evaluate the accuracy of LEAS to reach a goal position in Figure 3.18. For this test, we set the robot orientation back to the transition tile (rubbles for this scenario) and we uniformly sampled 50 configurations. The task to perform is to rotate the robot toward the fixed goal on the other side of the rubbles and get close to it, less than ϵ cm. We test several values $\epsilon \in [1.5, 20]$ cm. For all $\epsilon \geq 1.5$ cm, all 50 trajectories reach the goal under the threshold. However, we can observe that as the value of epsilon decreases, the number of steps to reach the goal increases. Indeed, the agent passes close by the goal but misses the area of radius ϵ around it and has to move back and forth to reach such an accuracy. In this scenario, LEAS reaches the goal at once for all threshold ≥ 7.5 cm. In this thesis, we set $\epsilon = 20$ cm to have a sufficient margin of error.

3.5.2 Future Improvements

For a pure navigation task, LEAS without a contact planner does not need to clearly identify the surfaces of the terrain and solely focus on the reachability and collision conditions from the height map. In this work we use a simple multilayer perceptron, as done in [Chi+19b; Chi+19a] with 1-D lidar values, that is simple to implement and fast to train for our navigation task. However, we believe that LEAS could greatly benefit from convolutional neural network architectures to extract features from the height map and learn a better terrain representation [Pen+17b; Tso+20; Gan+22].

The training terrain diversity was sufficient for LEAS to learn our navigation task, but more terrains could improve its generalization capabilities. To do so, we could improve our terrain generator, or use another simulation environment such as RaiSim [HLH18] allowing us to efficiently switch between different terrains.

Finally, we discussed using curriculum learning [Nar+20] to incrementally increase the complexity of the tasks to solve during the training, which did not improve the result of LEAS. But several methods in the literature could improve its learning efficiency and overall performance. Especially mirroring all states relative to the robot orientation axis are other strategies to explore, that could greatly improve the sampling efficiency during the training and train a policy with a symmetric behavior.

3.5.3 Conclusion

We presented LEAS, an RL steering method to locally navigate complex terrains and to generate guide paths subject to reachability and collision avoidance constraints. Such terrain-aware steering methods remove the need for a path planning algorithm, expensive to compute, in most of our basic scenarios. As a result, LEAS can directly be integrated as the module P1 of our locomotion pipeline (Figure 3.1).

Yet, we did not solve the feasibility problem between our navigation task ($P1$) and the contact planner ($P2$), but solely a surrogate approximation of it. The next two chapters will then extend LEAS training to adapt it to the exact feasibility of the contact planner.

Chapter 4

LEAS with an Acyclic Sampling Based Contact Planner

Contents

We have defined in the previous chapter a methodology to learn by reinforcement a local navigation method. At evaluation time, our method LEAS produces a guide path along which we want to compute a contact sequence.

Chapter 3 only proposed to use simple termination conditions during the training: the reachability $\tilde{\mathcal{R}}^*$ and the collision-free $\tilde{\mathcal{C}}$ conditions. Yet those approximated conditions are necessary but not sufficient to guarantee the existence of a feasible contact sequence along the guide.

We now propose to explore the following question: *What is a good guide path to generate a valid contact sequence ?* For that, we will rely on an existing implementation of a contact planner, which we refer to as the second stage of our framework ($P2$). In this chapter, we use an acyclic

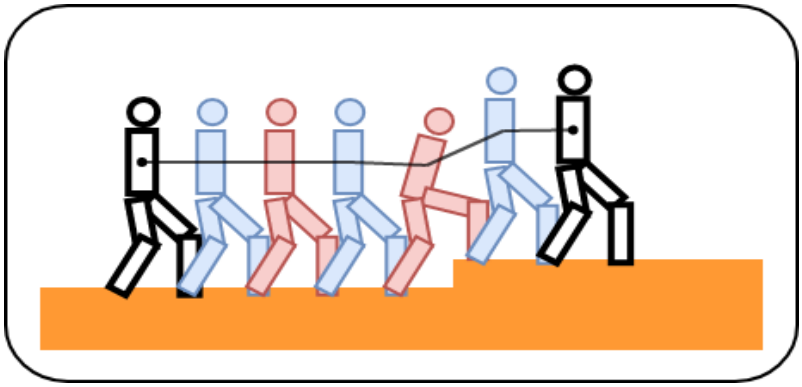


Figure 4.1: Given a guide path, our contact planner generates key configurations in contact along it.

Sample-Based Contact Planner [Ton+18a] for multiped robot locomotion, which for convenience we will refer to as SBCP. As discussed in Chapter 3, we want LEAS to achieve two objectives:

- Navigating using a local height map of the terrain toward a goal direction subject to some reachability and collision constraints;
- Generating guide paths ($P1$) maximizing the likelihood of finding feasible contact sequences along it with a contact planner ($P2$).

In the context of Motion-before-Contact, we will show how LEAS learns to address by reinforcement one of the key limitations of this strategy: the non-guarantee of the feasibility of the guide ($P1$) by a contact planner ($P2$).

This chapter is organized as follows: Section 4.1 is an overview of SBCP and its limitations due to the guide path. While this contact planner is not a contribution to this thesis, it is important to have a strong understanding of its structure and of the success and failure modes. Section 4.2 shows how to train LEAS with the contact planner as a black-box. Section 4.3 presents a benchmark and analysis of the results obtained by this contact planner with our steering methods. Finally, section 4.4 discusses about our implementation choices and potential improvements to our work.

4.1 Summary of the Sampling Based Contact Planner

4.1.1 Notations

The Sample-Based Contact Planner (SBCP) [Ton+18a] belongs to the motion-before-contact family of contact planners, using a rough robot root trajectory (guide path) to generate the contacts along. While it can perform multi-contact locomotion tasks such as standing up or climbing stairs using a handrail, in this work we are interested in biped walking for the scenarios tested in Chapter 3 where we only enable foot contacts with the terrain.

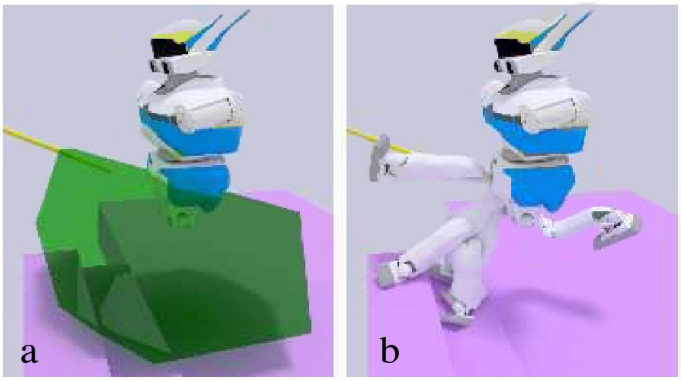


Figure 4.2: Sampling of limb configurations for the right leg of HRP-2 robot with (a) the leg range of motion and (b) some sampled configurations. Source: Tonneau et al. [Ton+18a].

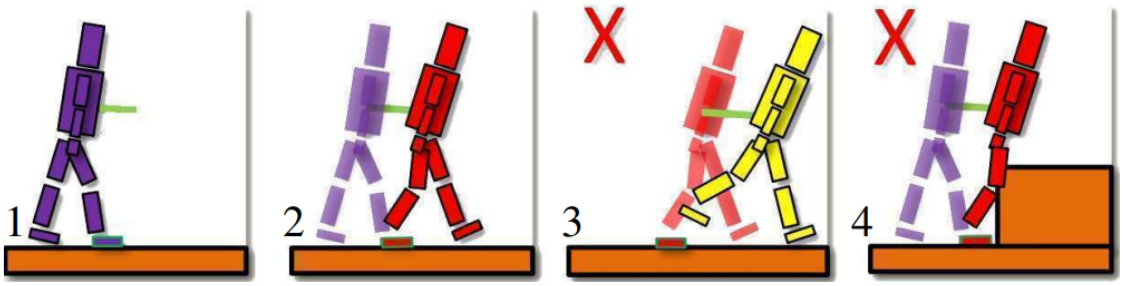


Figure 4.3: Contacts are maintained on the next root position (green line) if kinematically feasible and without collision (2), they are broken otherwise (2,3). Source: Tonneau et al. [Ton+18a].

We use the following syntax:

- q_{base} is the robot root configuration in $SE(3)$ (position and orientation);
- q_k is the partial robot configuration on limb $k \in \{left\_leg, right\_leg\}$ with 6 degrees of freedom each. The methods allows a more generic multiped setup but we limited our benchmarks to biped cases;
- $q = [q_{base}, q_{left\_leg}, q_{right\_leg}]$ is the robot whole body configuration.
- $G = [q^0_{base}, q^1_{base}, ..., q^{M-1}_{base}]$ is the guide path, described as a sequence of M root configurations;

4.1.2 Overview of the Contact Planner

Given an initial whole body configuration q^0, the role of SBCP is to populate a guide path in input $G = [q^0_{base}, q^1_{base}, ..., q^{M-1}_{base}]$ with a sequence of configurations in contact double support with the terrain and static equilibrium [q^0,q^1,..., $q^{M'-1}$] following **exactly** the guide path.

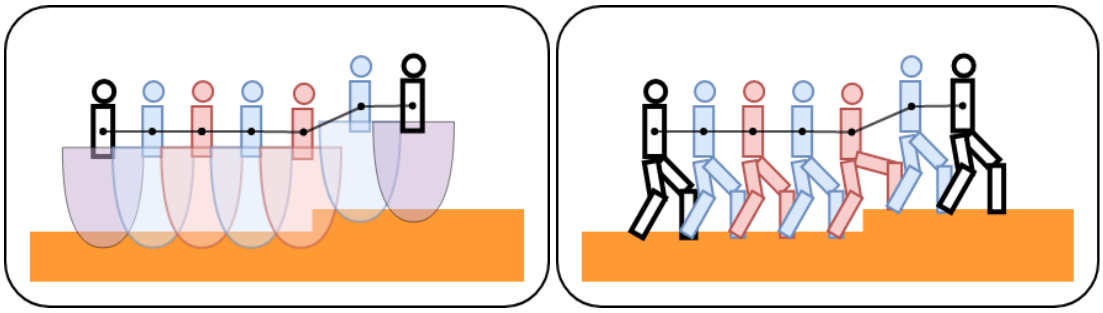
This is done in two steps: first, an offline sampling of limb configurations, second, an online search on these samples for the contacts to perform.

First, an offline database is generated containing configurations q_k for each limb k. To do so, we randomly sample N configurations for each limb inside their range of motion (Figure 4.2). This database will be searched at runtime, to select the most suitable limb configuration for a given root according to user-defined heuristics. We will discuss the heuristics used to search and select contact configurations, in particular the quantification of the robustness of the configuration balance, later in this chapter.

Starting from a robot configuration q^i in contact whose root is at q^i_{base}, the contact planner moves the robot root to the next position on the guide q^{i+1}_{base}. To do so, the contact planner performs the following steps (See Figure 4.3):

- Maintain previous contacts: we check kinematically if the contact with the terrain on limb k can be maintained from the next root position q^{i+1}_{base}. If not, the contact is broken.
- Repositioning: if more than one contact is broken in the previous step, one or more configurations in contact are added at q^i_{base}, to ensure that at most one contact is broken between q^i and q^{i+1}.
- Creating contacts: if only one contact is broken, we create a new one with the limb k that has been contact-free the longest. If the contact creation fails, we try to create contact with another free limb or reposition another limb in contact. If no contact can be computed to reach the next root configuration q^{i+1} after a fixed number MAX_TRIES of trials, SBCP returns the sub-sequence of successful configurations in contact up to q^i.

In this work, we focus on biped walking and so we perform contact only with the feet of the robot. Obviously, setting a value of MAX_TRIES $= 2$ for the repositioning is sufficient for biped locomotion, where a higher threshold does not improve further its success but increases the computation time. Finally we recall the motion-before-contact strategy of SBCP in Figure 4.4 with (a) generation of a guide path from an initial to a goal configuration respecting the reachability constraint $\tilde{\mathcal{R}}^*$ and collision-free $\tilde{\mathcal{C}}$, (b) creation of new contacts to sequentially reach each root configuration along the guide.



(a) (b)

Figure 4.4: Motion-before-contact strategy with SBCP: (a) generating a valid guide path and (b) computing the key configurations in contact along.

4.1.3 Limitation due to the Guide

In Chapter 3, we have shown how LEAS generates guide paths subject to reachability and collision constraints ($\tilde{\mathcal{R}}^*$ and $\tilde{\mathcal{C}}$) for configuration validity.

We now raise the key problem of the motion-before-contact strategy that is the non-guaranteed feasibility of the guide ($P1$) by the contact planner ($P2$). Our experiments will show that the major factor impacting the contact planning success is the guide path in input. The problem is that we do not know what is a good guide path for this contact planner. The reachability and collision conditions introduced for configuration validity are necessary but not sufficient to generate guide paths feasible by this contact planner. As a consequence, it is difficult to define new additional constraints or heuristics to our guide path generator ($P1$) to better approximate the contact planning feasibility ($P2$).

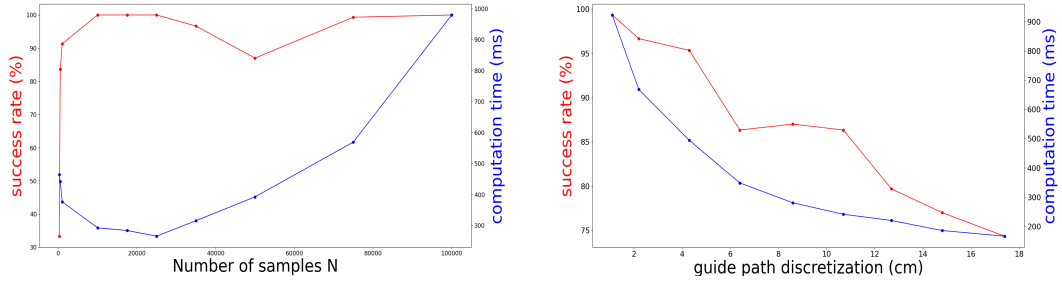
Our solution to solve such a feasibility problem is to learn with LEAS how to validate its guide paths with SBCP, which computes the contact sequence along them. In other words, we want LEAS to answer the question: *What is a feasible guide path for this contact planner ?*

4.1.4 Trade-off in the Parameters

Before jumping onto our solution with LEAS, we identify two critical parameters whose tuning strongly impacts the success rate and the computation time of the contact planner:

- N, the number of randomly sampled configurations q_k for each limb k;
- ΔD, the discretization step on the guide path, corresponding to the average distance between each root configuration q^i_{base} to q^{i+1}_{base} provided as input to the contact planner.

The first parameter N is set offline, when initializing the database of limb configurations. The second parameter ΔD corresponds to the guide path discretization. A trade-off has to be found on these values to comply with the requirements on the computation time and the success rate of the contact planner. That is why we provide an analysis of both parameters and explain our choices for this work. The results are presented in Figure 4.5.



(a) (b)

Figure 4.5: Analysis of the success rate and computation time of SBCP with a guide path computed by RB-Kino in function of: (a) the number of samples N and (b) the guide path discretization ΔD (average distance between the root configurations).

Number of samples. The random sampling strategy consists in building a database of available configurations for each limb of the robot that is searched during the contact creation. The number of samples N is thus critical to provide enough coverage of the robot leg range of motion. In this test, we measure the impact of the number of samples on the success rate and the computation time for complete successful guide paths only (Figure 4.5a). The success of a guide path is defined as the percentage of the successful guide with SBCP. For example, with a guide path of 100 configurations, if the contact planner fails to reach the configuration of index 50 then the success value is 50%. We perform this test on the climbing stairs scenario (see Section 4.3.3) where we generate some valid guide paths with RB-Kino from 100 uniformly sampled initial positions at the bottom of the stairs oriented toward the goal. We then evaluate the results of different sample values N for 3 different seeds, with an average guide path discretization step $\Delta D = 2$cm. We will explain the choice of ΔD later on in this section.

We observe that (red) the success rate rapidly converges to a maximum success rate on this task, and that (blue) the contact planning time increases with the number of samples N. From these results, we can deduce that a number of samples $N = 10K$ is sufficient to solve this particular locomotion task. However, reducing the value of this parameter beyond a certain point increases the computation time, where the contact planner requires contact repositioning to compensate for the lack of sampled configurations leading to a robust solution. As a result, we found the sweet spot to be $N = 25K$ for the stairs and that we keep fixed in our setting across our scenarios. It is unsure how this small number of samples copes with more complex terrains like our 5x30 training ground, but we can expect a lower feasibility space with SCBP and where the guide path quality becomes more important.

Guide path discretization. The discretization step ΔD on the guide path corresponds to the average distance between each root configuration q^i_{base} to q^{i+1}_{base}. A small ΔD means a higher number of root configurations on the guide, that results in a higher number of footsteps with SBCP. A small value increases its success rate but also its computation time as observed in Figure 4.5b. This test is performed on the same stairs scenario with the steering method RB-Kino. RB-Kino generates guide paths continuous in time with non-constant velocities that we discretize with a fixed timestep. We sample some timestep values and measure the corresponding average velocity on the guide, and thus its discretization step ΔD. We compute 50 trajectories for each measurement and average the results over 3 different seeds and three sample numbers, $N = \{25, 50, 100\}$K.

We compare these results to the original paper [Ton+18a] using the robot HRP-2 that has a similar maximum step size to the robot Talos we work on. They state that "in most scenarios the torso of HRP-2 moves about 15 cm (for easy scenarios) between two postures, but only 3 cm for the car egress scenario". From our measurements, we see that a small discretization step $\Delta D = 2$ cm on the guide increases the success rate of the contact planner. Indeed, it is easier for the robot to create a contact to reach a closer root configuration than a distant one. The choice of ΔD is a trade-off between the success rate and the number of steps (and so the computation time). One could just set an average discretization $\Delta D = 2$ cm that could maximize the success rate in most scenarios. However, this also results in a high contact planning time (over a second) and a hundred footsteps for a trajectory of 3 to 4 meters for the climbing stairs task, which is not tolerable. In this work, we aim for an average discretization $\Delta D = 10$ cm resulting in a satisfactory number of steps and computation time, while providing a sufficient success rate on SBCP with our previous steering methods. Just as the tuning of the parameter N, such a high value of ΔD lowers the success rate of the guide with SBCP, and so greater emphasizes the quality of the guide (i.e. the positioning of the root configurations relative to the terrain). On all our steering methods, we further discretize the guide when required to obtain an average $\Delta D = 10$ cm and thus provide a fairer comparison with closely matched settings.

We presented the general concept behind our sample-based contact planner ($P2$) computing a contact sequence following exactly the guide path in input ($P1$), and our design choice for two critical parameters. As we have seen, it is important to design the guide path to have good properties with respect to the contact planner. However, these properties are difficult to express explicitly, making it difficult to build heuristics or a more optimal decision process for computing the guide path. We then propose to formulate it as a maximization of the success likelihood of the contact planner, building upon LEAS for the learning algorithm. Let's now see the details of this proposition.

4.2 Implementation Details

Our goal with our steering method LEAS is to overcome the challenges coming from the connection of the guide path planner and the contact planner with a high-level approach. We want to generate guide paths that are feasible by the contact planner.

In the previous chapter, we have shown how LEAS performs in a navigation task with its local terrain-aware capability. In simple scenarios, LEAS does not heavily rely on path planning algorithms compared to our previous steering methods, resulting in an overall faster generation of guide paths. We now want to improve the feasibility of the generated guide paths by our sample-based contact planner. We employ the strategy described in the previous chapter with our Master-Workers architecture where the workers validate the trajectories generated by the LEAS agents, by computing the contact sequence along the guides with SBCP used as a black box.

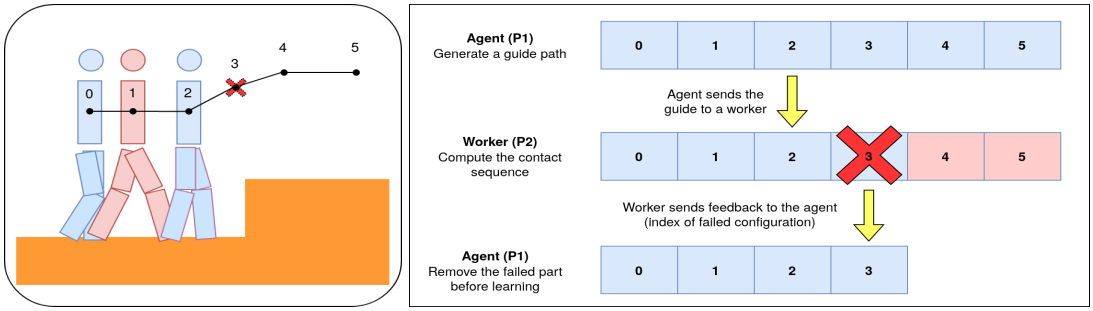
4.2.1 Parameters of LEAS with Contact Planner

RL policy. We employ the same network architecture, states, actions, rewards, and hyperparameters as described in Chapter 3 and set the number of asynchronous workers computing the contacts to 6. Our method LEAS takes as input a local height map of the terrain and a direction to the goal to locally navigate the terrain. States generated by LEAS are subject to the reachability $\tilde{\mathcal{R}}^*$ and collision-free $\tilde{\mathcal{C}}$ conditions, plus a new additional constraint on the whole guide path that is to succeed with the contact planner, that we will explain later on.

Contact planner parameters. We authorize the contact with the limbs $k = \{left\_leg, right\_leg\}$ to perform a biped locomotion task only, and generate for each limb a number of samples $N = 25$K with a fixed seed (empirically selected). Finally, the number of maximum repositioning attempts in the contact planner is set to MAX_TRIES $= 2$, sufficient for our biped locomotion task.

Guide path discretization. The discretization of the guide generated by LEAS corresponds to an average of $\Delta D = 2$cm between each configuration. As seen in the previous section, this value presents on SBCP the best success rate at the cost of a very high computation time, as it generates contact plans with too many contacts (over a hundred steps to walk 3 to 4 meters). That is why we want to aim for an average $\Delta D = 10$cm, presenting a suitable trade-off in computation time and success rate with this contact planner. To do so, we further discretize the guide path before giving it to the contact planner by only keeping 1 out of 5 root configurations on it, to aim for a discretization step of $\Delta D = 2 \times 5 = 10$cm in average (further discussed in section 4.4).

4.2.2 Validation by the Contact Planner



(a) (b)

Figure 4.6: Generation of a guide path that fails with SBCP: (a) generation of a valid guide path up to configuration 5 and fail of the contact planner at 3, (b) diagram of the learning process from this trajectory.

During the learning, the workers perform contact planning with SBCP to validate the trajectories generated by LEAS. Therefore, it learns how to generate feasible guide paths according to the validity conditions $\tilde{\mathcal{R}}^*$ and $\tilde{\mathcal{C}}$, plus the contact planner ($P2$). Our contact planner returns the contact sequence along the guide, up to the last successful root configuration q^i_{base}. The index of this configuration is retrieved and sent back to the master that prunes the section of the guide path that failed as depicted in Figure 4.6. LEAS has to find a trade-off between learning the navigation task presented in Chapter 3 and succeeding in the contact planning.

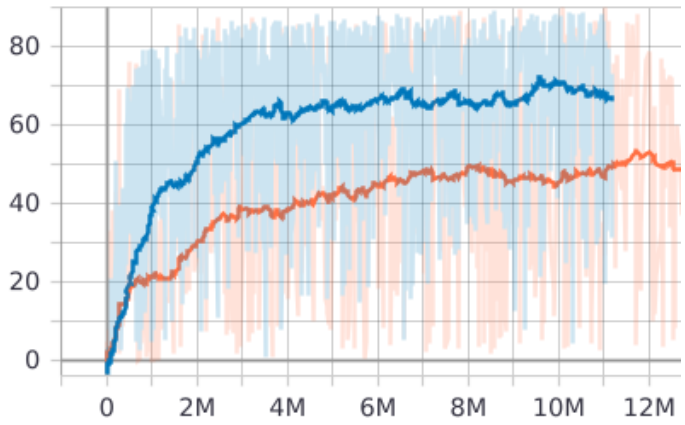


Figure 4.7: Learning curves of (blue) LEAS-P1 trained without contact planner and (red) LEAS-P2 trained with guide validation by SBCP.

4.2.3 Training

We train LEAS with SBCP on our training terrain presented in the previous chapter. The model is evaluated after 12 million steps corresponding to 8 hours of training on a PC with an Intel Core i7-8700 (12 cores, 3.20Ghz, 16GB ram). Learning curves of LEAS without a contact planner, referred to as LEAS-P1 (Chapter 3), and LEAS with the contact planner, LEAS-P2, are shown in Figure 4.7 (the maximum reward is equal to 100 for trajectories in a straight line at $v_{desired}$). The P2 validation increases its probability to meet a terminal condition (i.e. failing the contact planning) and forces LEAS to adapt its behavior to succeed with the contact planner, hence resulting in a lower average reward per episode.

4.3 Results

For all our tests, we uniformly sample some initial configurations on a flat area and set a fixed goal configuration on the other side of the obstacle to cross, corresponding to trajectories of 3 to 4 meters long in average. Each steering method has to generate guide paths with valid configurations ($\tilde{\mathcal{R}}^*$ and $\tilde{\mathcal{C}}$) up to the target and compute a contact sequence along it with SBCP. The success rate presented in the results corresponds to the percentage of valid guide paths reaching the goal and being successful with SBCP. We tune the discretization timestep of RB-Kino and RB-Lin such that the discretization step along the guide is on average equal to $\Delta D = 10$cm, just as LEAS.

We compare the method proposed in this chapter (LEAS-P2), with the method proposed in the previous chapter (LEAS-P1) and the two model-based steering methods used as benchmarks, RB-Lin and RB-Kino. We evaluate them on scenarios we know, from experience, to be challenging for contact planners. For each scenario, we analyze the changes in the guide path generation, hinting us about our contact planner capability relative to the guide.

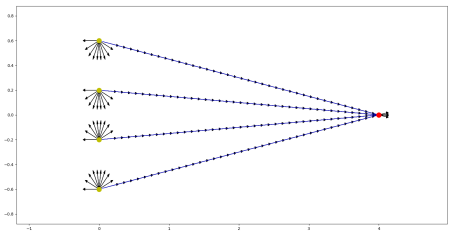
4.3.1 Scenario A: Rotation

Given initial and target configurations, we uniformly sample some initial robot orientations and set some fixed velocity values to evaluate the success of all steering methods with the contact planner. The orientation value corresponds to the angle between the target direction and the root orientation. We evaluate the success rate of each steering method that has to rotate and move the robot to the goal. This scenario simulates a problem that some heuristic methods cannot solve well. While these methods could be specifically patched to counteract it, it is interesting to understand these behaviors.

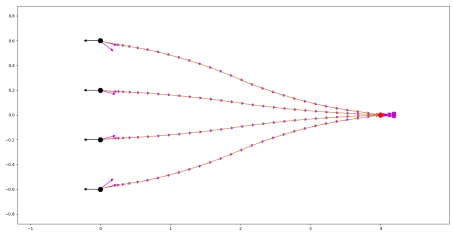
In this experiment, all steering methods succeed in generating a valid guide path, reachable and without collision, up to the target but present different results as seen in Table 4.1 and Figure 4.8.

Table 4.1: Scenario A: success rate for different orientations on flat ground.

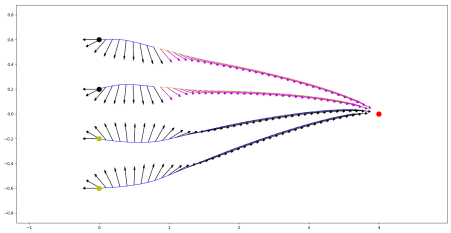
Parameters	RB-Lin	RB-Kino	LEAS-P1	LEAS-P2
0° to 30°				
$\|\|v\|\| = 0$	100%	100%	100%	100%
$\|\|v\|\| = 0.04$	x	100%	100%	100%
$\|\|v\|\| = 0.07$	x	100%	100%	100%
60° to 120°				
$\|\|v\|\| = 0$	100%	61 %	75%	100%
$\|\|v\|\| = 0.04$	x	100%	84%	100%
$\|\|v\|\| = 0.07$	x	100%	89%	100%
150° to 180°				
$\|\|v\|\| = 0$	100 %	0 %	22 %	100%
$\|\|v\|\| = 0.04$	x	0 %	35 %	100%
$\|\|v\|\| = 0.07$	x	43 %	42 %	100%



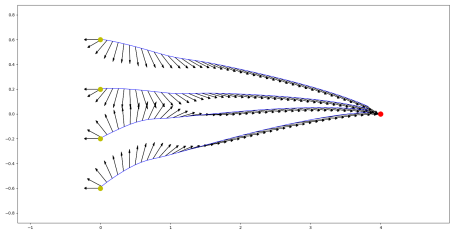
(a) RB-Lin



(b) RB-Kino



(c) LEAS-P1



(d) LEAS-P2

Figure 4.8: Scenario A, Trajectories for initial configurations at 180° and $||v|| = 0.04$ m/s with (blue) the successful and (yellow) the failed part of the guide, and (black arrow) the robot root orientation.

RB-Lin rotates at a fixed angular velocity $\omega_{max} = \frac{\pi}{9} = 20$ deg/s which is equal for $T = 0.2$ s to a maximum rotation of $\omega_{max} \times T \times 5 = 20°$ between each configurations on the guide (keeping 1 out 5 guide configurations). As expected, the results show that for any initial orientation, rotating the robot with a step angle of 20° before moving it to the goal generates feasible guide paths by the contact planner. RB-Lin shows that a rotation toward the goal, with a step angle 20° between the guide path configurations, can safely be performed before moving the robot. In our tests, all

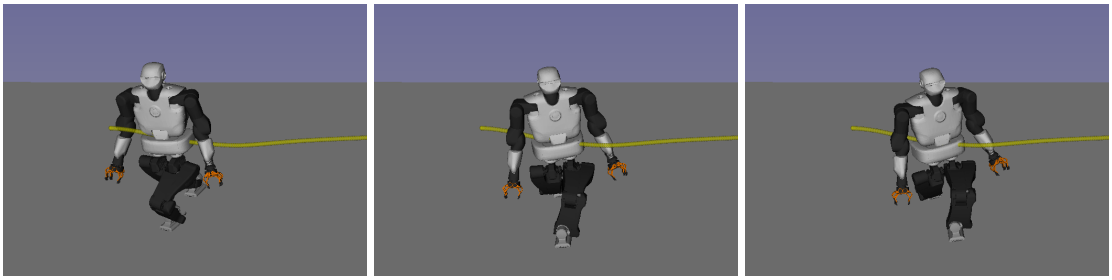


Figure 4.9: Scenario A: Sequence of configurations where LEAS-P1 fails the rotation along the guide.

step angles in the range $]0, 90]$ degrees were successful, but in practice, a step angle of $45°$ may be the maximum feasible by the Talos robot in reality.

RB-Kino succeeds for all orientations in $[0, 120]$ degrees. However, we can see the limitation of RB-Kino when starting with a (near) null velocity as discussed in the previous chapter, and as a result, fails the contact planning as such high rotation is not kinematically feasible. We recall that in practice we adopt the same strategy that RB-Lin by rotating the robot first before using RB-Kino, and thus we always avoid such cases.

Then we compare LEAS-P1 and LEAS-P2 that have been trained with and without feedback from the contact planner respectively. LEAS-P1 is subject to two major rewards motivating its behavior: R_{ori}, encouraging him to be oriented to the target and R_{dir} to move toward it at $v_{desired}$. As a result, LEAS-P1 balances these two rewards by moving and rotating at the same time. Additionally, LEAS-P2 is subject to another constraint that is to succeed with the contact planner ($P2$).

The results show that LEAS-P1 works for all initial orientations in $[0, 30]$ degrees but may fail for initial orientations superior to $60°$. Indeed, rotating while moving toward the target is more difficult and requires adapting simultaneously the robot velocity and angular velocity. Finally, as the initial velocity increases, we can note that so does the success rate with LEAS-P1 and RB-Kino.

In contrast, LEAS-P2 learns how to rotate and succeeds in every rotation scenarios, thus showing the benefit of our solution. Empirically, we noticed that LEAS-P2 tends to move in the direction of its orientation to help the rotation task (Figure 4.8d), whereas backward translations with LEAS-P1 could result in failure due to heuristics defined in the contact planner (Figure 4.8c). That is why some guide paths generated by LEAS-P1 leads to some configurations that do not permit further movement (Figure 4.9). In contrast, LEAS-P2 automatically discovers how to generate guide paths avoiding the selection of such configurations from the samples.

4.3.2 Scenarios B: Obstacle Avoidance

Table 4.2: Scenario B1: Comparison on the success rate and robustness.

Parameters	Terrains	RB-Lin	RB-Kino	LEAS-P1	LEAS-P2
SR	Hole	82%	74%	100%	100%
	Bridge	86%	78%	96%	100%
Robustness	Hole	21.7	20.9	32.2	33.3
	Bridge	20.84	25.8	24.36	27.3

Scenario B1 - Hole and Bridge. We compute contact sequences on guide paths generated by our steering methods on the hole and bridge scenarios (Figure 4.10). These two scenarios are

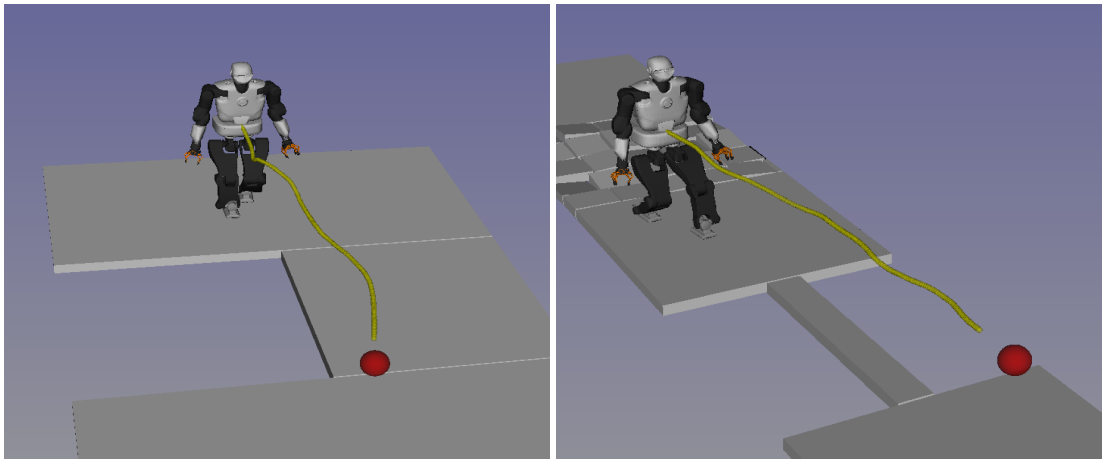


Figure 4.10: Scenario B1: The steering method has to generate a valid guide path (yellow) up to the goal (red), then compute a contact sequence along it with SBCP.

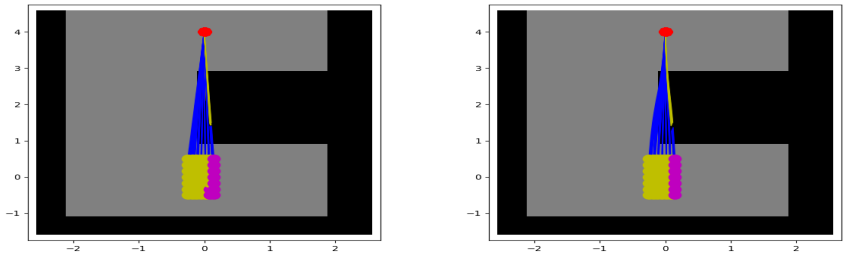
similar as they can both lead to weak contact configurations located very close to the edge of the obstacles that are unfeasible by the robot in reality (Figure 4.12). We evaluate for both terrains the success rate with the contact planner and measure the average $robustness$ of the configurations in contact when passing near the obstacle. The robustness quantifies how far the center of mass of the robot is from the boundaries of the friction cones of the contact forces, giving us a measure of the static equilibrium of one robot configuration (we refer the reader to [Ton+18a] for further details). In this work, we use the measure of the robustness to give us an overall estimation of the stability of the contact plan generated, where a lower value means a more difficult guide path in the input of the contact planner (i.e. too close to the obstacles, too high or low...) leading to unstable configurations in contact. Conversely, a high robustness value should be the result of a "good" feasible guide path.

We uniformly sample some initial configurations on a small area from where all our steering methods can generate valid guide paths ($\tilde{\mathcal{R}}^*$ and $\tilde{\mathcal{C}}$) up to the target. We set each initial robot configuration oriented toward the target, located on the other side of the obstacle, with an initial velocity $||v|| = 0.04$ m/s. Results are shown in Table 4.2 with SBCP success rate for both bridge and hole scenarios. Figure 4.11 shows generated trajectories on the hole scenario.

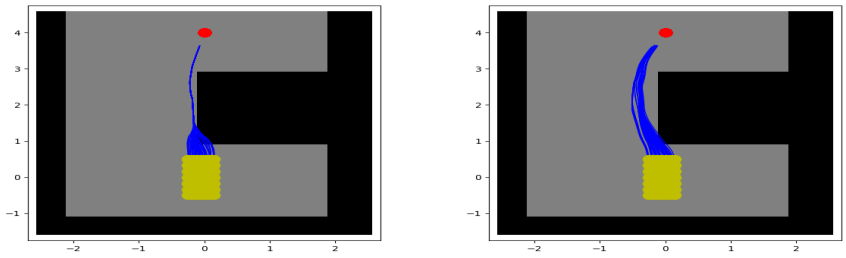
Results show that RB-Lin and RB-Kino generate valid guide paths up to the goal, but do not consider the terrain, resulting in configurations passing close to the void. As a consequence, these guide paths lead to a low success rate and some weakly robust contact plans on both terrains.

In comparison, LEAS-P1 and LEAS-P2 generate guide paths with a strong hole avoidance behavior leading to a higher success rate and more robust configurations in both scenarios. LEAS-P1, trained without contact planner feedback, performs well in these scenarios where the stricter reachability constraint $\tilde{\mathcal{R}}^*$ encourages to keep the robot root above the ground while keeping a safe margin of error. As a result, trajectories of LEAS-P1 tend to be further from the hole than RB-Kino and RB-Lin (Figure 4.11). However, we can still notice some successful guide paths with weakly robust configurations that are unfeasible on the real robot (Figure 4.12c). In contrast, the results of LEAS-P2 demonstrate how it can produce guide paths staying even further from the hole, thus generating more robust configurations and increasing the contact planning success rate.

Results in Table 4.2 show that there is a correlation between the contact planning success and the robustness measured. This is expected as SBCP from an unstable configuration often fails to generate a new contact to reach the next root position on the guide. However, we also notice that this correlation is not pertinent for all scenarios, like the bridge where RB-Kino presents a high robustness score close to LEAS-P2 but the worst success rate among our steering methods. Indeed some configurations can be selected as the most robust but they may not permit the generation of a new contact (Figure 4.12a and 4.12b).

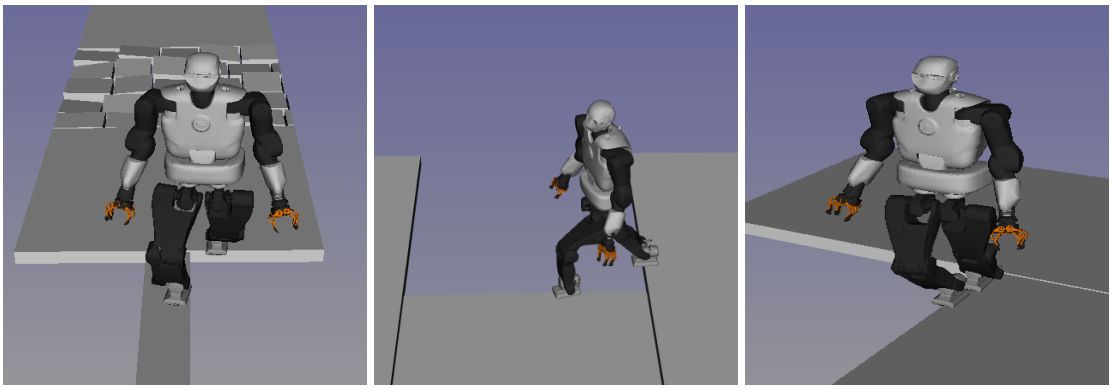


(a) RB-Lin (b) RB-Kino



(c) LEAS-P1 (d) LEAS-P2

Figure 4.11: Scenario B1 (hole): initial configurations succeeding (yellow dot) and failing (magenta dot) with SBCP.



(a) (b) (c)

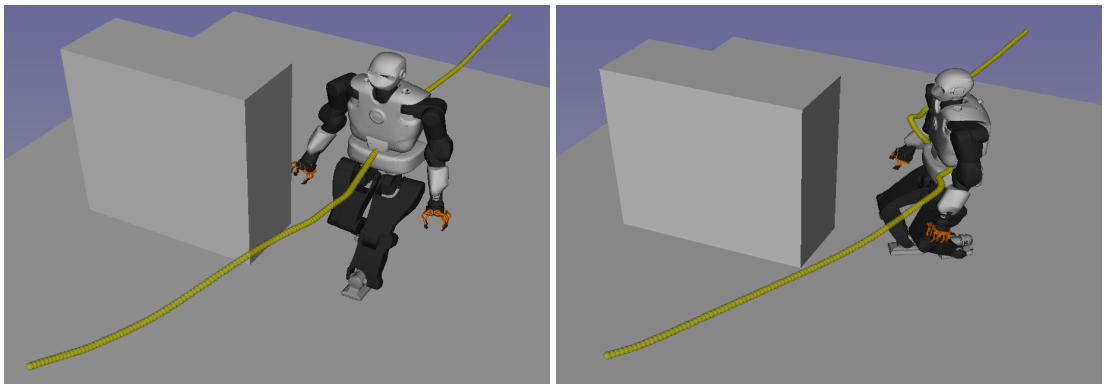
Figure 4.12: Scenario B1: (a)(b) blocking Talos configurations above the void and (c) configuration valid but unfeasible on the real robot.

We have shown in the hole and bridge scenarios how LEAS-P2 learns to increase the success rate of the contact planner and indirectly improve the robustness of the contact plan through its validation during the training.

Scenario B2 - Wall. We now evaluate on the wall scenario what kind of additional constraints LEAS can learn from the contact planner. In the wall scenario, LEAS must focus on generating a collision-free guide path ($\tilde{\mathcal{C}}$). Results are shown in Figure 4.13. As the original collision volume (i.e. the trunk) does not include the hands of the robot, guide paths generated by LEAS-P1 (a) pass very close to the wall and fail with SBCP as for the given root position and orientation, a collision occurs between the hand of Talos and the wall (Figure 4.13a). In contrast, LEAS-P2 (b) learned during the training how to avoid such collision with the hand by rotating the robot (Figure 4.13b). We note an interesting behavior where LEAS balances the rewards R_{ori} and R_{dir}, to keep its orientation and to move toward the target respectively, with the contact planning constraints, hence making the robot sidewalk along the wall while keeping it in its visual field.

We reproduced a similar experiment in our original work [Che+21], learning LEAS with $\tilde{\mathcal{R}}$ (i.e. without the stricter constraint to lie above the ground, cf. previous chapter), thus leading LEAS-P1 to have the same success rate than RB-Lin and RB-Kino on the hole and bridge scenarios (Table 4.2). Then we trained LEAS-P2 with this same reachability condition $\tilde{\mathcal{R}}$, plus feedback from the contact planner. As expected, the policy learned to move the robot while keeping a safe distance from the void and reached the same success rate as our actual version of LEAS-P1 and LEAS-P2 trained with the stricter reachability condition $\tilde{\mathcal{R}}^*$.

These experiments demonstrate what kind of behaviors LEAS can learn to generate feasible guide path by our contact planner. From these results, we decided to implement these constraints directly inside the validation function: first, the stricter constraint to force the robot to lie above the ground $\tilde{\mathcal{R}}^*$ (already implemented in the results), then extending the collision volume of the robot to include its hand positions when its arms are attached. While reducing the feasibility space of the generated guide paths, these approximations greatly accelerate the training ($\approx 20 - 30\%$) as the guide paths generated are subject to tighter validity constraints (i.e. root above the ground and with less probability of collisions), thus resulting in a faster contact generation.



(a) LEAS-P1: Collision (b) LEAS-P2: Avoid the contact

Figure 4.13: Scenario B2: (a) LEAS-P1 collision of the robot hand with the wall, (b) LEAS-P2 rotates the robot to avoid a such collision.

4.3.3 Scenario C: Uneven Terrains

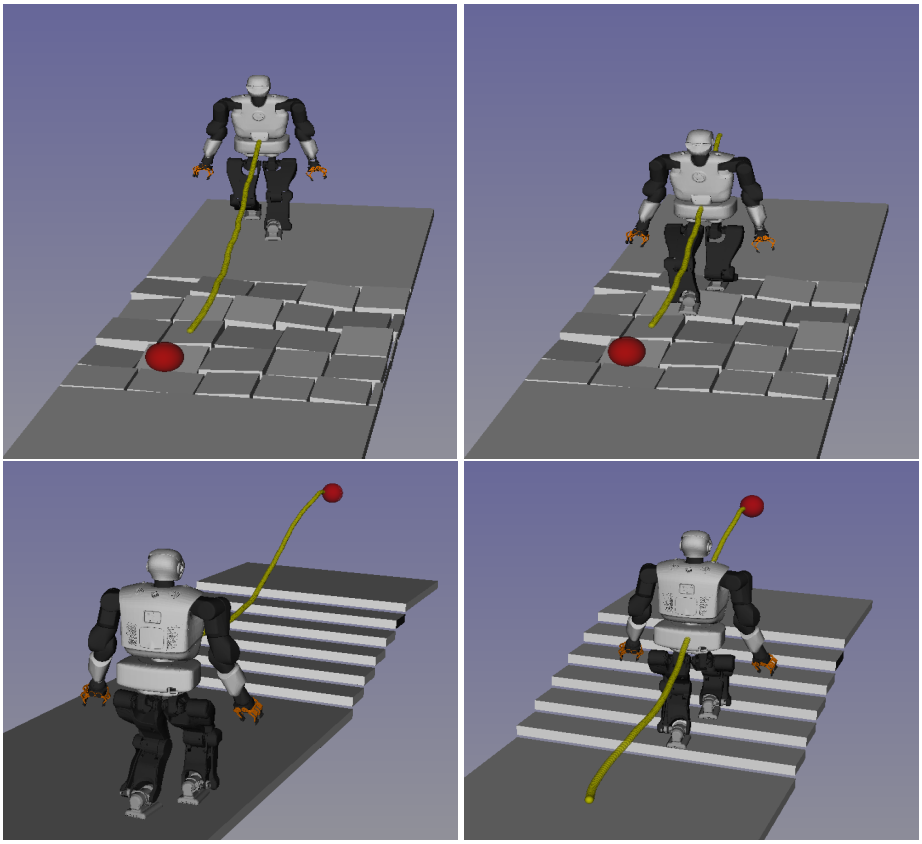


Figure 4.14: Scenario C: Contact planning on rubbles and stairs scenarios, with (yellow) the guide path generated by LEAS-P2 and (red) the goal.

We now present a comparison on two complex terrains: the stairs and the rubbles. In [Ton+18a], these scenarios were easily solved using the steering methods RB-Lin and RB-Kino with a near 100 % success rate. However, these tests were performed for a reduced set of initial configurations starting directly in front of the obstacle and oriented toward it. In this work, we extend these tests on a wider range of initial configurations with RB-Kino (with RRT path planning if required), LEAS-P1 and LEAS-P2. Then we measure their success rate and the robustness of the contact plans generated when crossing the terrain. For all our tests, we orient the robot directly toward the target with an initial velocity of $v = 0.04$ m/s.

Table 4.3: Scenario C: Comparison on the success rate and robustness.

Parameters	Terrains	RB-Kino+RRT	LEAS-P1	LEAS-P2
SR	Rubbles	100%	100%	100%
	Stairs	74%	100%	100%
Robustness	Rubbles	30.4	24.9	26.45
	Stairs	24.3	28	33.2

Scenario C1 - Rubbles. Starting from a flat ground area and oriented toward the obstacle, the robot has to cross some small uneven surfaces with different heights and orientations and reach a

fixed target on the other side (Figure 4.14). Results in Table 4.3 show no difference between RB-Kino which does not require path planning to succeed in this scenario, LEAS-P1, and LEAS-P2. All three steering methods present a success rate of 100% with SBCP, thus matching the results presented in [Ton+18a]. However, we can note a lower robustness score for LEAS-P1 and LEAS-P2, which we explain by the difference in height of the root configurations along the guide path that tends to be lower on LEAS with an average root height of $z = 0.86$ meters from the ground compared to RB-Kino that keeps a constant height $z = 0.95$ all along the trajectory. On the rubble scenario, lower configurations present a lesser robustness score but do not impact the success rate.

We can conclude that the rubble scenario is easily solved by the contact planner for any guide path generated by our steering methods. In the future, it could be interesting to focus more on the guide discretization step that is on average equal to $\Delta D \approx 10$ cm here. In this scenario, learning with LEAS how to generate guide paths with a higher ΔD and successful with SBCP, could improve the computation time as well as the quality of the contact plans.

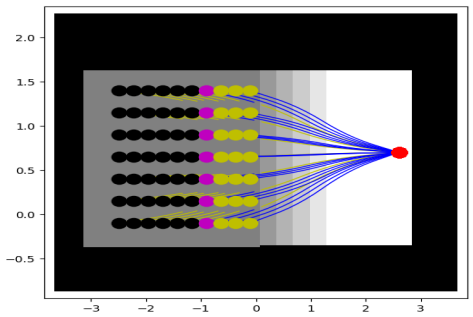
Scenario C2 - Stairs. We present a comparison of RB-Kino, RB-Kino(+RRT), LEAS-P1, and LEAS-P2 on a wide range of initial configurations for the stairs scenario (Figure 4.14). We evaluate our steering methods with SBCP on the stairs for the climbing up task only, as in our experiments the climbing down task was presenting similar results and limitations.

As presented in Figure 4.15a, RB-Kino succeeds to generate valid guide paths only when placed directly in front of the stairs and where further initial configurations lead to configurations unable to reach the ground (invalid with $\mathcal{R}$). For a broader comparison, we plug RB-Kino into a path planning algorithm (RRT) to compute valid guide paths from all previously failed configurations by RB-Kino. We can note in Figure 4.15b that the intermediate waypoints found to solve this simple scenario are far from efficient.

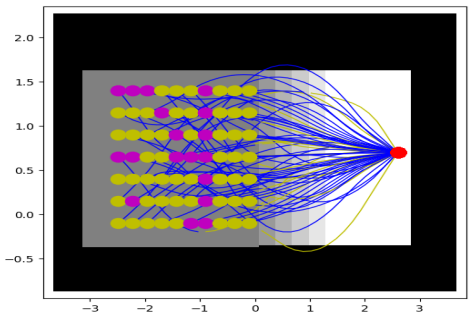
Results in Table 4.3 show that the success of RB-Kino+RRT suffers from a limitation, also hinted by with RB-Kino without RRT, where guide paths with configurations at the limit of the reachability condition $\mathcal{R}$ can lead to failed contact plans with SBCP (magenta dots in Figure 4.15). Some contact configurations resulting from a too high and too low guide path are shown in Figure 4.16, where no transition is feasible from the actual root position to the next. In order to avoid guide paths at the limit of the reachability condition, RB-Kino requires a manual selection of the initial configuration or manually added waypoints at the bottom of the stairs to succeed with SBCP.

In comparison, the steering methods LEAS-P1 and LEAS-P2 succeed without path planning to reach the target and to compute a contact sequence with SBCP from all initial configurations. An advantage of using deep RL to learn such a steering method is that the policy is encouraged to stay far from the critical states (near the reachability condition limits) and thus avoids implicitly difficult and blocking configurations with our contact planner. We denote the same correlation between the robustness score and the success rate (Table 4.3) as our previous test scenarios, and how LEAS-P2 learns to generate guide paths fitting SBCP, leading to more robust contact plans than LEAS-P1 and RB-Kino.

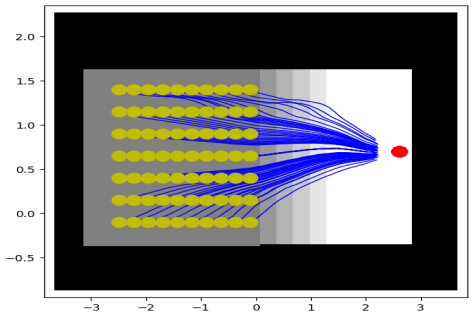
To better evaluate what makes a good guide path for this contact planner on the stairs scenario, we average all the guide paths generated on stairs and render the averaged trajectory on the vertical plane (Figure 4.17). As previously hinted, results show that RB-Kino+RRT tends to engage the stairs with a higher root height and to climb with a lower root position compared to LEAS-P1 and LEAS-P2, thus leading to difficult contact configurations (Figure 4.16). On the other hand, LEAS-P1 and LEAS-P2 tend to keep the root of the robot at a constant distance from the ground and produce some more robust contact plans. A comparison of the robustness score between LEAS-P1 and LEAS-P2 is complex as the difference between their trajectories is subtle. However, we can notice that LEAS-P1 and LEAS-P2 tend to keep a constant distance from the ground, on average 90 cm and 94 cm respectively (maximum distance root-ground with Talos is 105 cm), which explains the difference in robustness.



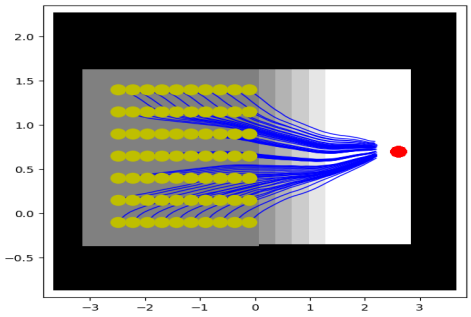
(a) RB-Kino



(b) RB-Kino + RRT

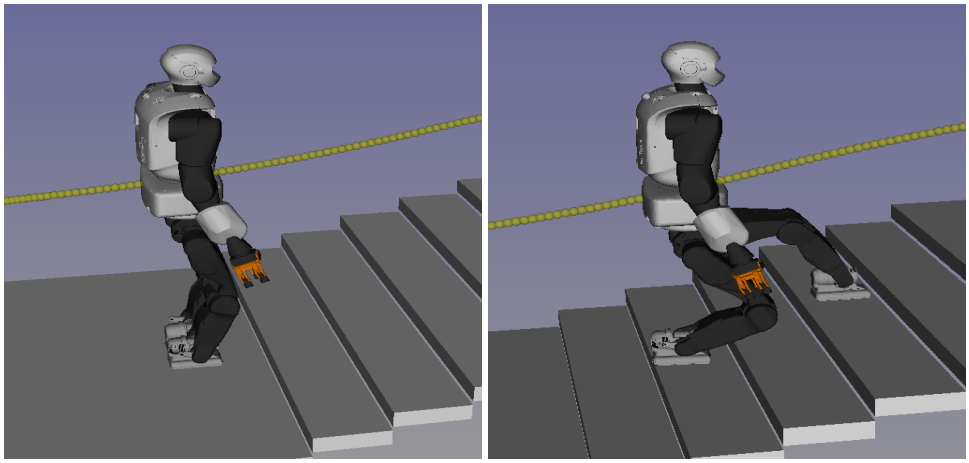


(c) LEAS-P1



(d) LEAS-P2

Figure 4.15: Scenario C2: initial configuration with guide path (yellow dot) valid and successful with SBCP, (magenta dot) valid but failing with SBCP, (black dot) invalid and (red dot) the target.



(a) (b)

Figure 4.16: Scenario C2: Difficult root configuration on the guide for SBCP: (a) too high where the robot has its legs in extension and can not reach the next root position, (b) too low where the robot climbs the stairs crouched.

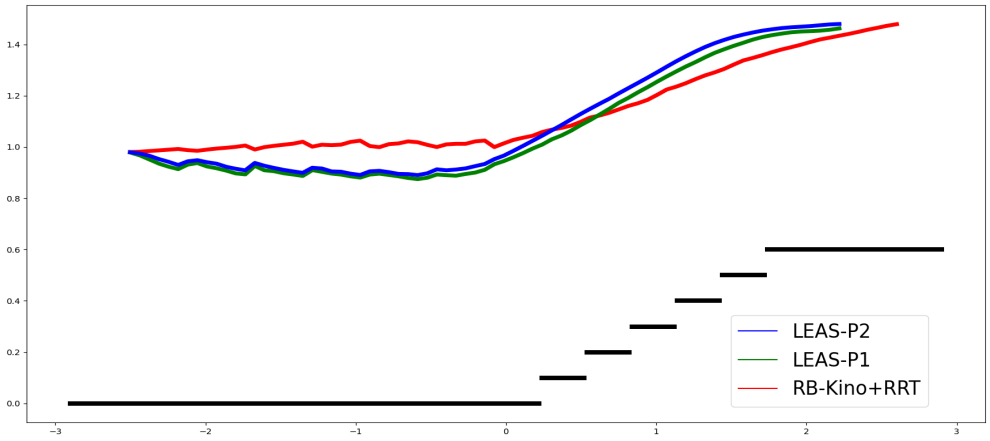


Figure 4.17: Stairs scenario: guide path comparison on z-axis generated by LEAS-P1, LEAS-P2 and RB-Kino+RRT.

4.4 Discussion

4.4.1 Implementation Choices

Figure 4.18: Without reward on the orientation, LEAS-P2 learns that the sidewalking strategy is the most robust and successful with SBCP.

Constraint on the orientation. In LEAS design, the reward R_{ori} penalizes the agent for not orienting the robot toward the goal. As discussed previously, this design choice can be limiting in cluttered environments where sidewalking is required, but we decided it was necessary.

In our first design, we initially trained LEAS-P2 without this reward on the orientation, expecting it to learn that a straight walk is the best strategy (i.e. moving while being oriented toward the goal). However, the result was as shown in Figure 4.18, where LEAS discovered that the most successful approach to plan contacts was by sidewalking. Experiments on our other terrains led to the same result. From a higher level perspective, this strategy makes sense as humans do it too, to walk on complex terrains where they require more stability (e.g. crossing a narrow bridge or climbing down stiff stairs).

However, sidewalking is not the behavior we desire for our biped robot locomotion task and as a consequence, we decided to penalize it on LEAS with R_{ori}. We tuned the associated weight w_{ori} to balance this penalty and the behavior learned to succeed in the contact planning, as previously shown on the wall scenario B2 (4.13b) where the robot sidewalks to avoid the collision and walk straight on the rest of path. However, this reward design may not be suitable for other scenarios and has to be explored in more detail.

Guide discretization for LEAS. With the desired velocity $v_{desired}$ and the timestep T set in LEAS, we obtain an average discretization step (i.e. distance between each root configuration) on the guide of $\Delta D = 2$cm. As shown in section 4.1.4, such a low value presents the best success rate with our contact planner at the cost of a high number of steps in the contact plan, and so a high computation time. We aim for a value $\Delta D = 10$cm for the guide paths in the input of SBCP, that offers a suitable trade-off. Such value is easily reached in RB-Kino and RB-Lin, which generate guide paths as a function of the time $G = f(t)$, by increasing the timestep T. LEAS could also directly achieve such a discretization step with the same strategy. However, a higher timestep leads to actions with a bigger impact on the system, more probability to meet a critical state and so, a less stable learning overall. Another point to consider is the impact of such change on the reward, and so the navigation results presented in the previous chapter and the need to retrain a new policy for this new timestep value.

That is why we opted for a less intrusive method by further discretizing the guide generated by LEAS, keeping 1 out of 5 five root configurations on it, resulting in $\Delta D = 2 \times 5 = 10$cm. This enabled us to use the same policy LEAS-P1 for our comparisons, trained in the previous chapter, and to retrain a new policy from scratch, LEAS-P2, on the same reward basis but adding the contact planning validation ($P2$).

A question one could ask is if a fixed discretization step ΔD along the guide is desirable. Indeed, we saw that the tuning of this value presents a trade-off on our contact planner where it can be pertinent to set a small ΔD for complex sections of the guide path to increase its success (e.g. egress scenario where going out of the car required a value $\Delta D = 3$ cm in [Ton+18a]), or adjust it to a higher value on easier sections where the contact planner is almost sure to succeed (e.g. walking on flat ground with $\Delta D = 15$ cm). On this contact planner, we made the choice to aim for a fixed discretization step $\Delta D = 10$ cm to balance computation time and success rate. However, having a variable discretization depending on the terrain is the most pertinent extension of this work in terms of computation and quality of contact plan with SBCP. This problem will be further investigated in Chapter 5 on a different contact planner.

4.4.2 Learning Strategy

To learn how to generate guide paths fitting the contact planner, one could ask if the strategy of pruning the failing part of the trajectory is the best and if some alternatives exist. We thought of several reward designs to learn from the feedback of SBCP:

Sparse Reward. The agent has to learn from an environment rarely providing him a signal reward (e.g. 1 if the goal is reached, 0 otherwise for all the other steps, -1 if invalid configuration). This solution requires less reward engineering but represents a real challenge for RL where random exploration rarely results in success. That is why it requires additional strategies as learning from additional goals [And+17; Rie+18] or from demonstration [Vec+17; Raj+17]. In our work, this method limits LEAS to basic scenarios, e.g. crossing one *transition* tile (stairs, rubbles, or obstacle), which is not only what we desire for our navigation task. However, we can use such a sparse reward design to perform a sanity check, to verify that LEAS has the capabilities to generate guide paths with improved success rates with SBCP, for a given scenario. We performed such sanity checks on the stairs and rubbles scenarios, pretraining the policy with behavioral cloning (implemented in Stable Baseline [Hil+18]) from trajectories generated by LEAS-P1, to guide the exploration. We then fine-tuned the pretrained policy with the RL algorithm PPO in the real environment with the validation by SBCP. Once again, this resulted in a fast sidewalking toward the goal, thus verifying what as been discussed about the constraint on the orientation: side walking is the most robust way to walk with our contact planner, even for high discretization steps values $\Delta D > 15$ cm.

Robustness as a reward. As seen in the results, there is a correlation between the robustness score and the success rate of the contact planner. That is why it could be pertinent to retropropagate the measure of robustness during the training. This solution is complex to implement as it requires further reward engineering to balance it with the other rewards related to the navigation task. We saw that LEAS-P2 implicitly learned to generate more robust configurations without being specified, but it would be interesting in the future to see if such a measure could furter improve its results.

Pretraining with LEAS-P1. In the previous chapter, we trained LEAS-P1 without a contact planner for a pure navigation task for 10 million steps in 2 hours. In this chapter, we retrained from scratch LEAS-P2 to also learn how to improve the guide feasibility by the contact planner. LEAS-P2 was trained for 12 million steps in 8 hours which is almost four times more than LEAS-P1 due to the contact planning computation. The major criticism we could do about the training is that we did not use the knowledge gained by LEAS-P1 to help the training of LEAS-P2. We used pretraining in our tests with the sparse reward design, to guide the agent exploration. We can in the same fashion pretrain LEAS with continuous rewards, pretraining the policy with trajectories of LEAS-P1 and fine-tuning it in the real environment with the contact planner to obtain LEAS-P2.

However, such pretraining from demonstrations can lead to the *catastrophic forgetting* problem, losing the previously learned experience and starting to relearn from scratch during the fine-tuning, even worsening the results compared to learning from scratch. This problem has been alleviated somehow in recent works [Vec+17; Raj+17] that needs to be investigated, and especially learning from non-expert demonstration [CDT18; Zho+19] that could potentially improve the learning of LEAS-P2.

4.4.3 Conclusion

We used our RL steering method LEAS to properly takes into account the feasibility of our sample-based contact planner. This contact planner ($P2$) generates configurations in contact with the terrain following exactly a root trajectory given in input by the guide path planner ($P1$).

As discussed in our literature review, such decomposition lowers the complexity of the path planning problem but suffers from the non-guarantee of the feasibility of the guide with the contact planner. In this chapter, we have experimentally emphasized such phenomena through the success rates of our previous methods, RB-Kino and RB-Lin with SBCP. We have shown that the basic training introduced in Chapter 3, LEAS-P1, that did not used a contact planner but only the approximated validity constraints ($\tilde{\mathcal{R}}^*$ and $\tilde{\mathcal{C}}$), despite its high success rate on most of our test scenarios, still suffers from this feasibility problem, especially on the rotation and wall scenarios.

In contrast, LEAS-P2 learns how to generate guide paths directly fitting the contact planner, reaching a near 100% success rate in most of our scenarios. Learning this steering method by reinforcement answers the question "*What is a feasible guide path for this contact planner?*" and thus solves the limitation of previous motion-before-contact strategies. We then analyzed the solutions of LEAS-P2 for these scenarios and developed additional constraints to approximate the feasible region of SBCP.

In this work, it is important to note that our strategy with LEAS-P2 was sufficient to solve the feasibility of the guide with this contact planner used as a black box. Our experiments provide further insight into why our methods succeeded. Yet, it suffers from the limitations inherited from our contact planner ($P2$), which we know do not systematically leads to valid whole-body locomotion trajectories ($P3$). In the following chapter, we will investigate the generalization of LEAS to more advanced planners.

Chapter 5

LEAS with Mixed-Integer Programming Contact Planners

Contents

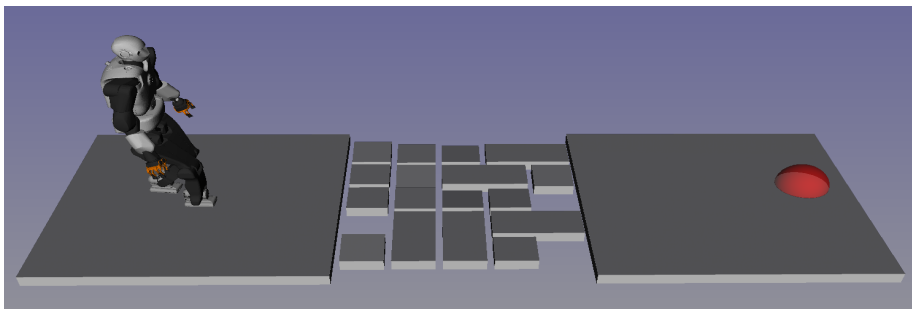


Figure 5.1: Given the desired number of steps n and the set of candidate surfaces $\mathcal{S}$ (grey), the contact planner has to select the surfaces the robot has to step on, along with its foot placements on it to reach the objective area (red).

In this chapter, we will explore the training of LEAS using two contact planners formulated with a Mixed-Integer Programming (MIP) approach. The main objective is to find possible extension of the previous method that would lead to valid contact sequences, using contact planners

that consider better the robot capabilities to connect two successive contacts with the whole-body movement ($P3$).

The sampling contact planner presented in the previous chapter relies on heuristics to search for contacts in the robot environment. While it can efficiently solve the contact planning problem, its sampling approach cannot guarantee its completeness. Optimization-based approaches are an appealing approach to solve this limitation. However, they have to handle both the discrete choice of contact surfaces and the continuous optimization of placing contacts on them. Previous works on contact planning [DT14; Son+20] solved this problem using a MIP approach or its relaxed form respectively. We will explain both approaches and further investigate if our method LEAS can generate feasible guide path by them. Specifically, we will answer the question: *how can LEAS improve the MIP-based optimization of contact sequences through guide path generation?*

This chapter is organized as follows: Section 5.1 presents the notations used in this chapter. Section 5.2 is an overview of the mixed-integer contact planner. We will explain the advantage of using a guide path in this formulation, along with the issues it raises. We then show how LEAS can learn to alleviate these limitations. Section 5.3 explains the reformulation of the MIP into a feasibility linear program, called SL1M [Ton+20]. We will present further insight into this reformulation, as well as the experiments conducted with LEAS to improve its convergence. This insight will be used to give further research directions to improve the behavior of LEAS trained with SL1M.

5.1 Notations

Across this chapter, we use similar notations to the previous work [Son+20].

Notations	Description
n	number of planned footsteps
m	number of terrain contact surfaces
m_i	number of terrain contact surfaces for i-th footstep
$\mathcal{S}$	union of potential contact surfaces available
$\mathcal{S}^j \subset \mathcal{S}$	j-th contact surface
$\mathcal{S}_i \subset \mathcal{S}$	subset of m_i contact surfaces considered in i-th footstep
$\mathcal{S}_i^j \subset \mathcal{S}_i$	j-th candidate contact surface in i-th footstep
p_i	i-th footstep position
r_i	i-th footstep orientation
a_i^j	integer slack variable for j-th surface in i-th footstep
α_i^j	positive real slack variable for j-th surface in i-th footstep
β_i^j	real slack variable for j-th surface in i-th footstep
$\mathcal{I}, \mathcal{G}$	initial and goal constraint sets
$\mathcal{F}$	feasibility constraint set
q	virtual robot root configuration in $SE(3)$
H	local height map around the robot to get surface candidates

The terrain is represented as the union of m surfaces $\mathcal{S} = \bigcup_{j=1}^{m} \mathcal{S}^j$ (Figure 5.1). Each contact surface $\mathcal{S}^j$ is a convex polygon in a 3D plane. For any foot position $\mathrm{p} \in \mathbb{R}^3$, we have:

$$\mathrm{p} \in \mathcal{S}^j \iff \mathrm{p}^\intercal \mathrm{d}^j = e^j \ \wedge \ S^j \mathrm{p} \leq s_j \tag{5.1}$$

Where $\mathrm{d}^j \in \mathbb{R}^3$ is the normal of surface $\mathcal{S}^j$, and $e^j \in \mathbb{R}$. The constant matrix $S^j \in \mathbb{R}^{h \times 3}$ and the vector $s^j \in \mathbb{R}^h$ define the h half-spaces that bounds the surface $\mathcal{S}^j$.

For simplicity, we denote $\mathcal{F}$ the set of dynamic and kinematic feasibility constraints. They guarantee the robot to follow the footstep plan in equilibrium without violating joint limits (Appendix B.1). In this work, we will formulate the initial constraint as $\mathcal{I} : \{\mathrm{P}, \mathrm{p}_1 = \mathrm{p}_\mathcal{I}\}$, and the goal constraint as $\mathcal{G} : \{\mathrm{P}, \mathrm{p}_1 \in \mathcal{S}^\mathcal{G}\}$, where $\mathrm{p}_\mathcal{I}$ is a constant and $\mathcal{S}^\mathcal{G}$ the destination surface.

5.2 Surface Selection and Contact Planning Problem

5.2.1 Mixed-Integer Optimization

We formulate the simplified contact planning problem as follows:

$$\begin{aligned} \textbf{find} \quad & \mathrm{P} = [\mathrm{p}_1, ..., \mathrm{p}_n],\ \mathrm{p}_i \in \mathbb{R}^3 \\ & \mathrm{R} = [\mathrm{r}_1, ..., \mathrm{r}_n],\ \mathrm{r}_i \in \mathbb{R}^3 \\ \textbf{min} \quad & l(\mathrm{P}, \mathrm{R}) \\ \textbf{s.t.} \quad & \mathrm{P} \in \mathcal{I} \cap \mathcal{G} \cap \mathcal{F} \\ & \mathrm{p}_i \in \mathcal{S} \quad \forall i,\ 1 \leq i \leq n \end{aligned} \tag{5.2}$$

We want to find a user-defined number n of footsteps positions p_i and orientations r_i, that minimizes an objective l(P,R). The sequence P and R must satisfy some initial and goal conditions, $\mathcal{I}$ and $\mathcal{G}$, as well as the set of kinematic and dynamic feasibility constraints $\mathcal{F}$. Finally, all foot positions p_i must lie on a surface in $\mathcal{S}$.

As the condition $\mathrm{p}_i \in \mathcal{S}$ is represented by inequality constraints(5.1), we can rewrite it as a Linear Programming (LP) problem using slack variables and the big M method [Lof]:

$$\begin{aligned} \textbf{find} \quad & \mathrm{p}_i \in \mathbb{R}^3 \\ & \mathrm{a}_i = [a_i^1, ..., a_i^m],\ a_i^j \in \{0, 1\} \\ & \beta_i = [\beta_i^1, ..., \beta_i^m],\ \beta_i^j \in \mathbb{R} \\ \textbf{s.t.} \quad & \mathrm{card}(\mathrm{a}_i) = m - 1 \qquad (5.3) \\ & \forall j \in \{1, .., m\} : \\ & \quad S^j \mathrm{p}_i \leq s^j + M a_i^j \mathbf{1} \qquad (5.4) \\ & \quad (\mathrm{p}_i)^\intercal \mathbf{d}^j = e^j + \beta_i^j \\ & \quad ||\beta_i^j||_1 \leq M a_i^j \end{aligned}$$

Where $\mathbf{1}$ is a vector of appropriate size filled with ones. We introduce the slack variables a_i^j and β_i^j:

- If $a_i^j = 0$, the corresponding surface $\mathcal{S}^j$ is selected and the i-th foot position p_i lies on it, thus implying that $\beta_i^j = 0$.
- If $a_i^j = 1$, the constraints relative to the surface $\mathcal{S}^j$ always have a solution and can be ignored.

The big-M method introduces a sufficiently large scalar M to solve our problem, but small enough to not hinder its convergence [Rub]. Indeed, for $a_i^j = 1$ and a sufficiently high M value the constraint (5.4) is always true, and conversely for $a_i^j = 0$ where the feet position p_i must lies in $\mathcal{S}^j$ to be true. The cardinality function (5.3) counts the number of non-zero entries in a vector, hence enforcing each footstep position p_i to lie on exactly one surface in $\mathcal{S}$. As a result, the big-M method and the cardinality condition permit us to represent a logical **XOR** operator for the surface selection problem.

Contact-before-motion formulation. We present the mixed-integer formulation for contact planning without guide (contact-before-motion) from [Son+20], originally introduced by Deits et

al. [DT14]:

$$
\begin{aligned}
\textbf{find} \quad & \mathrm{P} = [\mathrm{p}_1, ..., \mathrm{p}_n],\ \mathrm{p}_i \in \mathbb{R}^3 \\
& \mathrm{R} = [\mathrm{r}_1, ..., \mathrm{r}_n],\ \mathrm{r}_i \in \mathbb{R}^3 \\
& \mathrm{A} = [\mathrm{a}_1, ..., \mathrm{a}_n],\ \mathrm{a}_i \in \{0,1\}^m \\
& \beta = [\beta_1, ..., \beta_n],\ \beta_i \in \mathbb{R}^m \\
\textbf{min} \quad & l(\mathrm{P}, \mathrm{R}) \\
\textbf{s.t.} \quad & \{\mathrm{P}, \mathrm{R}\} \in \mathcal{I} \cap \mathcal{G} \cap \mathcal{F} \\
& \mathrm{p}_n \in \mathcal{S}^{goal} \\
& \forall i \in \{1, .., n\} : \\
& \quad \mathrm{card}(\mathrm{a}_i) = m - 1 \\
& \quad \forall j \in \{1, .., m\} : \\
& \quad\quad S^j \mathrm{p}_i \leq s^j + M a_i^j \mathbf{1} \\
& \quad\quad (\mathrm{p}_i)^\intercal \mathbf{d}^j = e^j + \beta_i^j \\
& \quad\quad ||\beta_i^j||_1 \leq M a_i^j
\end{aligned}
\tag{5.5}
$$

The cardinality constraints (5.5) enforce that exactly one surface $\mathcal{S}^j \subset \mathcal{S}$ is selected for each step. We then consider the problem as solved if all footsteps and cardinality constraints are respected.

Such a formulation can then be solved using a classical MIP approach, that is the LP-based branch-and-bound algorithm to handle the combinatorics (here the discrete surface selection) [Gurb]. Finally, it is important to note that state-of-the-art MIP solvers, such as Gurobi [Gura], have implemented additional methods to improve the solving efficiency such as presolvers [Ach+20], cutting planes [Bal+96] and various heuristics.

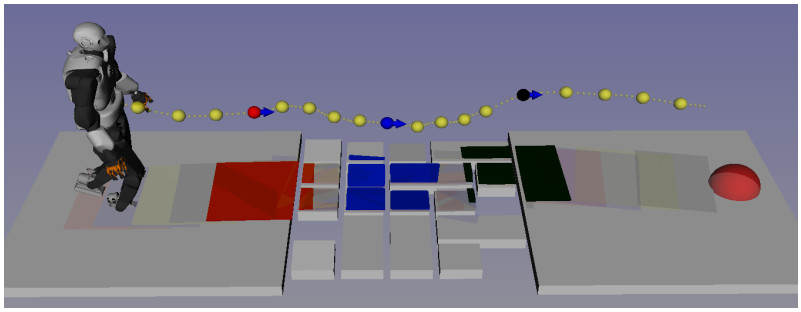
Contact-before-motion formulation analysis. This formulation can explore all terrain contact surfaces in $\mathcal{S}$ while optimizing footstep positions and orientations on them. As a result, it offers a guarantee of completeness under the problem constraints.

However, in most scenarios such as Figure 5.1, we cannot know other than empirically how many steps n are required to reach the distant objective. The tuning of the number of steps n is avoided in [DT14] by defining a maximum bound for the required number of steps. The MIP problem is then solved with a cost encouraging the robot to reach the goal with the minimum number of steps. Once done, the redundant footsteps in the goal area are then discarded. A critical limitation of this approach is the manual tuning of the maximum number of steps. On the one hand, overestimating this number results in unnecessary computation due to the number of redundant footsteps. On the other hand, underestimating it can make the problem infeasible.

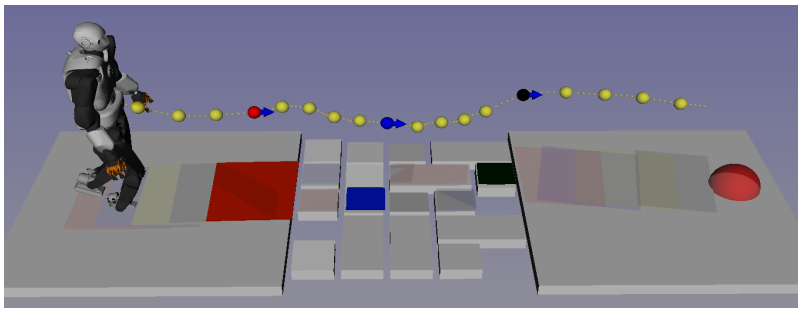
Overall, as the number of steps n and candidate surfaces m increases, so do exponentially the dimension and complexity of the problem (n^m). As a consequence, this method can result in contact planning times up to several seconds for a few steps.

5.2.2 Motion-before-contact Approach

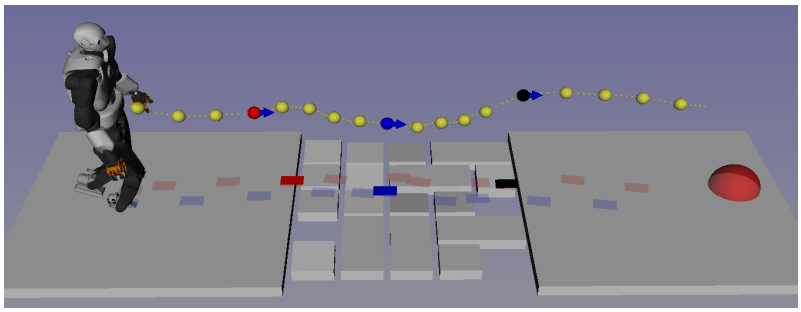
To fix the limitation of tuning the number of steps n, and preprocess the search footstep orientations R, hence reduce the problem complexity, we can reformulate it with a motion-before-contact approach, i.e. using a guide path.



(a) Guide path planning and candidate surfaces



(b) Contact surface selection



(c) Contact placement

Figure 5.2: MIP with guide path: (a) the steering method plans a guide and gets a reduced set of candidate surfaces $\mathcal{S}_i$, represented by the colored patches, one for each discretized configuration q_i along the guide, (b) the contact planner then selects the surfaces to step on, and (c) it places the contacts on it. We highlight three configurations in red, blue and black for better visualization.

Motion-before-contact formulation. We explain the mixed-integer approach from Song et al. [Son+20] using a guide path. The formulation is as follows:

$$
\begin{aligned}
\textbf{given} \quad & \mathrm{R} = [\mathrm{r}_1, ..., \mathrm{r}_n],\ \mathrm{r}_i \in \mathbb{R}^3 && (5.6)\\
& \mathcal{X} = [\mathcal{S}_1, ..., \mathcal{S}_n] && (5.7)\\
\textbf{find} \quad & \mathrm{P} = [\mathrm{p}_1, ..., \mathrm{p}_n],\ \mathrm{p}_i \in \mathbb{R}^{3\times n}\\
& \mathrm{A} = [\mathrm{a}_1, ..., \mathrm{a}_n],\ \mathrm{a}_i \in \{0,1\}^{m_i}\\
& \beta = [\beta_1, ..., \beta_n],\ \beta_i \in \mathbb{R}^{m_i}\\
\textbf{min} \quad & l(\mathrm{P}, \mathrm{R})\\
\textbf{s.t.} \quad & \{\mathrm{P}, \mathrm{R}\} \in \mathcal{I} \cap \mathcal{G} \cap \mathcal{F}\\
& \forall i \in \{1, .., n\} :\\
& \quad \mathrm{card}(\mathrm{a}_i) = m_i - 1\\
& \quad \forall j \in \{1, .., m_i\} : && (5.8)\\
& \quad\quad S_i^j \mathrm{p}_i \leq s_i^j + M a_i^j \mathbf{1}\\
& \quad\quad (\mathrm{p}_i)^\intercal \mathbf{d}^j = e^j + \beta_i^j\\
& \quad\quad ||\beta_i^j||_1 \leq M a_i^j
\end{aligned}
$$

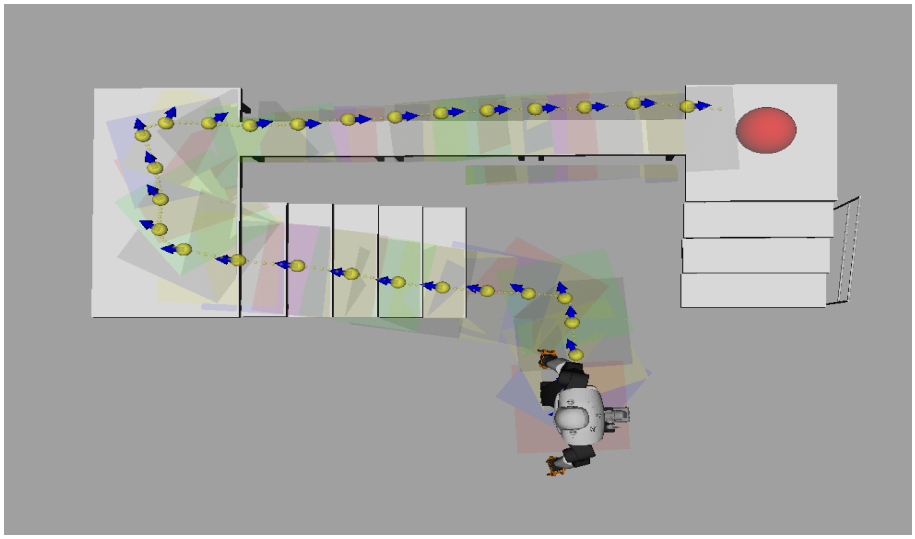


Figure 5.3: Guide path for surface selection: (yellow) robot root configurations discretized on the guide path, (blue) orientation of the root, (colored rectangles) the candidate surfaces around the root.

Simplification of the problem. As shown in Figures 5.2 and 5.3, each discretized robot root configuration q_i along the guide contains several information. As a result, they can be used to simplify the previous formulation:

- The number of steps n can be deduced from the guide path. Indeed, the guide paths can be discretized to obtain a desired discretization ΔD between each root configuration q_i. Contrary to the contact planner presented in the previous chapter, one step is made with a cyclic gait for each discretized root configuration along the guide.
- The footstep orientations can follow the robot root orientations (blue arrows in Figure 5.3). With this approach, the rotation sequence R can directly be given as input (5.6), thus drastically reducing the problem complexity.
- The candidate surfaces for each step $\mathcal{S}_i$ can be pruned to constrain the search space around a discrete root configuration along the guide. As a result, we can give as input the sequence $\mathcal{X}$, in which we associate to each footstep their corresponding candidate surfaces subset $\mathcal{S}_i$ (5.7). Each subset contains m_i surfaces with $m_i \leq m$. As a result, the number of candidate surfaces explored in the problem is smaller (5.8), as well as the number of slack variables in a_i and β_i. It is important to note that if the i-th step has only one surface candidate ($m_i = 1$), the corresponding slack variable value can be fixed: $a_i^1 = 0$.

Previous results. In the previous work [Son+20], Song et al. use RB-Kino with RRT [Fer+17] presented in the previous chapter to generate guide paths. They then demonstrate the clear computation time advantage of using a guide in this formulation. Furthermore, they accelerate the contact planning by solving separately a feasibility problem (i.e. without $l(\mathrm{P}, \mathrm{R})$) to obtain the surfaces to step on, then optimize the footstep placements on them with a quadratic cost (i.e. with $l(\mathrm{P}, \mathrm{R})$). As a result, their algorithm can plan a few steps in tens of milliseconds using the commercial solver Gurobi [Gura].

We reproduce their results on our long scenario composed of rubbles, a bridge, and three stairs accounting for a total of $m = 56$ surfaces (Figure 5.4). We perform both approaches: contact-before motion that considers all terrain surfaces as potential candidates for each step ($m_i = m$), and motion-before-contact that prunes non-relevant contact surfaces ($\overline{m_i} = 4$). We average the results over 20 tests, with a sufficiently high number of steps to ensure a successful contact planning.

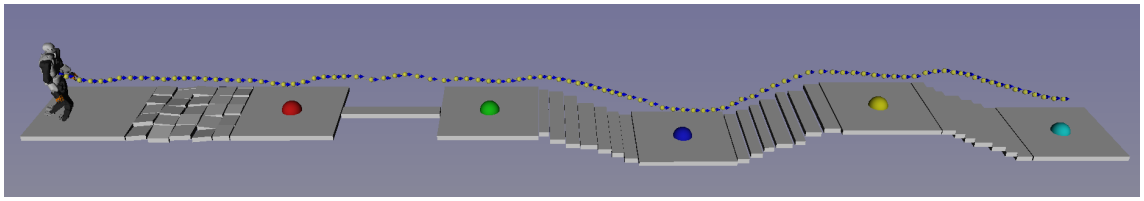


Figure 5.4: Long range scenario: the robot is tested on trajectories of different lengths from its actual position to any colored waypoint. The guide is represented by the yellow trajectory.

Table 5.1: Comparison between contact-before-motion (without guide) and motion-before-contact (with guide) on our long-range scenario for different path lengths (Figure 5.4).

Waypoints	red	green	blue	yellow	cyan
number of steps n	18	36	56	74	94
w/o guide computation time (ms)	330	3400	4700	27200	31000
w/ guide computation time (ms)	30	75	120	218	292

Results in Table 5.1 show the clear advantage of the motion-before-contact approach over its counterpart, ranging from 10 times to 100 times faster as the number of steps increases. We did not analyze the computation time of the guide as we did not yet fully optimize LEAS implementation, written in Python. At the moment, it corresponds to 15 ms per step with most of the resources allocated to retrieving the local height map. From our results, it still results in better performance when added to the contact planning time, with in the worst case (red): $18 \times 15 + 30 = 300$ ms that is still inferior to the 330 ms contact planning time without guide.

Problem statement. Two key aspects are to be considered with the MIP contact planner: the number of steps n and the number of surface candidates for each step m_i. As we have seen in the previous results, the problem complexity exponentially grows with both parameters. This limitation is alleviated by the use of the guide path to prune non-relevant candidate surfaces for each step.

In the previous scenario, we manually tuned the discretization steps, which we will refer to as ΔD_i between each step, and indirectly the number of footsteps along the guide to be feasible by the contact planner. However, finding the right discretization strategy highly depends on the difficulty of the scenario. Indeed, a low traversability terrain trivially requires a high number of steps to reach the objective, and conversely easy terrains such as flat ground.

As hinted by our previous experiments, always using sufficiently high discretization steps could enable us to reduce the number of steps in the problem, hence further alleviating its combinatorics and so its computation time. Following this idea, we formulate the question: *Can we automatically compute guide paths with sufficiently high discretization steps depending on the terrain, while being feasible by the contact planner?* We previously discussed this question in Chapter 4, where we ask if a fixed average discretization step along the guide is desirable in terms of contact planning quality and complexity. We chose to discretize the guide with a fixed desired value $\Delta D = 10$ cm to alleviate the heavy computation load of our sampling-based contact planner. Here, we investigate this question with the MIP contact planner.

5.2.3 Implementation Details

Our goal is to adapt the discretization steps ΔD_i along guide paths depending on the traversed terrain. One footstep is performed for each of the n discretized configurations q_i along the guide. Such a guide must then produce feasible problems for the MIP contact planner. To do so, we use our steering method LEAS to generate guide paths by adapting its velocity, hence the ΔD_i along, depending on its local terrain observations.

In this work, we use Gurobi solver [Gura] with its presolver and heuristics to solve the MIP formulation for contact planning (Section 5.2.2).

RL policy. We employ the same network architecture, actions, rewards, and hyperparameters as described in chapter 3. The number of asynchronous workers computing contacts with the MIP planner is set to 6. To help LEAS estimate the terrain traversability, we add a scalar to the states that represents the number of reachable surfaces around the robot. Its utility will be further discussed later on.

Our method LEAS takes as input the local height map, the direction to the goal, and the number of surfaces around the robot to locally navigate the terrain. The methodology is similar to Chapter 4. States generated by LEAS are subject to some reachability and collision-free conditions, $\tilde{\mathcal{R}}^*$ and $\tilde{\mathcal{C}}$ respectively (see Chapter 3), plus the additional constraint that is to succeed the guide path with the MIP contact planner.

Candidate surfaces. We obtain the candidate surfaces S_i directly from the reduced local height map H_i around the robot configurations q_i (colored rectangles in Figure 5.3). We choose this method as we can easily get such a height map with our implementation, but any other method could be applied.

It is important to note that compared to the sampling contact planner of the previous section, the MIP planner generates contact sequences that do not follow exactly the guide. While the root trajectory could be used as a part of the cost to optimize, in this work, it is solely used to collect candidate surfaces along it.

That is why opted for small sized height map H_i (80x80 cm). Indeed, it may lower the number of candidate surfaces and so the problem feasibility. However, it also constrains the footstep placements in the guide vicinity, as well as lowers the problem complexity by reducing the number of candidate surfaces m_i. We will further discuss this implementation choice and its alternatives in Section 5.2.5.

Discretization of the guide. The discretization of the guide generated by LEAS corresponds to an average of $\Delta\overline{D} = 2$ cm between each configuration (See Chapter 3).

With the MIP contact planner, a footstep is made for each discretized configuration along the guide. Our goal is to get sufficiently high discretization steps ΔD_i between each configuration along the guide, to lower the contact planning complexity while ensuring its feasibility.

In this chapter, we aim for a maximum $\Delta D_i = 28$ cm. This value corresponds to the near maximum step length that the Talos robot could perform. To do so, we further discretize the guide in input of our MIP contact planner by only keeping 1 out of $N_{ref} = 14$ configurations on it. Depending on the terrain, LEAS may not be able to generate feasible guide paths with this number (i.e. navigate the terrain at the desired velocity). As a result, LEAS has to learn how to adapt the robot root velocity and indirectly ΔD_i along the guide to succeed in contact planning.

Contact planning choices. In this thesis, we focus on solving a feasibility problem without quadratic cost, i.e. $l(\mathrm{P}, \mathrm{R}) = 0$. This implementation choice is motivated by the fact that we want to avoid the high computation time spent by the MIP solver on optimizing this cost. However,

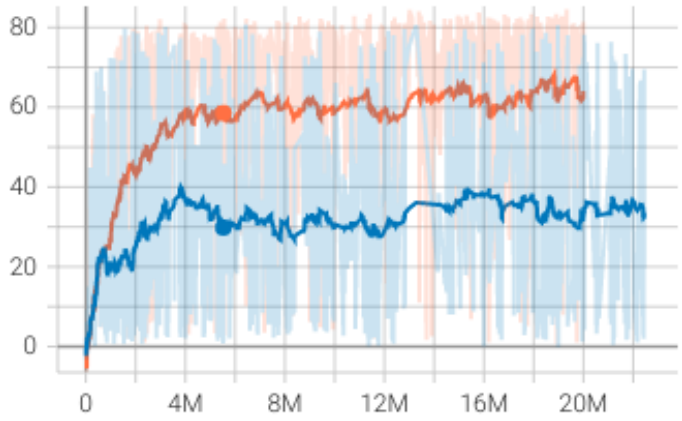


Figure 5.5: Learning curves of (red) LEAS-P1 trained without contact planner and (blue) LEAS-P2 trained with feedback from the MIP contact planner.

once one surface is assigned to each footstep, a new problem with this cost can be solved with a quadratic solver, but at the cost of optimality.

The MIP formulation selects contact surfaces and optimizes contact placements on it for all footsteps simultaneously. As a consequence, the contact planner cannot return the last successful step along the guide, contrary to the sampling-based contact planner of the previous section. To train LEAS with feedback from the contact planner as described in Chapter 3, two different methods can be used.

The first strategy is a Dichotomic Search to obtain the last successful step on the guide path. It is a long-horizon approach, and so guarantees the completeness of the contact planning up to the last successful step.

The second strategy is to plan contacts in a model-predictive control fashion such as [Ris+22]. To do so, they generate a short guide path, then solve the contact planning for n steps (with n a small number). They then keep the first step generated and repeat the process. The algorithm stops when the goal is reached or the contact planning fails. This strategy is a short-horizon approach, and thus alleviates the combinatorics aspect of the problem. However, it is prone to local minima. This limitation is avoided in their implementation by solving the problem with a regularization cost in $l(\mathrm{P}, \mathrm{R})$, but at the cost of a longer computation time than a feasibility problem.

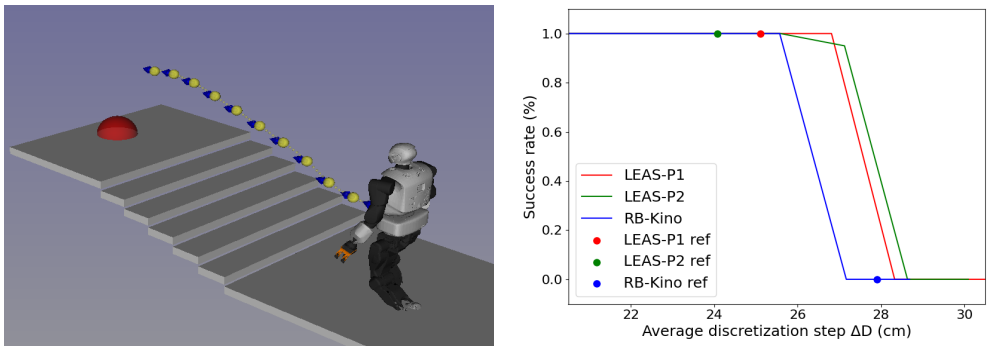
In this work, we want to avoid adding such costs for computation efficiency purposes. Moreover, short-horizon planning cannot guarantee completeness, which is not desirable for our task. That is why we choose the dichotomic search strategy to solve the long-horizon contact planning problem along the whole guide path.

Training. We train LEAS with feedback from the MIP contact planner on the training terrain presented in Chapter 3. The model is evaluated after 13 million steps corresponding to 8 hours of training on a PC with an Intel Core i7-8700 (12 cores, 3.20Ghz, 16GB ram).

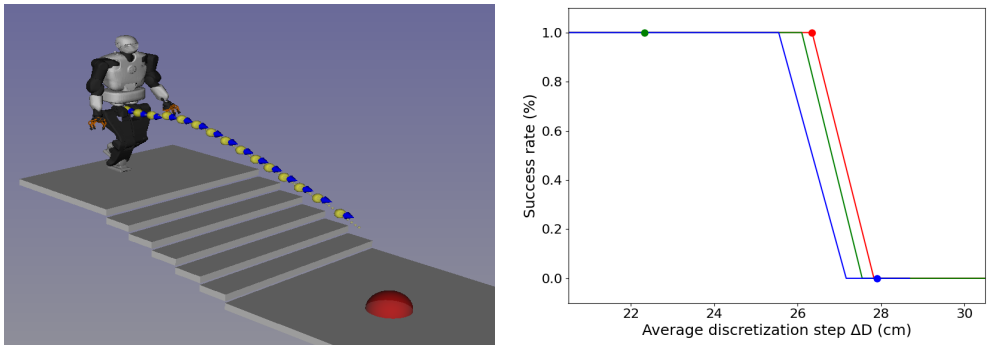
Learning curves of LEAS without a contact planner, referred to as LEAS-P1, and LEAS-P2 with the MIP contact planner are shown in Figure 5.5 (the maximum reward is equal to 100 for trajectories in a straight line). The episode rewards of LEAS-P2 are lower than LEAS-P1 due to the contact planning constraints. As described earlier, we discretize the guide so that when moving at the desired velocity $v_{desired} = 0.10$ m/s, the discretization step is equal to $\Delta D = 28cm$. This value is at the limit of the feasibility constraints $\mathcal{F}$ of the TALOS robot on flat ground (Appendix B.1). Consequently, our steering method has to move the robot root slower to compute feasible guides by the contact planner, thus resulting in a lower average reward.

5.2.4 LEAS Results

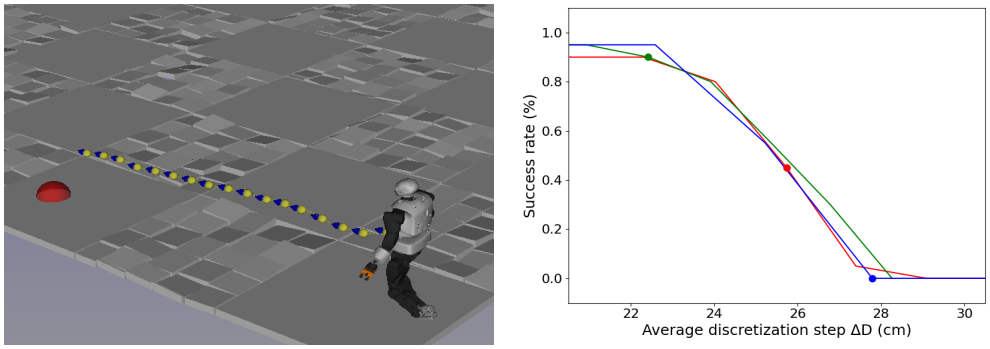
We expect LEAS-P2 to find, depending on the terrain, some sufficiently high discretization steps ΔD_i along the guide with the parameter N_{ref}, that are feasible by the contact planner. On difficult terrains such as rubbles and stairs, LEAS should lower the robot root velocity, and thus ΔD_i, to ensure the contact planning success. On contrary, we expect the robot to move with a higher velocity on easy ones such as flat ground.



(a) Stairs (up)



(b) Stairs (down)



(c) Rubbles

Figure 5.6: Comparison of the steering methods success with MIP contact planning for different discretization steps. Colored dots correspond to results with the value N_{ref} used across all scenarios.

Basic scenarios. We first evaluate the impact of the average $\Delta\overline{D}$ value along the guide on the MIP contact planning success. It enables us to evaluate the difficulty of our basic scenarios (stairs and rubbles in Figure 5.6).

We compare the results of LEAS-P1, LEAS-P2, and RB-Kino. For each scenario, the initial robot root is oriented toward the goal with a velocity $v_{init} = 0.05$ m/s. Each steering method acts

on the root velocity to potentially reach the desired velocity $v_{desired} = 0.10$ m/s, corresponding to discretization steps at the edge of the feasibility constraints.

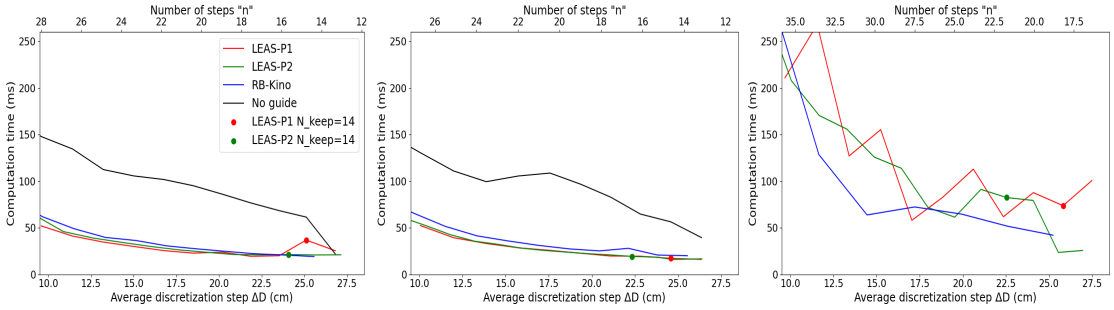
We test our steering methods for different N values, keeping 1 out of N configurations along the guide ($N \in \{6, 20\}$), which covers a wide range of discretization steps. Indeed, increasing the N value implies a higher average $\Delta\overline{D}$ along the guide and inversely. Each N value is tested on 30 trajectories per scenario. We represent the result of the steering methods for $N = N_{ref}$ by some colored dots (the value LEAS-P2 has been trained on).

We first observe the success rate of the steering methods depending on their average discretization step. On both stairs scenarios, results show that the contact planning is always a success for all $\Delta\overline{D} \leq 27cm$ (Figures 5.6a and 5.6b). However, it is not the case for the rubble scenario that is more complex (Figure 5.6c). Indeed, depending on the reduced set of candidate surfaces for each step, no solution may be found satisfying the equilibrium constraints, and thus the problem may become infeasible.

LEAS-P1 is trained to follow the reference velocity $v_{desired}$. However, it rarely reaches this velocity in complex scenarios, thus resulting in an average $\Delta\overline{D} \approx 26$ cm with N_{ref}. RB-Kino is tuned to keep an average constant velocity $v_{desired}$ all along the guide. As a result, its discretization is mostly uniform along it with $\delta\overline{D} \approx 28$ cm.

With N_{ref}, RB-Kino fails our 3 scenarios. This result is expected as its average discretization step draws near (or out) the limits of feasibility constraints. As a consequence, RB-Kino requires a fine-tuning on the guide discretization adapted to each scenario. In comparison, LEAS-P1 navigates the terrain slower and a value N_{ref} is sufficient for the stairs scenarios. However, this value does not hold anymore on the rubbles (only 50% of success). Consequently, LEAS-P1 also requires a fine-tuning of its discretization to generate feasible guide paths in this scenario.

Our solution LEAS-P2 on the other hand succeeds in all scenarios for N_{ref} it has been trained on (green dots). Results show that our method always adopts a sufficiently high discretization step to succeed in the MIP contact planning along the guides. As a result, it does not require further fine-tuning. However, we can observe that LEAS-P2 considers the stairs (down) scenario as difficult. This behavior may require further investigation as it could be due to our training arena that contains much steeper stairs (requiring a low velocity to succeed in contact planning), or the terrain representation on the height map that will be discussed later on.



(a) stairs (up) (b) stairs (down) (c) rubbles

Figure 5.7: Contact planning time on guide paths of different average discretization steps. Stairs (a) and (b) contains few surfaces, contrary to the rubbles (c).

In Figure 5.7, we compare the computation times of each steering method depending on the total number of steps to compute. On the stairs up and down scenarios (number of surfaces $m = 7$), they are also compared to the MIP contact planner without a guide. We do not compute it on the rubble scenario as it results in a memory overflow due to the number of surfaces on the terrain (m=520). While a high number of steps (i.e. small discretization steps) increases the contact planning success, it also increases its computation time. As previously discussed, the computation grows exponentially with the number of candidate surfaces as observed on the rubbles ($\overline{m_i} = 6$).

On contrary, the contact planning time remains relatively low on our stairs scenarios for the tested discretization step range, with less than 100 ms when using a guide ($\overline{m_i} = 2.5$). As we can expect, LEAS-P2 for N_{ref} always exhibits an average discretization step with a low computation time, while being successful with the MIP contact planner.

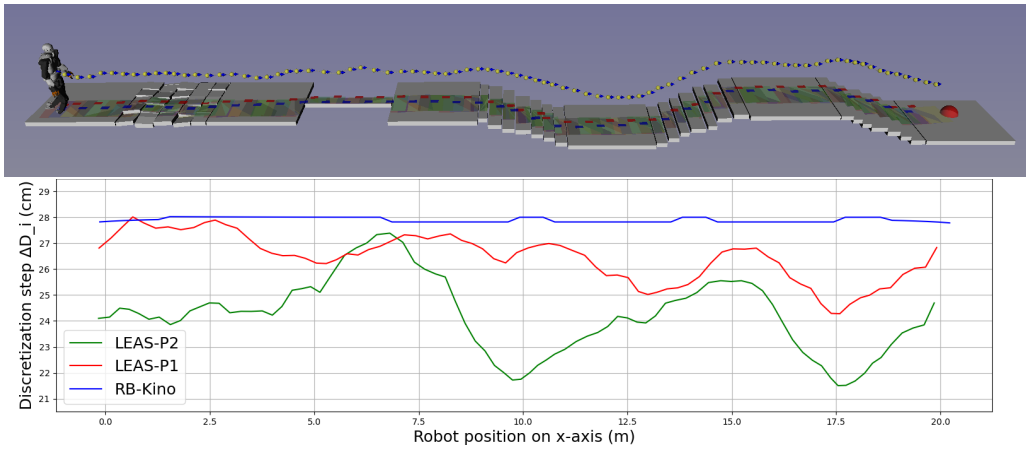


Figure 5.8: Example on our long scenario. For each steering method, we plot the discretization steps along the guide path for N_{ref}. The x-axis matches the terrain pictured above. The ΔD value of LEAS-P2 correlates with the terrain difficulty.

Long scenario. We compare the steering methods on our long-range scenario (Figure 5.8). Here, we will further focus on the impact of adapting the discretization ΔD_i between each root configuration step on a sequence of easy and difficult terrains.

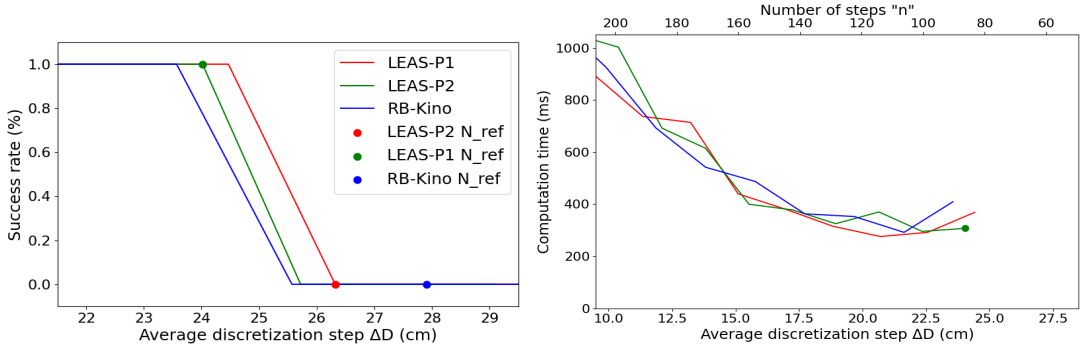
The steering methods have to generate a guide path up to the red objective, then succeed in the MIP contact planning. As RB-Kino cannot generate valid guide paths up to the goal without a path planner, we manually place waypoints on the terrain. This scenario is particularly difficult because the steering methods should lower the discretization steps depending on the terrain difficulty. In that regard, we render the different ΔD_i values between each step along the guide for N_{ref}. The x-axis matches the terrain pictured so that we can observe their behavior in each terrain area.

We tuned RB-Kino to have an average discretization step of $\Delta\overline{D} \approx 28$ cm along the guide. Just like in the basic scenarios, LEAS-P1 presents an average discretization step $\Delta\overline{D} \approx 26$ cm. As both steering methods do not plan guides in function of the terrain difficulty, they both fail the contact planning on our long-range scenario for N_{ref} (Figure 5.9a).

On the other hand, our solution LEAS-P2 adapts the discretization step for each terrain traversed (Figure 5.8), which correlates with the results of the basic scenarios. We can observe that it exhibits smaller discretization steps on terrains it deemed difficult (rubbles and stairs down), and higher ones on the others (bridge, stairs up, and ground floor). Thanks to its adaptability, LEAS-P2 can succeed in the MIP contact planning for N_{ref} in our long scenario (Figure 5.9a), whereas other methods require further fine-tuning of the guide discretization.

Figure 5.9a shows the success rate of the steering methods depending on their average discretization steps. We observe that this scenario is feasible by all three steering methods for $\Delta\overline{D} \leq 24$ cm. Just like in the basic scenarios, LEAS-P2 can generate feasible guide paths by the contact planner while exhibiting a sufficiently high discretization between each step. In this scenario, LEAS-P2 thus automatically finds a suitable discretization strategy to succeed in the contact planning while reducing the overall number of steps.

We will now observe the impact of the number of steps, and the gains we can obtain through this optimization (Figure 5.9b). Our long range is scenario is feasible for a number of steps $n \geq 90$. We expected the contact planning time to be lower as the number of steps converges to this minimum bound. Using Gurobi solver along with its presolver and heuristics, we first observe that



(a) (b)

Figure 5.9: Long scenario for different discretization steps, MIP contact planning (a) success rate and (b) computation time on guide paths of different average discretization steps.

it is complex to analyze. While it is true that the very high number of steps ($n = 200$) results in an exponential increase, there is no clear benefits in reducing this number from $n = 130$ to $n = 90$ where we observe similar computation time. In our experiments, we notice a great disparity between the problem results of similar length. Indeed, Gurobi heuristics are effective but unpredictable in their performance, hence making difficult the evaluation of the problem complexity. Further tests thus need to be performed on other MIP solvers.

5.2.5 Conclusion and Discussion

Conclusion. We have shown that the motion-before-contact strategy drastically reduces the MIP contact planning time.

Exploring a lead from the previous work [Son+20], we presented a solution to automatize the guide path discretization using our solution LEAS. Our steering method learns to adapt the robot root velocity along the guide depending on the terrain difficulty. As a result, it can generate feasible problems for the MIP contact planner while reducing the overall number of steps, hence the problem complexity.

Traversability estimation. As discussed in Chapter 3, tuning the upper and lower bounds of the z-value in the height map is critical to have a correct terrain representation. Trivially, setting wider bounds will permit better detection of large obstacles, or differentiate stairs and void. However, it hinders the detection of small height variations on the terrain such as rubbles. Several strategies could be explored to fix such problems, such as using convolutional neural networks as previously discussed, or giving as observation a second tighter bounded height map.

In this thesis, we keep the height map bounds used in previous chapters and we add to the number of potential candidate surfaces around the robot as an observable state. This state permits LEAS to better detect difficult terrains, however, it is still unclear how this parameter impacts the agent decision and additional ablation tests are required. In the future, other methods could also be explored to estimate the terrain traversability [LB18; BFH19].

Contact planning awareness and guide discretization. We discretize the guide path in input of the contact planner by keeping 1 out of N_{ref} configurations. However, our RL agent does not know which state corresponds to a step. As a consequence, our steering method only adapts the robot velocity along the guide depending on the terrain, and implicitly the discretization steps ΔD_i. In another experiment, we added another observable scalar representing a counter before the next step to be performed as well as the distance and local height map from the previous step. However, it did not improve our results.

Another option is for LEAS to output directly the next root configuration q from which a contact is to be made. This approach can remove the need to filter the guide path ($N = 1$). Hence, this control could potentially permit the agent to directly control the discretization step and to accurately select suitable candidate surfaces $\mathcal{S}_i$. In our experiments, we implemented such an approach by raising the timestep T and controlling LEAS in velocity (instead of acceleration). But as discussed in Chapter 4, it was not achievable in practice regarding the learning stability, because of the large actions the RL agent does.

Pruning candidates surfaces. In this thesis, we obtain the candidate surfaces for each step from a height map centered around the robot root configuration. While the height map size could be enlarged to offer more possibilities and thus increase the problem feasibility. We choose such a small size to constrain the search for contacts only in the guide path vicinity.

Other strategies are available to obtain the candidate surfaces $\mathcal{S}_i$. In the previous work [Son+20], Song et al. use for that the range of motion of the robot legs. If an intersection exists between this range of motion at q_i and a surface $\mathcal{S}^j$, then the entire surface is added to the set $\mathcal{S}_i$. Adding the complete reachable surfaces $\mathcal{S}^j$ sure increases the feasibility of the problem. However, it may not constrain enough the search for footsteps around the guide, which can be a limitation depending on the desired locomotion task.

Reinforcement learning and combinatorics. We emphasize the fact that the MIP contact planner solves the discrete choice of contact surfaces, which is a combinatorial problem. Our steering method adapts the robot velocity along the guide path depending on the terrain, that then composes the contact planning problem to solve. However, LEAS cannot observe the past trajectory. As a consequence, it is not directly (or not at all) aware of its combinatorial aspect.

As discussed in the previous work [Son+20], pruning surfaces along the guide can be seen as a MIP presolve routine (cuts) specific to contact planning. This strategy often leads to small feasibility problems to be solved with only a few to no exploration of the combinatorics when using a presolver [Ach+20]. However, this is not the case for more complex scenarios in which contact planning time increases due to the computation of the heuristics or the branch-and-bound algorithms that handles the combinatorial aspect of the problem (i.e. selection of surfaces).

While we demonstrate the efficacy of the motion-before-contact approach with Gurobi solver [Gura], the contact planning time highly depends on the MIP solvers used as well as their various heuristics employed. Moreover, state-of-the-art MIP solvers can be quite large in terms of memory size which limits their embedding on real robots. Due to the highly combinatorial aspect of the surface selection problem, an interesting idea comes naturally to us: *Can we reformulate the problem and remove its combinatorial aspect?*

5.3 Reformulation of a Feasibility Problem: SL1M

As presented in the previous section, the performance of the MIP contact planner is heavily impacted by the combinatorial aspect of the problem.

In [Ton+20; Son+20], a reformulation called SL1M (Sparse L1-norm Minimization) is proposed. Its objective is to take advantage of the sparsity-inducing properties of the $l1$-norm, hence breaking the combinatorics of the surface selection problem as we will explain later in this section. Such a promising solution could remove the need for branch-and-bounds algorithms, and thus MIP solvers, replacing them with simple linear solvers easily embeddable on the robot.

5.3.1 Relaxation of the Mixed-Integer Problem

The relaxed formulation of the MIP formulation for contact planning with guide path [Son+20] is as follows:

$$
\begin{aligned}
\textbf{given} \quad & R = [\mathrm{r}_1, ..., \mathrm{r}_n],\ \mathrm{r}_i \in \mathbb{R}^3 \\
& \mathcal{X} = [\mathcal{S}_0, ..., \mathcal{S}_n] \\
\textbf{find} \quad & \mathrm{P} = [\mathrm{p}_1, ..., \mathrm{p}_n],\ \mathrm{p}_i \in \mathbb{R}^3 \\
& \alpha = [\alpha_1, ..., \alpha_n],\ \alpha_i \in \mathbb{R}_+^{m_i} \qquad (5.9) \\
& \beta = [\beta_1, ..., \beta_n],\ \beta_i \in \mathbb{R}^{m_i} \\
\textbf{min} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m_i} \alpha_i^j \qquad (5.10) \\
\textbf{s.t.} \quad & \{\mathrm{P}, \mathrm{R}\} \in \mathcal{I} \cap \mathcal{G} \cap \mathcal{F} \\
& \forall i \in \{1, .., n\} : \\
& \quad \forall j \in \{1, .., m_i\} : \\
& \quad\quad S_i^j \mathrm{p}_i \leq s_i^j + M \alpha_i^j \mathbf{1} \\
& \quad\quad (\mathrm{p}_i)^\intercal \mathbf{d}^j = e^j + \beta_i^j \\
& \quad\quad ||\beta_i^j||_1 \leq M \alpha_i^j
\end{aligned}
$$

Reformulation. Contrary to the MIP formulation, the integrality constraints on the slack variables are relaxed (5.9). As a result, the problem can be solved with a linear or quadratic solver.

The most important change is the reformulation of the feasibility problem into a cardinality minimization problem. The cardinality constraints of the MIP formulation are thus replaced by a $l1$-norm minimization (5.10). Indeed, $l1$-norm minimization has long been used in optimization problems to induce sparsity (i.e. encourage the convergence of the slack variables to 0) [BV04]. In this formulation for contact planning, having an α_i^j with a 0 value means that the candidate surface $\mathcal{S}_i^j$ has been selected for the i-th footstep. Simply said, the $l1$-norm encourages the surface selection for each step. In the next Section 5.3.2, we will provide further insight on the $l1$-norm to visually understand how it encourages this sparsity in the contact planning context.

Solution to the relaxed problem. The problem is directly solved if exactly one surface is selected for each footstep:

$$
\forall i \in \{1, ..., n\},\ \exists! j \in \{1, ..., m\},\ \alpha_i^j = 0 \qquad (5.11)
$$

If the problem is not solved after this relaxation, as is usually the case, several strategies are available to enforce this condition.

As previously discussed, a branch-and-bound algorithm can efficiently solve this problem. However, it requires the use of a heavy MIP solver.

Another approach is to explore combinatorics with heuristics. In [Son+20], Song et al. fix all the slack variables α_i for which the cardinality constraint is satisfied (i.e. $\alpha_i = 0$). They then test all the combinations for the remaining free variables until either (a) a solution is found, (b) a maximum number of combinatorial explorations is reached, or (c) all possible combinations are exhausted.

As one could guess, to reduce the contact planning time, the problem needs to be solved right after relaxation or with as few combinatorial explorations as possible.

5.3.2 Insight on the Relaxation

To better understand how the $l1$-norm encourages surface selection, we propose several scenarios with a visual understanding of SL1M solutions.

Big-M method and $l1$-norm without constraints. We give the following simplified formulation of our problem:

$$\begin{aligned} \textbf{given} \quad & \mathcal{S}, \mathrm{p} \\ \textbf{find} \quad & \alpha = [\alpha^1, ..., \alpha^m],\ \alpha^j \in \mathbb{R}_+ \\ \textbf{min} \quad & \sum_{j=1}^{m_i} \alpha^j \\ \textbf{s.t.} \quad & \forall j \in \{1, .., m\} : \\ & S^j \mathrm{p} \leq s^j + M\alpha^j \mathbf{1} \end{aligned} \tag{5.12}$$

For each position p in the 2D space, we aim to minimize the $l1$-norm relative to the surfaces $\mathcal{S}^j \subset \mathcal{S}$. Results for different surface samples are shown in Figure 5.10. Black rectangles represent the terrain surfaces $\mathcal{S}^j$. Once the problem is optimized, we understand that each α^j measures a distance between the point position p and the closest edge of surface $\mathcal{S}^j$. We represent on a 2D map the $l1$-norm cost (5.12), i.e. the sum of these distances, where the gradient changes from bright to dark to represent higher and lower cost values respectively.

In the top right figure, we highlight in pink the delimitations between the different gradient cost areas, where the gradient of the $l1$-norm changes in strength or direction. These delimitations show that there always exists a gradient change on the edges of the surfaces, which already hints at how sparsity is encouraged in SL1M.

We now want to understand what is the consequence of such a $l1$-norm property in SL1M contact planning formulation (5.10).

Scenario A. In all following scenarios, we simplify the feasibility constraints $\mathcal{F}$ (See Appendix B.1). We will only use the constraints on the foot positions, represented by the red and yellow rectangles in our scenarios (left and right foot respectively). As a consequence, the foot position p_i is only constrained by the previous foot position p_{i-1}. The rotation r_i is kept constant for all footsteps, so that all foot constraints are oriented in the same direction. We do so to have a better visual understanding of SL1M relative to these simplified constraints.

In both scenarios (Figures 5.11 and 5.12), the robot from its initial right foot position p_1 (yellow dots) has to reach the surface $\mathcal{S}^4$ in two steps, meaning that p_3 can lie anywhere on $\mathcal{S}^4$. For these examples, we arbitrarily assign to the step position p_2 the candidate surfaces of indices 2 and 3 (i.e. $\mathcal{S}^2$ and $\mathcal{S}^3$). The $l1$-norm cost map is only plotted for this step, as it is the only one given multiple surface candidates.

In scenario A.1 (Figure 5.11), the feet positions p_2 and p_3 computed by SL1M relaxation both lies at the edges of their respective constraints. This is expected as we are solving a linear problem using the simplex algorithm, which explores the extremities and the edges of the feasibility constraints. Also, the feet position p_2 is attracted by the $l1$-norm minimum area (dark) between the surfaces $\mathcal{S}^2$ and $\mathcal{S}^3$. Here, the problem is solved after the first relaxation as one surface has been selected for each step.

In scenario A.2 (Figure 5.12), we now increase the distance between surfaces $\mathcal{S}^2$ and $\mathcal{S}^3$. The result shows that the foot position p_2 still lies at the edge of the foot constraint, and is placed in the $l1$-norm minimum area, i.e. with a zero gradient (Figure 5.12a). However, it does not lie on a surface ($\alpha_2^2 > 0$ and $\alpha_2^3 > 0$), so our problem is not solved. As discussed in Section 5.3.1, we can

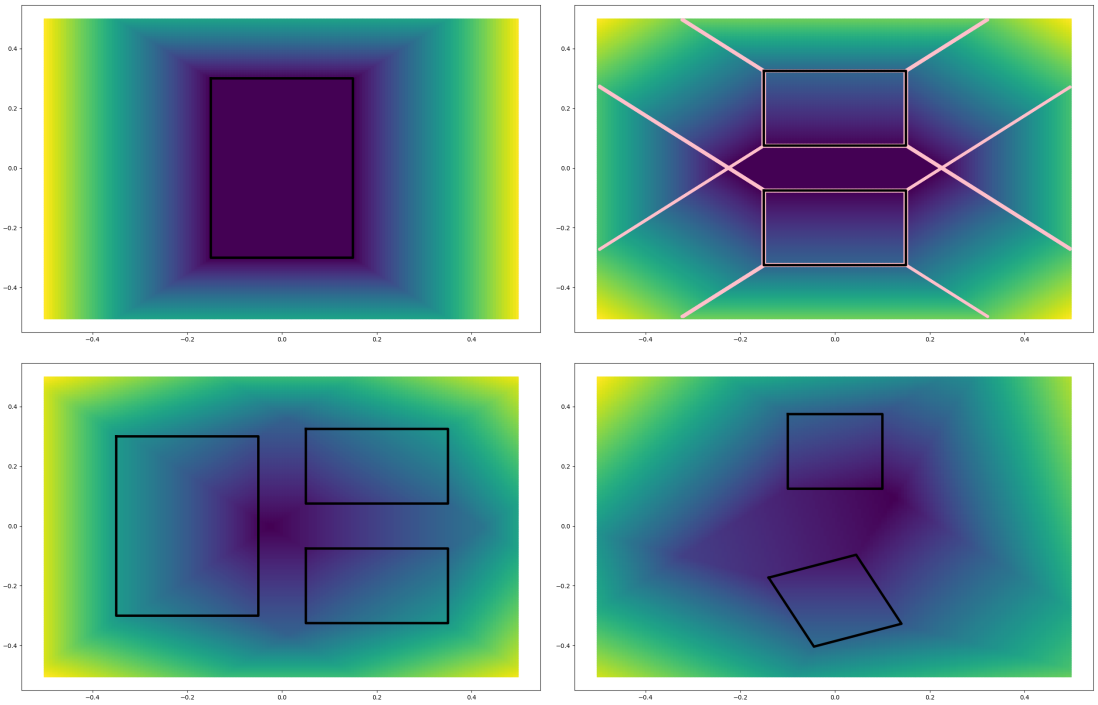


Figure 5.10: Example of $l1$-norm cost (5.12) relative to different surfaces sample in 2D: dark shades represent a low cost and bright shades a high cost. The pink lines in the top right figure delimits the gradient changes.

use some heuristics to explore the combinatorics. The heuristics previously presented generate 2 sub-problems as p_2 has 2 candidate surfaces. The first combination tested is $p_2 \in \mathcal{S}^2$, its closest surface. In this scenario, the heuristics is successful with the first combination (Figure 5.12b).

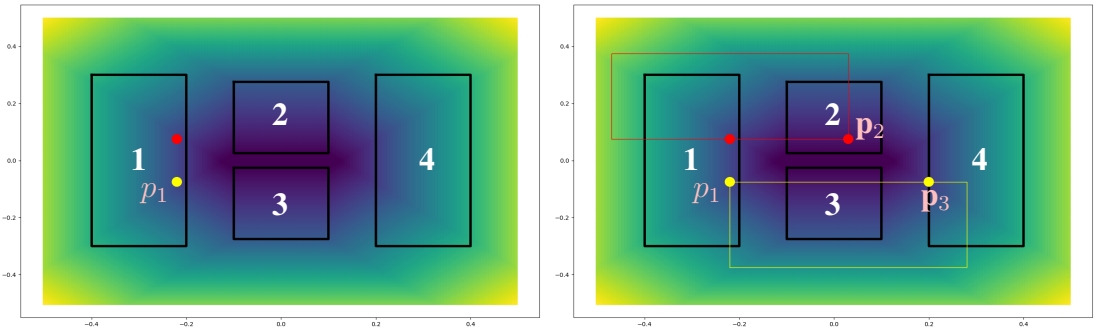
These simple scenarios enable us to observe one of the main limitations of the linear formulation, that is that the extremities of the constraints are always explored first. Consequently, if an extremity already lies inside the minimal cost area but is not on a surface, we irremediably require to explore the combinatorics. However, the contacts are generally planned for a higher number of steps which, as we will see, impacts the relaxed solution.

Scenario B. As shown in figure 5.13, the robot has to cross a complex terrain composed of 10 surfaces and reach the surface $\mathcal{S}^{10}$ in 5 steps. We arbitrarily assign to each step position the following candidate surface indices:

1. $p_2 \Rightarrow j = [1, 2, 3]$
2. $p_3 \Rightarrow j = [4, 5]$
3. $p_4 \Rightarrow j = [4, 5, 6, 7, 8, 9]$
4. $p_5 \Rightarrow j = [4, 5, 6, 7, 8, 9]$
5. $p_6 \Rightarrow j = [10]$

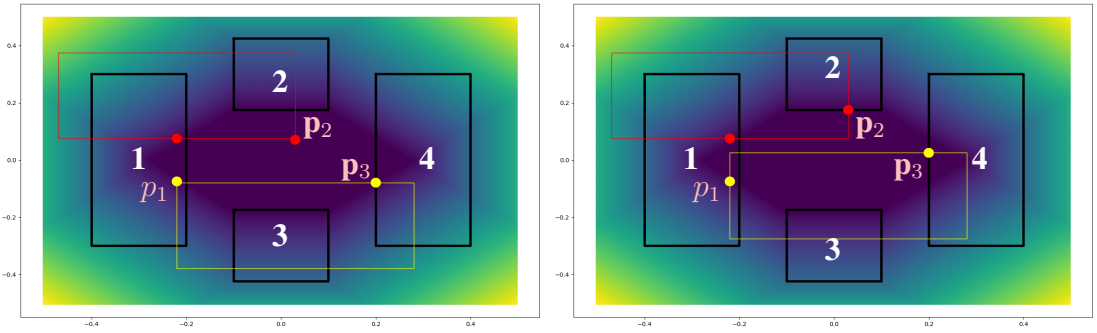
We plot 3 different $l1$-norm cost maps corresponding to the steps $\{p_2\}$, $\{p_3\}$, $\{p_4, p_5\}$ respectively. As we can see, the cost map of $\{p_4, p_5\}$ contains higher values than the others as they have more candidate surfaces. As a consequence, they will be more attracted toward their minimal cost area than the two others, p_2 and p_3 (Figure 5.13a).

In this scenario, SL1M selects some surfaces for $p_2 \in \mathcal{S}^2$, $p_3 \in \mathcal{S}^5$, and obviously $p_6 \in \mathcal{S}^{10}$ that is a hard constraint. However, it does not solve the problem as p_4 and p_5 do not belong to any



(a) Initial foot placements. (b) Relaxed solution.

Figure 5.11: Scenario A.1: (red) left and (yellow) right feet. The robot has to perform two steps to reach surface $\mathcal{S}^4$.



(a) Relaxed solution. (b) With heuristics.

Figure 5.12: Scenario A.2: (a) the robot has to reach $\mathcal{S}^4$ in two steps, (b) SL1M does not solve the problem as p_1 does not lie on a surface, and (c) the heuristics succeeds the sub-problem with $\mathcal{S}^2$, the nearest surface to p_1.

surface. As a consequence, the heuristics generates 36 sub-problems as p_4 and p_5 can potentially lie on 6 different surfaces each. By testing all lowest cost combinations first, the heuristics finds a feasible solution after 5 explorations (Figure 5.13b).

As we can observe, the $l1$-norm encourages sparse solutions, hence the surface selection. However, it does not guarantee that each step lies on a surface. Consequently, SL1M formulation often requires combinatorial explorations to solve the problem.

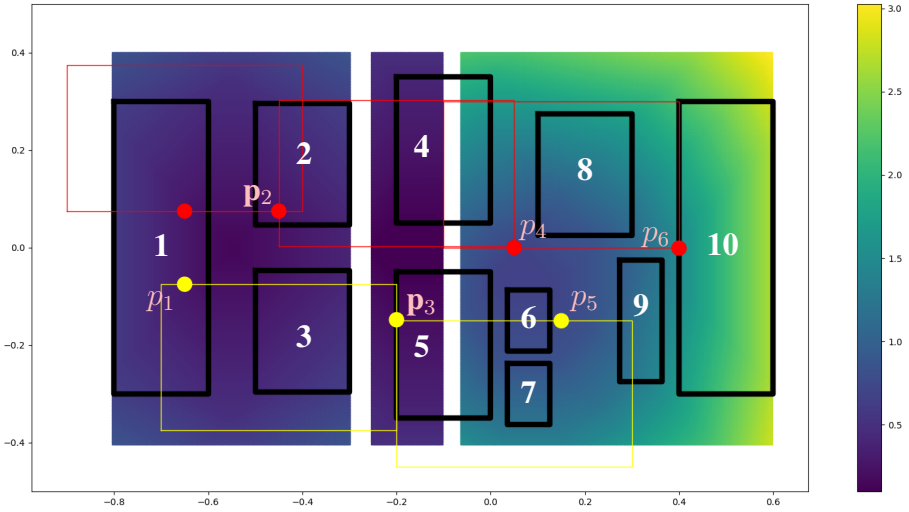
5.3.3 Problem Statement

SL1M formulation is promising to solve the contact planning problem with a simple linear optimization solver. We have seen that the $l1$-norm cost encourages the sparsity among the slack variables α_i, and so the surface selection for each step.

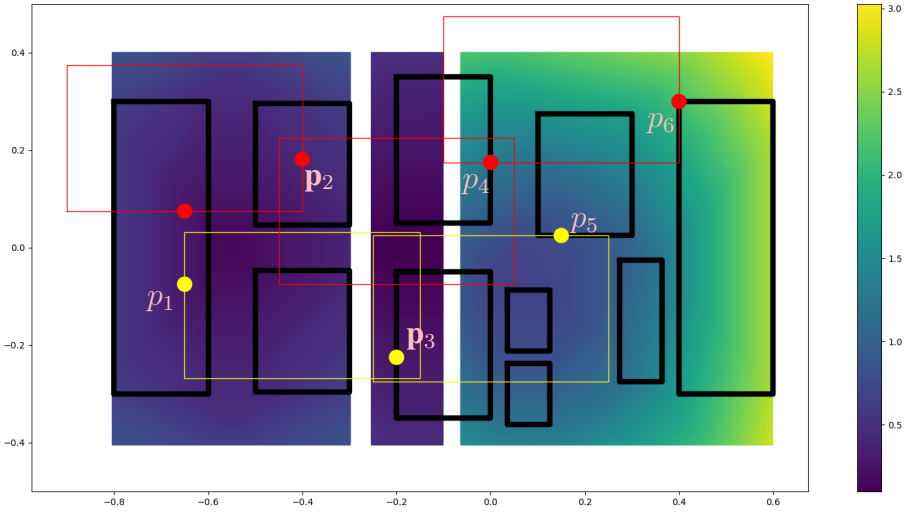
However, we observed that SL1M rarely solves the problem directly, and thus requires the use of heuristics to explore the combinatorics for undecided steps, which grows exponentially with the number of footsteps and surface candidates [Ton+20].

Moreover, the heuristics presently used can lead to unfeasible problems. Indeed, fixing selected surfaces in the SL1M solution further constrains our set of possibilities. While reducing the combinatorics to explore, it can also lead to unfeasible sub-problems. As a consequence, we may lose the guarantee of completeness. At the moment, we do not have better heuristics or solutions other than the branch-and-bound algorithm to fix this limitation.

Another important remark is that the constraints have been simplified in the previous examples.



(a) Relaxed solution.



(b) With heuristics.

Figure 5.13: Scenario B: (a) SL1M does not solve the problem for p_4 and p_5, and (b) the heuristics finds a solution after 5 combinatorial explorations. We plot 3 different $l1$-norm cost maps for $\{\mathrm{p}_2\}$, $\{\mathrm{p}_3\}$ and $\{\mathrm{p}_4, \mathrm{p}_5\}$ respectively.

The full set of kinematic and dynamic constraints $\mathcal{F}$ (Appendix B.1) further reduces the feasibility spaces, which in our experiments makes the combinatorial exploration with the heuristics less likely to succeed.

To fix these limitations, we explore a lead from the previous work [Son+20], that is the impact of the guide path and its discretization on the SL1M problem. In their result, they noticed that its tuning is critical to generate easier SL1M problems. Indeed, as shown in the previous section on MIP, we need sufficiently small discretization steps ΔD for the problem to be feasible. However, SL1M introduces a new dimension to the problem with its formulation regarding the $l1$-norm.

We thus conduct several experiments to answer the following question: *Can we use the guide to help SL1M convergence to feasible sparse solutions?*

5.3.4 Experiments Conducted and Discussion

We aim to increase the success rate of SL1M contact planner, with as little combinatorial exploration as possible using the heuristics. This objective is also related to the work in the previous Section 5.2.3. Indeed, if a problem is not feasible by the MIP contact planner, then it is not feasible by SL1M, its relaxed form. To achieve this objective, we tested the same approaches we had with LEAS and the other contact planners.

Testing our previous approaches. Just as done with the MIP contact planner, we first trained LEAS to follow a desired velocity $v_{desired} = 0.10$ m/s. We first focused on generating feasible problems for SL1M without maximizing the discretization steps along the guide. That is why we aimed for an average discretization step $\Delta\overline{D} = 20$, for which we demonstrated that it was sufficient to cope with the contact planning feasibility constraints $\mathcal{F}$ in all our test scenarios (Section 5.2.4). We also performed a dichotomic search of the last successful step along the guide, feedbacked during the training. We expected LEAS to adapt the robot root velocity, hence the discretization steps along the guide, to generate feasible SL1M problems. Two LEAS versions were trained, (1) with and (2) without SL1M heuristics.

However, this experiment was not successful as both RL policies did not converge. While LEAS trained with the heuristics (1) exhibited slightly higher episode rewards than its counterpart, both (1) and (2) converged to a similar unexpected behavior: the policies were making the robot idle on flat ground to accumulate positive rewards, without crossing any transition tile. Indeed, they deemed the transition tiles (rubbles, stairs, bridge) too difficult to be crossed with the contact planner SL1M for both modes. As a consequence, they accumulated the idle rewards on flat ground ensuring the surface selection with only one candidate surface per step.

Such behavior is understandable knowing the insight from the previous section. Indeed, SL1M contact planning solution is highly sensitive to the number of steps as well as the number and geometry of candidate surfaces for each step along the guide. Our steering method control may not be sufficient to solve this problem. As a consequence, it avoided the difficulty of SL1M contact planning by not performing its navigation task.

Sanity check with a sparse reward. To force LEAS in confronting both the navigation task and SL1M contact planning, we performed a sanity test on the stairs scenario.

To enforce LEAS in reaching a distant objective, we replaced its continuous rewards with a simple sparse reward:

$$R = \begin{cases} 1 & \text{if objective reached with P1 and SL1M} \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

To maximize the episode reward, LEAS was thus forced to succeed in climbing the stairs with the contact planner. As LEAS rarely reaches the objectives with the following reward, the network was pretrained using trajectories generated by LEAS-P1 to guide its exploration. We increased the discount factor γ from 0.97 to 0.9999 to encourage it in reaching the objective while focusing on the contact planning success instead of the number of steps, without modifying the sparse reward.

As SL1M always succeeds in this low combinatorial problem scenario with its heuristics (1), we only trained our steering method without it (2) to better analyze if the guide impacts the relaxation itself.

After training, we observed some improvements in SL1M success rate, but with unexpected behavior. LEAS was making the robot sidewalk at the maximum feasible discretization step (when sidewalking) up to the objective. This result is similar to our previous experiment on the sampling-based contact planner (Chapter 4), where sidewalking is a preferred walking strategy in regard to the robot equilibrium constraints, plus in this case the overall SL1M formulation complexity.

To avoid this undesirable sidewalking behavior, we then forced the robot orientation toward the goal. To do so we added the continuous reward R_{ori} as defined in Chapter 3. After training, the success rate was obviously lower than without orientation enforcement. The policy learned that the best strategy was to reach the target in the least possible number of steps. This strategy makes sense as by reducing the size of the problem, we can potentially reduce the number of footsteps not lying on a surface. Another interesting behavior we observed is that the robot was staying on the edges of the terrain with a non-straight orientation, thus further reducing the number of candidate surfaces and so the problem combinatorics. Other than reducing the problem complexity through these behaviors, we did not oberve any other that could help us better understand how to help SL1M relaxation.

Just as with MIP contact planner, we also tried to add past observable states along the trajectory as well as a counter indicating when the next step is to come. However, it was not successful in capturing the combinatorial complexity of SL1M formulation.

Conclusion and Prospective. We explained the SL1M contact planning formulation whose objective is to remove the combinatorial aspect of the surface selection problem. Following the intuition of the previous work [Son+20], we explored precisely how the guide path helps this relaxed formulation.

Each contact planning problem is characterized by the number of steps and the number and geometry of candidate surfaces for each step. Beyond the combinatorial aspect of the problem as in the MIP planner, SL1M introduces a new dimension to it with the $l1$-norm that is efficient for simple scenarios, but hard to grasp on complex ones.

Our experiments with LEAS have shown that we can impact SL1M solution through the guide path. However, its control over it is quite limited and mostly led to undesirable strategies like sidewalking that reduce the problem combinatorics.

As previously discussed, LEAS can not oberve the past trajectory and candidate surfaces that compose the problem. As a consequence, it is not fully aware of its combinatorial aspect which is critical in SL1M formulation. An interesting approach could be to investigate the use of recurrent neural networks in LEAS to permit an internal representation of this past trajectory.

Arguably, solving SL1M through the guide path may not be the right approach to the problem. Indeed, this problem goes beyond the concepts of reachability or traversability, and could turn out too complex to be handled by a simple navigation task. Consequently, we may have to improve the formulation itself. Many ideas come to our mind, such as adding weights to the slack variables in function of the surface probability to be stepped on. This way, the $l1$-norm minimal area would be more attracted toward these surfaces. However, it is difficult to decide how to assign these weights, and machine learning could be particularly useful for this task. In future work, further exploring how to improve this formulation is an exciting direction. Finally, this also encompasses a broader range of applications not specific to contact planning which all aim to solve more efficiently Mixed-Integer Programming problems.

Chapter 6

Conclusion

This thesis explored a hybrid approach for **legged navigation** in complex terrains. The question of the **path feasibility** was the core of our research. This very broad concept has then been narrowed down to our context of legged robot locomotion, where we defined the path feasibility as follows: *How to plan paths more likely to be extended into valid contact sequences?*

Our work is built on top of a motion-before-contact planner, the Loco3D framework, whose architecture divides the locomotion problem into three sequentially solved sub-problems: ($P1$) is the planning of the robot **guide path** to navigate the terrain; ($P2$) is the **contact planning** along the guide; and ($P3$) is the optimization of the whole-body movement performing the contacts. As discussed in our literature review, we focused on the main bottleneck of this approach, that is the feasibility of the guide paths ($P1$) by a given contact planner ($P2$). We progressed one step beyond the previous work [Ton+15] that introduced the **reachability** condition, necessary but insufficient to fully capture the contact planner capability and its closely related concept of terrain **traversability**.

Research contribution

The core contribution of this thesis to formulate the local steering method of the guide path planner as a **reinforcement learning** optimal policy, called LEAS.

In Chapter 3, we first demonstrated its navigation skills in complex environments, subject to collision avoidance and reachability conditions, hence already providing a ready-to-use navigation method ($P1$).

In Chapters 4 and 5, we have shown how our steering method can address the feasibility issue between ($P1$) and ($P2$). Whereas other works approached the problem via various heuristics or terrain traversability estimation, our contribution is to directly learn what is a feasible path from the main concerned: the contact planner. We have shown that LEAS can learn a navigation task while using the contact planner as a validator.

Our steering method thus answered our research question: *How to learn by reinforcement a navigation task for better contact planning feasibility?* We have explored the answers with our three different contact planning strategies. Our results demonstrated that navigating with only a simple local height map of the terrain was sufficient to approximate a part, if not all, of their feasibility space. Our steering method can be readily used jointly with our sample-based and MIP contact planners, which shows its versatility. Even though the mixed results we had with SL1M (Section 5.3), we have shown that our steering method was able to find some strategies to increase

its success rate within the limit of its controls. Interpreting the strategies learned by LEAS is promising to have further insight into our contact planners and improve them.

Limitations of the study

This thesis employed the contact planners as black boxes with their original formulation. Consequently, several implementations and parameter choices were made relative to their connection to the guide path.

On the sampling-based planner (Chapter 4), we fixed a desired configuration step along the guide to balance its relatively high computation time and its success rate. Consequently, we did not investigate the guide discretization as done with the MIP contact planner. At the moment, the contact planner generates many key contact postures along the guide, that have to be filtered by some manually-tuned heuristics. However, they often present unpredictable performance regarding the quality of the contact plan. That is why directly adopting a sufficiently high discretization step would be desirable to avoid using such heuristics.

The MIP contact planner (Section 5.2) used jointly with a guide is currently our most efficient solution for motion-before-contact planning. While we demonstrated how LEAS can be used to generate feasible and efficiently solved problems, additional tests are required on different MIP solvers to validate its advantages over our other steering methods on the problem complexity.

The relaxed formulation SL1M (Section 5.3) raises a problem mixing both combinatorics with a complex geometrical nature, difficult to handle. Our steering method did not have enough control over the variables that composed it. That is why it could be interesting to investigate what additional actions could be performed to help its convergence. In particular, directly letting the RL agent select suitable candidate surfaces for each step is a promising research direction.

Perspectives

Following our results, we believe our steering method could naturally be extended in two manners. First, we mainly focused on gaited humanoid robot locomotion, but our method could be easily applied to quadruped robots or even non-gaited locomotion. Second, all three contact planners used were long-horizon planners. Yet, it could be interesting to test our approach on short-horizon planners that may be subject to different feasibility criteria, e.g. [Rai86; Ris+22].

Overall, we believe that path feasibility should be learned directly from experience with the robot itself. In this thesis, we focused on the path feasibility by a given contact planner. Extending this concept to the whole-body movement could get us closer to our goal of a fast and safe solution for legged robot locomotion in complex environments.

Appendix A

Reinforcement Learning: Overview

Reinforcement Learning can be defined as an agent interacting with an environment according to a policy π. The interaction sequence can be modeled as a Markov Decision Process (MDP) that is a tuple (S,A,P,R,μ,γ), where:

- S is the state space of the environment.
- A the set of discrete or continuous actions available.
- $P : S \times A \to \Delta S$ is the transition function of the MDP dynamics.
- $R : S \times A \to \mathbb{R}$ is the reward function defining the desired agent behavior.
- μ is the initial state distribution.
- $\gamma \in [0, 1]$ a discount factor.

As the full environment state is often not observable by our agent, we can use partially observable MDP like in this thesis, where our steering method only uses local observations to navigate through the environment.

The agent starts an episode in an initial state s_0 sampled according to μ. Depending on the agent state $s_t \in S$, the objective of the policy π is to make the agent perform an action $a = \pi(s)$ in its environment, while maximizing its future cumulative rewards $J(\pi) = \mathbb{E}[\sum_{k=t}^{\infty} \gamma^k r_{t+k} | \pi]$. We denote the optimal policy $\pi^* = \arg\max_{\pi} J(\pi)$.

Appendix B

Mixed-Integer Programming Formulation details

B.1 Feasibility Constraints.

We denote $\mathcal{F}$ the set of kinematic and dynamic feasibility constraints, as explained in [Ton+20]. It includes some constraints on the robot center of mass for its equilibrium and the reachability of the planned contacts. As a result, they guarantee the feasibility of the robot contacts, which are characterized by their position p and orientation r.

Center of mass constraints We guarantee the equilibrium and balance constraints using the 2PAC formulation [Ton+18b]. These constraints will be succintly explained and we refer the reader to [Ton+20] for further details. Using this formulation, we only need to select 2 Center Of Mass (COM) positions for each phase k from step p_{k-1} to p_k, that are $\mathrm{c}_{k,0}$ and $\mathrm{c}_{k,1}$ to guarantee continuous feasibility (Figure B.1).

In the context of biped walking, a sufficient condition for static equilibrium is to ensure that the center of mass lies above the support effector. The constraint can be formulated as follows:

$$\begin{aligned} F_{k-1}(\mathrm{c}_{k,0} - \mathrm{p}_{k-1}) &\leq f_{k-1} \\ F_k(\mathrm{c}_{k,1} - \mathrm{p}_k) &\leq f_k \end{aligned} \tag{B.1}$$

where F_k and f_k are the matrix and vector defining the foot polygonal shape at position p_k (considering the contact lies on flat ground). Constraints (B.1) depends only on the xy coordinates of the COM. As a result, by convexity of the static equilibrium regions, the straight lines $[\mathrm{c}_{k,0}, \mathrm{c}_{k,1}]$ continuously satisfies the static equilibrium constraint, as well as $[\mathrm{c}_{k-1,1}, \mathrm{c}_{k,0}]$ and $[\mathrm{c}_{k,1}, \mathrm{c}_{k+1,0}]$ as the COM stays above the corresponding support effector.

Reachability constraints. We also use the center of mass positions $\mathrm{c}_{k,0}$ and $\mathrm{c}_{k,1}$ to guarantee kinematic reachability. A 3D polytope $\mathcal{R}$ is obtained for each effector (feet in our case) via offline random sampling, approximating the reachable COM workspace. The resulting polytope is expressed as follows: $\mathcal{R} : \{\mathrm{c} \in \mathbb{R}^3,\ R\mathrm{c} \ \leq\ r\}$, where R and r are the matrix and vector defining the polytope.

For each phase k, we consider the orientation of the foot frame constant and equal to r_{k-1}. We note $\mathcal{R}_k$ the rotated polytope associated with contact p_k. For phase k, the contraints on the COM positions $\mathrm{c}_{k,0}$ and $\mathrm{c}_{k,1}$ can be formulated as follows:

$$R_l(\mathrm{c}_{l,e} - \mathrm{p}_l) \leq r_l \quad \forall l \in \{k-1, k\} \tag{B.2}$$

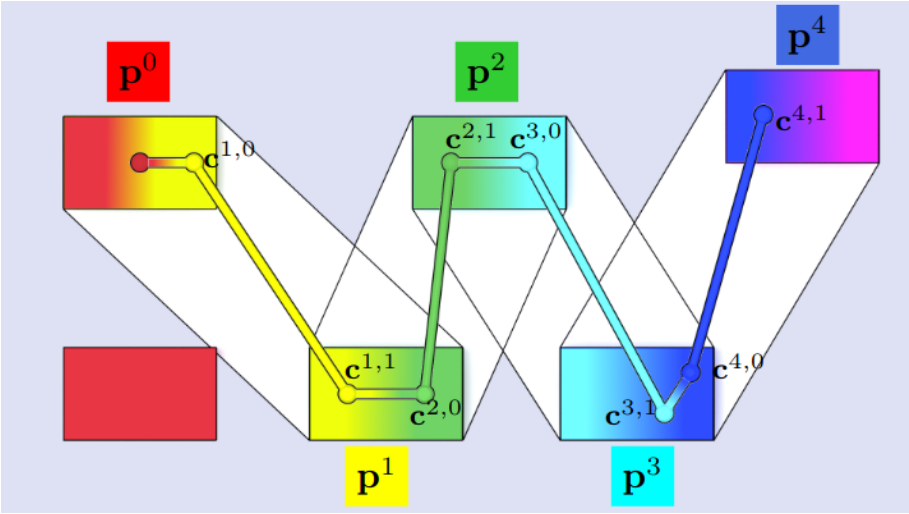


Figure B.1: Computation of a feasible center of mass quasi-static trajectory using the 2PAC method [Ton+18b]. Source: Tonneau et al. [Ton+20]

Relative foot position constraints. Just as for the COM reachability, a 3D polytope $\mathcal{Q}$ is randomly sampled offline (or manually given) to approximate the reachable workspace of each foot with respect to the other. The polytope rotated by r_{k-1} then translated by p_{k-1} can be expressed as $\mathcal{Q}_k : \{\mathrm{p} \in \mathbb{R}^3, Q_k\mathrm{p} \leq q_k\}$, where Q_k and q_k are the matrix and vector defining the rotated polytope. For phase k, the relative foot position constraints can be formulated as follows:

$$\mathcal{Q}_{k-1}(\mathrm{p}_k - \mathrm{p}_{k-1}) \leq q_{k-1} \tag{B.3}$$

B.2 Complete MIP Formulation

The complete formulation of the contact-before-motion MIP contact planning problem with the center of mass positions and the feasibility constraints of previous Appendix B.1 is:

$$
\begin{aligned}
\textbf{find} \quad & \mathrm{P} = [\mathrm{p}_1, ..., \mathrm{p}_n],\ \mathrm{p}_i \in \mathbb{R}^{3\times n} \\
& R = [\mathrm{r}_1, ..., \mathrm{r}_n],\ r_i \in \mathbb{R}^{3\times n} \\
& \mathrm{C} = [\mathrm{c}_{0,1}, \mathrm{c}_{1,0}, \mathrm{c}_{1,1}, ..., \mathrm{c}_{n,0}, \mathrm{c}_{n,1}],\ \mathrm{c}_{k,e} \in \mathbb{R}^3 \\
& A = [\mathrm{a}_1, ..., \mathrm{a}_n],\ \mathrm{a}_i \in \{0,1\}^m \\
& \beta = [\beta_1, ..., \beta_n],\ \beta_i \in \mathbb{R}^m \\
\textbf{min} \quad & l(\mathrm{P}, \mathrm{R}, \mathrm{C}) \\
\textbf{s.t.} \quad & \{\mathrm{P}, \mathrm{R}, \mathrm{C}\} \in \mathcal{I} \cap \mathcal{G} \\
& \forall i \in \{1, .., n\} : \\
& \quad \mathrm{card}(a_i) = m - 1 \\
& \quad \forall j \in \{1, .., m\} : \\
& \quad\quad S^j \mathrm{p}_i \leq s^j + M a_i^j \mathbf{1} \\
& \quad\quad (p_i)^\intercal \mathbf{d}^j = e^j + \beta_i^j \\
& \quad\quad ||\beta_i^j||_1 \leq M a_i^j \\
& \quad\quad F_{i-1}(\mathrm{c}_{i,0} - \mathrm{p}_{i-1}) \leq f_{i-1} \\
& \quad\quad F_i(\mathrm{c}_{i,1} - \mathrm{p}_i) \leq f_i \\
& \quad\quad R_i(\mathrm{c}_{i,e} - \mathrm{p}_i) \leq r_i \\
& \quad\quad R_{i-1}(\mathrm{c}_{i-1,e} - \mathrm{p}_{i-1}) \leq r_{i-1} \\
& \quad\quad \mathcal{Q}_{i-1}(p_i - \mathrm{p}_{i-1}) \leq q_{i-1}
\end{aligned}
$$

Abstract

This thesis explores how to generate paths for legged robot locomotion.

One approach to tackle the locomotion problem is its division into three sequential modules: navigation to generate a guide path that the robot has to follow, contact planning along this guide path, and finally the robot whole-body motion. This division greatly reduces the locomotion problem complexity, but raises the critical question of the "feasibility" between the different modules. In this context, this thesis explores the feasibility problem between the navigation and the next modules, in other words: "How to generate feasible paths by the robot?"

A naive approach is to use a reduced model of the robot with two conditions: the robot trunk must not collide with the environment, and the robot feet must be able to reach the ground all along the path. But these two conditions are not sufficient to approximate path feasibility. To refine these conditions, another approach is to consider the traversability of the terrain, to generate more likely easier paths for the robot. This thesis explores a different approach that is to learn by reinforcement how to generate feasible paths directly from the contact planner.

My contribution is a local steering method, named Leas, which locally navigates the terrain in the desired direction using a height map. Leas learns from the contact planner validation what is a feasible path by it, and consequently adapts its navigation behavior.

This steering method has been connected to three contact planners, each having different strategies. I will explain its results and limitations for legged robot locomotion in complex environments.

Keywords

Reinforcement learning, Navigation, Locomotion, Robotics, Humanoids

Résumé

Le but de ma thèse est d'apprendre comment générer des chemins pour la locomotion de robots à pattes.

Une approche possible au problème de la locomotion est une division en trois modules séquentiels qui sont: la navigation pour générer un chemin (ou guide) que le robot devra suivre, la planification de ses pas tout le long du chemin, puis enfin le mouvement corps complet du robot pour les réaliser. Cette division permet de réduire la complexité du problème, mais amène la question critique de la "faisabilité" entre les différents modules. Dans ce contexte, cette thèse s'intéresse à la question de la faisabilité entre le module de navigation et les autres modules, autrement dit: "Comment générer des chemins faisables par le robot?"

Une approche naïve repose sur un modèle réduit du robot apportant deux conditions: le tronc du robot ne soit pas en collision avec l'environnement, et les pieds du robot doivent pouvoir atteindre le sol tout le long du chemin. Mais ces deux conditions ne sont pas suffisantes pour approximer la faisabilité des chemins. Pour raffiner ces conditions, une deuxième approche est de s'intéresser au concept de traversabilité des terrains, afin de de générer des chemins plus faciles pour le robot. Cette thèse explore une autre approche qui est d'apprendre par renforcement à générer des chemins faisables directement via le planificateur de contact.

Ma contribution est une méthode de pilotage, nommée Leas, qui grâce à une carte d'élévation locale navigue le terrain dans une direction désirée. Leas apprend via la validation par le planificateur de contact ce qu'est un chemin faisable par lui, et modifie ses comportements de navigation en conséquence. Cette méthode de pilotage a été connectée à trois planificateurs de contacts ayant des stratégies différentes. Je vais montrer ses résultats et ses limitations pour la locomotion de robot à pattes dans des environnements complexes.

Mots clefs

Apprentissage par renforcement, Navigation, Locomotion, Robotique, Humanoïde

Bibliography

[18] "A constrained A* approach towards optimal path planning for an unmanned surface vehicle in a maritime environment containing dynamic obstacles and ocean currents". In: *Ocean Engineering* (2018) (cit. on p. 19).

[Abd12] Ahmad Abdul Karim. "Procedural locomotion of multi-legged characters in complex dynamic environments : real-time applications". PhD thesis. 2012 (cit. on p. 14).

[Ace+19] Bernardo Aceituno-Cabezas et al. "Simultaneous Contact, Gait and Motion Planning for Robust Multi-Legged Locomotion via Mixed-Integer Convex Optimization". In: *CoRR* (2019) (cit. on pp. 9, 13).

[Ach+20] Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. "Presolve Reductions in Mixed Integer Programming". In: (2020) (cit. on pp. 74, 84).

[Agh+21] Ali Agha et al. *NeBula: Quest for Robotic Autonomy in Challenging Environments; TEAM CoSTAR at the DARPA Subterranean Challenge*. 2021 (cit. on p. 1).

[Alo+20] Eloi Alonso, Maxim Peter, David Goumard, and Joshua Romoff. "Deep Reinforcement Learning for Navigation in AAA Video Games". In: *CoRR* (2020) (cit. on pp. 22, 32).

[And+17] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *CoRR* (2017) (cit. on p. 69).

[And+20] Marcin Andrychowicz et al. "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study". In: *CoRR* (2020) (cit. on p. 39).

[Apg+] Taylor Apgar, Patrick Clary, Kevin R. Green, Alan Fern, and Jonathan W. Hurst. "Fast Online Trajectory Optimization for the Bipedal Robot Cassie". In: *Robotics: Science and Systems* () (cit. on p. 8).

[Bal+96] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. "Gomory cuts revisited". In: *Operations Research Letters* (1996) (cit. on p. 74).

[BT02] Devin J. Balkcom and J.C. Trinkle. "Computing Wrench Cones for Planar Rigid Body Contact Tasks". In: *The International Journal of Robotics Research* (2002) (cit. on p. 13).

[Bau+11] Léo Baudouin, Nicolas Perrin, Thomas Moulard, Florent Lamiraux, Olivier Stasse, and Eiichi Yoshida. "Real-time Replanning Using 3D Environment for Humanoid Robot". In: *IEEE-RAS, HUMANOIDS* (2011) (cit. on p. 14).

[BM18] Julián Esteban Herrera Benavides and Robinson Jimenez Moreno. “Flood Fill Algorithm Dividing Matrices for Robotic Path Planning”. In: (2018) (cit. on p. 19).

[Ber+11] Jur van den Berg, Stephen Guy, Ming Lin, and Dinesh Manocha. “Reciprocal n-Body Collision Avoidance”. In: 2011 (cit. on p. 22).

[Ber+19] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. “DReCon: Data-Driven Responsive Control of Physics-Based Characters”. In: *ACM SIGGRAPH* (2019) (cit. on p. 10).

[BG07] Priyadarshi Bhattacharya and Marina L. Gavrilova. “Voronoi diagram in optimal path planning”. In: (2007) (cit. on p. 19).

[Blo+22] Michael Bloesch et al. “Towards Real Robot Learning in the Wild: A Case Study in Bipedal Locomotion”. In: *PMLR* (2022) (cit. on p. 11).

[Bou99] Paul Bourke. “Nearest neighbour weighted interpolation”. In: (1999) (cit. on p. 20).

[Bou+18] Karim Bouyarmane, Stéphane Caron, Adrien Escande, and Abderrahmane Kheddar. “Multi-contact Motion Planning and Control”. In: (2018) (cit. on p. 16).

[Bou+09a] Karim Bouyarmane, Adrien Escande, Florent Lamiraux, and Abderrahmane Kheddar. “Potential Field Guide for Humanoid Multicontacts Acyclic Motion Planning”. In: *IEEE ICRA* (2009) (cit. on p. 16).

[Bou+09b] Karim Bouyarmane, Adrien Escande, Florent Lamiraux, and Abderrahmane Kheddar. “Potential field guide for humanoid multicontacts acyclic motion planning”. In: *ICRA* (2009) (cit. on p. 28).

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004 (cit. on p. 85).

[BFH19] Martim Brandão, Maurice Fallon, and Ioannis Havoutis. “Multi-controller multi-objective locomotion planning for legged robots”. In: (2019) (cit. on pp. 16, 17, 83).

[Bra+22] Angelo Bratta, Avadesh Meduri, Michele Focchi, Ludovic Righetti, and Claudio Semini. *ContactNet: Online Multi-Contact Planning for Acyclic Legged Robot Locomotion*. 2022 (cit. on p. 14).

[Bre06] Timothy Bretl. “Motion Planning of Multi-Limbed Robots Subject to Equilibrium Constraints: The Free-Climbing Robot Problem”. In: *The International Journal of Robotics Research* (2006) (cit. on pp. 13, 14).

[Bre19] Daniel Brewer. “Tactical Pathfinding on a NavMesh”. In: *Game AI Pro 360* (2019) (cit. on p. 19).

[Car+18] Stéphane Caron, Adrien Escande, Leonardo Lanari, and Bastien Mallein. “Capturability-based Analysis, Optimization and Control of 3D Bipedal Walking”. In: *CoRR* (2018) (cit. on pp. 6, 13).

[CK16] Stéphane Caron and Abderrahmane Kheddar. “Multi-contact walking pattern generation based on model preview control of 3D COM accelerations”. In: *IEEE Humanoids* (2016) (cit. on p. 1).

[Car17] Justin Carpentier. “Computational foundations of anthropomorphic locomotion”. PhD thesis. Universite Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), 2017 (cit. on p. 8).

[Car+16] Justin Carpentier, Steve Tonneau, Maximilien Naveau, Olivier Stasse, and Nicolas Mansard. “A Versatile and Efficient Pattern Generator for Generalized Legged Locomotion”. In: *IEEE-ICRA* (2016) (cit. on pp. 6, 12, 13).

[Car+17] Justin Carpentier et al. “Multi-contact Locomotion of Legged Robots in Complex Environments – The Loco3D project”. In: *RSS Workshop on Challenges in Dynamic Legged Locomotion* (2017) (cit. on pp. 3, 6, 7, 28–30).

[CFS06] Joseph Carsten, Dave Ferguson, and Anthony Stentz. “3D Field D: Improved Path Planning and Replanning in Three Dimensions”. In: (2006) (cit. on p. 19).

[CS19] Germán Castro and Claude Sammut. “Learning footstep planning on irregular surfaces with partial placements”. In: (2019) (cit. on p. 14).

[Cha+18] Konstantinos I. Chatzilygeroudis, Vassilis Vassiliades, Freek Stulp, Sylvain Calinon, and Jean-Baptiste Mouret. “A survey on policy search algorithms for learning robot controllers in a handful of trials”. In: *CoRR* (2018) (cit. on p. 11).

[Che+21] Jason Chemin, Pierre Fernbach, Daeun Song, Guilhem Saurel, Nicolas Mansard, and Steve Tonneau. “Learning to steer a locomotion contact planner”. In: *IEEE ICRA* (2021) (cit. on pp. 28, 31, 36, 38, 39, 42, 63).

[CLD22] Jason Chemin, Pierre-Alexandre Léziart, and Ewen Dantec. *Environment RL for talos and solo robots*. 2022. URL: https://github.com/JasonChmn/robots_RL (cit. on p. 30).

[Che+16] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P. How. “Decentralized Non-communicating Multiagent Collision Avoidance with Deep Reinforcement Learning”. In: *CoRR* (2016) (cit. on p. 22).

[CK04] J. Chestnutt and J.J. Kuffner. “A tiered planning strategy for biped navigation”. In: *IEEE-RAS International Conference on Humanoid Robots* (2004) (cit. on p. 16).

[Che07] Joel E. Chestnutt. “Navigation planning for legged robots”. In: (2007) (cit. on pp. 16, 23).

[Che+03] Joel Chestnutt, James Kuffner, Koichi Nishiwaki, and Satoshi Kagami. “Planning Biped Navigation Strategies in Complex Environments”. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2003 (cit. on p. 14).

[Che+09] Joel Chestnutt, Koichi Nishiwaki, James Kuffner, and Satoshi Kagami. “Interactive control of humanoid navigation”. In: (2009) (cit. on pp. 12, 16).

[CDG08] Christine Chevallereau, Dalila Djoudi, and Jessy W. Grizzle. “Stable Bipedal Walking With Foot Rotation Through Direct Regulation of the Zero Moment Point”. In: *IEEE Transactions on Robotics* (2008) (cit. on p. 8).

[Chi+19a] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. “Learning Navigation Behaviors End-to-End With AutoRL”. In: *IEEE-RAL* (2019) (cit. on p. 49).

[Chi+19b] Hao-Tien Lewis Chiang, Jasmine Hsu, Marek Fiser, Lydia Tapia, and Aleksandra Faust. "RL-RRT: Kinodynamic Motion Planning via Learning Reachability Estimators From RL Policies". In: *IEEE-RAL* (2019) (cit. on pp. 22, 43, 49).

[CB16] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016 (cit. on p. 9).

[CDT18] Gabriel Victor de la Cruz, Yunshu Du, and Matthew E. Taylor. "Pre-training with Non-expert Human Demonstration for Deep Reinforcement Learning". In: *CoRR* (2018) (cit. on p. 70).

[CCM14] Antoine Cully, Jeff Clune, and Jean-Baptiste Mouret. "Robots that can adapt like natural animals". In: *CoRR* (2014) (cit. on p. 11).

[DVT14] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. "Whole-body motion planning with centroidal dynamics and full kinematics". In: *IEEE-RAS* (2014) (cit. on p. 9).

[DLL12] Duong Dang, Jean-Paul Laumond, and Florent Lamiraux. "Experiments on whole-body manipulation and locomotion with footstep real-time optimization". In: (2012) (cit. on p. 8).

[Dan+22] Ewen Louis Dantec et al. "Whole-Body Model Predictive Control for Biped Locomotion on a Torque-Controlled Humanoid Robot". 2022 (cit. on pp. 6, 7, 12, 13).

[DT14] R. Deits and R. Tedrake. "Footstep planning on uneven terrain with mixed-integer convex optimization". In: (2014) (cit. on pp. 13, 14, 37, 72, 74).

[DTM16] Andrea Del Prete, Steve Tonneau, and Nicolas Mansard. "Fast algorithms to test robust static equilibrium for legged robots". In: *IEEE ICRA* (2016) (cit. on p. 13).

[Dij59] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numer. Math.* (1959) (cit. on p. 18).

[Dua+21] Helei Duan, Jeremy Dao, Kevin Green, Taylor Apgar, Alan Fern, and Jonathan Hurst. "Learning Task Space Actions for Bipedal Locomotion". In: (2021) (cit. on p. 8).

[Duc+14] František Duchoň et al. "Path Planning with Modified a Star Algorithm for a Mobile Robot". In: *Procedia Engineering* (2014) (cit. on p. 19).

[EV10] Arjan Egges and Ben Van Basten. "One step at a time: Animating virtual characters based on foot placement". In: *The Visual Computer* (2010) (cit. on p. 16).

[EK08] Adrien Escande and Abderrahmane Kheddar. "Planning contact supports for acyclic motion with task constraints and experiment on HRP-2". In: *ISER* (2008) (cit. on p. 28).

[EKM06] Adrien Escande, Abderrahmane Kheddar, and Sylvain Miossec. "Planning support contact-points for humanoid robots and experiments on HRP-2". In: *IEEE-RSJ* (2006) (cit. on p. 14).

[Esp+18] Lasse Espeholt et al. “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures”. In: *ICML*. 2018 (cit. on pp. 40, 41).

[FL08] Dave Ferguson and Maxim Likhachev. “Efficiently Using Cost Maps For Planning Complex Maneuvers”. In: *Lab Papers (GRASP)* (2008) (cit. on p. 19).

[Fer+20] P. Fernbach, S. Tonneau, O. Stasse, J. Carpentier, and M. Taïx. “C-CROC: Continuous and Convex Resolution of Centroidal Dynamic Trajectories for Legged Robots in Multicontact Scenarios”. In: *IEEE Transactions on Robotics* (2020) (cit. on p. 13).

[Fer+17] Pierre Fernbach, Steve Tonneau, Andrea Del Prete, and Michel Taïx. “A Kinodynamic steering-method for legged multi-contact locomotion”. In: *IEEE IROS* (2017) (cit. on pp. 20, 23, 29, 43, 76).

[Fer21] Paolo Ferrari. “Motion Planning Techniques for Humanoid Robots”. PhD thesis. 2021 (cit. on p. 14).

[FBT97] D. Fox, W. Burgard, and S. Thrun. “The dynamic window approach to collision avoidance”. In: *IEEE Robotics and Automation Magazine* (1997) (cit. on p. 21).

[Fra+20] Anthony Francis et al. “Long-Range Indoor Navigation With PRM-RL”. In: *IEEE Transactions on Robotics* (2020) (cit. on pp. 20, 22, 25).

[FHM18] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *CoRR* (2018) (cit. on pp. 10, 39).

[FDY17] Toshio Fukuda, Paolo Dario, and Guang-Zhong Yang. “Humanoid robotics: History, current state of the art, and challenges”. In: *Science Robotics* (2017) (cit. on p. 6).

[Gan+22] Siddhant Gangapurwala, Mathieu Geisert, Romeo Orsolino, Maurice Fallon, and Ioannis Havoutis. “RLOC: Terrain-Aware Legged Locomotion Using Reinforcement Learning and Optimal Control”. In: *IEEE TRO* (2022) (cit. on pp. 7, 13, 14, 32, 49).

[GPS13] Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. “Flexible Muscle-Based Locomotion for Bipedal Creatures”. In: *ACM SIGGRAPH* (2013) (cit. on p. 10).

[Ger10] Roland Geraerts. “Planning short paths with clearance using explicit corridors”. In: (2010) (cit. on p. 21).

[GO07] Roland Geraerts and Mark H. Overmars. “The Corridor Map Method: Real-Time High-Quality Path Planning”. In: *IEEE ICRA* (2007) (cit. on p. 21).

[Gon+18] Yukai Gong et al. “Feedback Control of a Cassie Bipedal Robot: Walking, Standing, and Riding a Segway”. In: *CoRR* (2018) (cit. on pp. 7, 8).

[Gri+19a] Robert J. Griffin, Georg Wiedebach, Stephen McCrory, Sylvain Bertrand, Inho Lee, and Jerry Pratt. “Footstep Planning for Autonomous Walking Over Rough Terrain”. In: *IEEE-RAS (Humanoids)* (2019) (cit. on p. 14).

[Gri+19b] Felix Grimminger et al. “An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research”. In: *CoRR* (2019) (cit. on p. 6).

[Gri+14] Jessy Grizzle, Chevallereau Christine, Aaron Ames, and Ryan Sinnet. "3D Bipedal Robotic Walking: Models, Feedback Control, and Open Problems". In: *Automatica* (2014) (cit. on p. 8).

[Gura] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. Accessed: 2022-09-01. URL: https://www.gurobi.com/documentation/9.5/refman/index.html (cit. on pp. 74, 76, 78, 84).

[Gurb] Gurobi Optimization, LLC. *Mixed-Integer Programming (MIP) - A Primer on the Basics*. Accessed: 2022-09-01. URL: https://www.gurobi.com/resource/mip-basics/ (cit. on pp. 9, 74).

[Haa+18a] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* (2018) (cit. on pp. 10, 39).

[Haa+18b] Tuomas Haarnoja, Aurick Zhou, Sehoon Ha, Jie Tan, George Tucker, and Sergey Levine. "Learning to Walk via Deep Reinforcement Learning". In: *CoRR* (2018) (cit. on p. 11).

[Haf+20] Roland Hafner et al. "Towards General and Autonomous Learning of Core Skills: A Case Study in Locomotion". In: *CoRR* (2020) (cit. on p. 11).

[HRL15] Perttu Hämäläinen, Joose Rajamäki, and C. Karen Liu. "Online Control of Simulated Humanoids Using Particle Belief Propagation". In: *ACM SIGGRAPH* (2015) (cit. on p. 9).

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* (1968) (cit. on p. 18).

[Hauss] Kris Hauser. *Robotic Systems*. Draft in progress (cit. on p. 20).

[Hau+08] Kris Hauser, Timothy Bretl, Jean-Claude Latombe, Kensuke Harada, and Brian Wilcox. "Motion Planning for Legged Robots on Varied Terrain". In: *The International Journal of Robotics Research* (2008) (cit. on p. 14).

[Haw+20] Brandon Haworth, Glen Berseth, Seonghyeon Moon, Petros Faloutsos, and Mubbasir Kapadia. "Deep Integration of Physical Humanoid Control and Crowd Navigation". In: *Motion, Interaction and Games* (2020) (cit. on p. 22).

[Hee+17] Nicolas Heess et al. "Emergence of Locomotion Behaviours in Rich Environments". In: *CoRR* (2017) (cit. on p. 10).

[Hen+18] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. "Deep Reinforcement Learning That Matters". In: 2018 (cit. on p. 39).

[Her+10] Andrei Herdt, Holger Diedam, Pierre-Brice Wieber, Dimitar Dimitrov, Katja Mombaur, and Moritz Diehl. "Online Walking Motion Generation with Automatic Foot Step Placement". In: *Advanced Robotics* (2010) (cit. on p. 8).

[HPW10] Andrei Herdt, Nicolas Perrin, and Pierre-Brice Wieber. "Walking without thinking about it". In: (2010) (cit. on p. 8).

[Hil+17] Arne-Christoph Hildebrandt et al. "Real-Time Path Planning in Unknown Environments for Bipedal Robots". In: *IEEE-RAL* (2017) (cit. on pp. 16, 23).

[Hil+18] Ashley Hill et al. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018 (cit. on pp. 39, 41, 69).

[Hol+20] Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. "Learned Motion Matching". In: *ACM SIGGRAPH* (2020) (cit. on p. 8).

[HKS17] Daniel Holden, Taku Komura, and Jun Saito. "Phase-Functioned Neural Networks for Character Control". In: *ACM SIGGRAPH* (2017) (cit. on p. 8).

[Hor+12] Armin Hornung, Andrew Dornbush, Maxim Likhachev, and Maren Bennewitz. "Anytime search-based footstep planning with suboptimality bounds". In: *IEEE-RAS (Humanoids)* (2012) (cit. on p. 14).

[Hu+19a] Yuanming Hu et al. "ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics". In: (2019) (cit. on p. 10).

[Hu+19b] Yuanming Hu et al. "DiffTaichi: Differentiable Programming for Physical Simulation". In: *CoRR* (2019) (cit. on p. 10).

[HC04] Han-Pang Huang and Shu-Yun Chung. "Dynamic visibility graph for path planning". In: *IEEE IROS* (2004) (cit. on p. 19).

[HA92] Y.K. Hwang and N. Ahuja. "A potential field approach to path planning". In: *IEEE Transactions on Robotics and Automation* (1992) (cit. on p. 21).

[HLH18] Jemin Hwangbo, Joonho Lee, and Marco Hutter. "Per-contact iteration method for solving contact dynamics". In: *IEEE-RAL* (2018) (cit. on p. 49).

[Iba+21] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. "How to train your robot with deep reinforcement learning: lessons we have learned". In: *The International Journal of Robotics Research* (2021) (cit. on p. 11).

[Kaj+02] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa. "A realtime pattern generator for biped walking". In: *IEEE ICRA* (2002) (cit. on p. 8).

[Kaj+03] S. Kajita et al. "Biped walking pattern generation by using preview control of zero-moment point". In: (2003) (cit. on pp. 6, 13).

[Kaj+14] Shuuji Kajita, Hirohisa Hirukawa, Kensuke Harada, and Kazuhito Yokoi. *Introduction to Humanoid Robotics*. 2014 (cit. on p. 6).

[Kal+11] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. "Learning, planning, and control for quadruped locomotion over challenging terrain". In: *The International Journal of Robotics Research* (2011) (cit. on pp. 16, 24).

[Kav+96] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE-RAL* (1996) (cit. on pp. 14, 20).

[KE08] Abderrahmane Kheddar and Adrien Escande. "Challenges in Contact-Support Planning for Acyclic Motion of Humanoids and Androids". In: *ISR: International Symposium on Robotics* (2008) (cit. on p. 16).

[Kir+20] Bangalore Ravi Kiran et al. "Deep Reinforcement Learning for Autonomous Driving: A Survey". In: *CoRR* (2020) (cit. on p. 22).

[KBP13] Jens Kober, J. Bagnell, and Jan Peters. “Reinforcement Learning in Robotics: A Survey”. In: *The International Journal of Robotics Research* (2013) (cit. on p. 11).

[KRN08] J. Zico Kolter, Mike P. Rodgers, and Andrew Y. Ng. “A control architecture for quadruped locomotion over rough terrain”. In: (2008) (cit. on pp. 16, 24).

[Kuf99] James Kuffner. “Autonomous agents for real-time animation”. In: (1999) (cit. on p. 15).

[Kui+15] Scott Kuindersma et al. “Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot”. In: *Autonomous Robots* (2015) (cit. on p. 9).

[Kum+20] Iori Kumagai, Mitsuharu Morisawa, Shizuko Hattori, Mehdi Benallegue, and Fumio Kanehiro. “Multi-Contact Locomotion Planning for Humanoid Robot Based on Sustainable Contact Graph With Local Contact Modification”. In: *IEEE-RAL* (2020) (cit. on p. 14).

[KS14] Tobias Kunz and Mike Stilman. “Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits”. In: *IEEE IROS* (2014) (cit. on pp. 20, 21, 29, 43).

[Kwi+22] Ariel Kwiatkowski et al. “A Survey on Reinforcement Learning Methods in Character Animation”. In: *Computer Graphics Forum* (2022). URL: https://hal.archives-ouvertes.fr/hal-03600947 (cit. on pp. 10, 22, 39).

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991 (cit. on p. 20).

[LaV98] Steven M. LaValle. “Rapidly-exploring random trees : a new tool for path planning”. In: *The annual research report* (1998) (cit. on pp. 14, 20).

[LaV06] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006 (cit. on p. 20).

[LJ01] Steven M. LaValle and Jr. James J. Kuffner. “Randomized Kinodynamic Planning”. In: *The International Journal of Robotics Research* (2001) (cit. on pp. 20, 21).

[Le +22] Quentin Le Lidec, Louis Montaut, Cordelia Schmid, Ivan Laptev, and Justin Carpentier. “Leveraging Randomized Smoothing for Optimal Control of Non-smooth Dynamical Systems”. 2022 (cit. on p. 10).

[Lec21] Dimitri Leca. “Navigation autonome d’un robot agricole”. PhD thesis. Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), 2021 (cit. on p. 21).

[LWL18] Jaedong Lee, Jungdam Won, and Jehee Lee. “Crowd Simulation by Deep Reinforcement Learning”. In: *ACM MIG* (2018) (cit. on p. 22).

[Lee+20] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. “Learning Quadrupedal Locomotion over Challenging Terrain”. In: *CoRR* (2020) (cit. on pp. 7, 8, 24).

[Lee+19] Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. “Scalable Muscle-Actuated Human Simulation and Control”. In: *ACM SIGGRAPH* (2019) (cit. on p. 10).

[Léz+] Pierre-Alexandre Léziart, Thomas Flayols, Felix Grimminger, Nicolas Mansard, and Philippe Souères. "Implementation of a Reactive Walking Controller for the New Open-Hardware Quadruped Solo-12". In: *ICRA 2021* () (cit. on pp. 6, 12, 13).

[Li+18] Tianyu Li, Akshara Rai, Hartmut Geyer, and Christopher G. Atkeson. "Using Deep Reinforcement Learning to Learn High-Level Policies on the ATRIAS Biped". In: *CoRR* (2018) (cit. on p. 12).

[Li+21] Zhongyu Li et al. "Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots". In: (2021) (cit. on p. 8).

[Lid02] Lars Liden. "Strategic and Tactical Reasoning with Waypoints". In: (2002) (cit. on p. 19).

[LGT03] Maxim Likhachev, Geoff Gordon, and Sebastian Thrun. "ARA*: Anytime A* with Provable Bounds on Sub-Optimality". In: (2003) (cit. on p. 19).

[LB17] Yu-Chi Lin and Dmitry Berenson. "Humanoid navigation in uneven terrain using learned estimates of traversability". In: (2017) (cit. on p. 24).

[LB18] Yu-Chi Lin and Dmitry Berenson. "Humanoid Navigation Planning in Large Unstructured Environments Using Traversability - Based Segmentation". In: *IEEE IROS* (2018) (cit. on pp. 16, 23, 83).

[Liu+17] Chang Liu, Seungho Lee, Scott Varnhagen, and H. Eric Tseng. "Path planning for autonomous vehicles using model predictive control". In: (2017) (cit. on p. 21).

[Lof] J. Lofberg. *Big-m and convex hulls*. Accessed: 2022-09-01. URL: https://yalmip.github.io/tutorial/bigmandconvexhulls (cit. on p. 73).

[Luo+20] Ying-Sheng Luo, Jonathan Hans Soeseno, Trista Pei-Chun Chen, and Wei-Chao Chen. "CARL: Controllable Agent with Reinforcement Learning for Quadruped Locomotion". In: *CoRR* (2020) (cit. on p. 10).

[Maj+21] Amjad Majid et al. "Deep Reinforcement Learning Versus Evolution Strategies: A Comparative Survey". In: *CoRR* (2021) (cit. on pp. 10, 40).

[MSW20] I. Maroger, O. Stasse, and B. Watier. "Walking Human Trajectory Models and Their Application to Humanoid Robot Locomotion". In: (2020) (cit. on p. 23).

[MSE01] C.F. Martin, Shan Sun, and M. Egerstedt. "Optimal control, statistics and path planning". In: *Mathematical and Computer Modelling* (2001). Computation and control VI proceedings of the sixth Bozeman conference (cit. on p. 21).

[Mir+16] Joseph Mirabel et al. "HPP: a new software for constrained motion planning". In: (2016) (cit. on pp. 17, 36, 41).

[MB08] Swati Mishra and Pankaj Bande. "Maze Solving Algorithms for Micro Mouse". In: *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems* (2008) (cit. on p. 19).

[MK19] J.C. Mohanta and Anupam Keshari. "A knowledge based fuzzy-probabilistic roadmap method for mobile robot navigation". In: *Applied Soft Computing* (2019) (cit. on p. 20).

[MTP12] Igor Mordatch, Emanuel Todorov, and Zoran Popović. “Discovery of Complex Behaviors through Contact-Invariant Optimization”. In: *ACM SIGGRAPH* (2012) (cit. on p. 9).

[MTG17] Steffen Müller, Thanh Trinh, and Horst-Michael Gross. “Local Real-Time Motion Planning Using Evolutionary Optimization”. In: *Towards Autonomous Robotic Systems* (2017) (cit. on p. 22).

[Nak+11] Danial Nakhaeinia, S.H. Tang, Samsul Noor, and O. Motlagh. “A review of control architectures for autonomous navigation of mobile robots”. In: *International Journal of Physical Sciences* (2011) (cit. on p. 18).

[Nar+20] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. “Curriculum learning for reinforcement learning domains: A framework and survey”. In: *Journal of Machine Learning Research* (2020) (cit. on pp. 39, 49).

[Nav+12] Nicolás Navarro-Guerrero, Cornelius Weber, Pascal Schroeter, and Stefan Wermter. “Real-world reinforcement learning for autonomous humanoid robot docking”. In: *IEEE-RAS* (2012) (cit. on p. 11).

[Nav+17] M. Naveau, M. Kudruss, O. Stasse, C. Kirches, K. Mombaur, and P. Souères. “A Reactive Walking Pattern Generator Based on Nonlinear Model Predictive Control”. In: *IEEE-RAL* (2017) (cit. on p. 8).

[NCK12] Koichi Nishiwaki, Joel Chestnutt, and Satoshi Kagami. “Autonomous Navigation of a Humanoid Robot over Unknown Rough Terrain Using a Laser Range Sensor”. In: *International Journal of Robotics Research* (2012) (cit. on p. 14).

[NJ20] Joseph Norby and Aaron M. Johnson. “Fast Global Motion Planning for Dynamic Legged Robots”. In: (2020) (cit. on p. 23).

[Nor+22] Joseph Norby et al. “Quad-SDK: Full Stack Software Framework for Agile Quadrupedal Locomotion”. In: *ICRA Workshop on Legged Robots*. 2022 (cit. on pp. 16, 23, 24).

[Ond+10] Jan Ondrej, Julien Pettré, Anne-Hélène Olivier, and Stéphane Donikian. “A Synthetic-Vision Based Steering Approach for Crowd Simulation”. In: *ACM SIGGRAPH* (2010) (cit. on p. 20).

[Ope+19] OpenAI et al. “Dota 2 with Large Scale Deep Reinforcement Learning”. In: (2019) (cit. on pp. 39–41).

[Par+20] Hwangpil Park, Ri Yu, Yoonsang Lee, Kyungho Lee, and Jehee Lee. “Understanding the Stability of Deep Control Policies for Biped Locomotion”. In: *CoRR* (2020) (cit. on p. 10).

[Pat+19] B.K. Patle, Ganesh Babu L, Anish Pandey, D.R.K. Parhi, and A. Jagadeesh. “A review: On path planning strategies for navigation of mobile robot”. In: *Defence Technology* (2019) (cit. on p. 20).

[Pen+17a] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization”. In: *CoRR* (2017) (cit. on p. 11).

[PBP16] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. "Terrain-adaptive Locomotion Skills Using Deep Reinforcement Learning". In: *ACM SIGGRAPH* (2016) (cit. on p. 10).

[Pen+17b] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. "DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning". In: *ACM SIGGRAPH* (2017) (cit. on pp. 7, 25, 30, 32, 35, 49).

[Per+17] Francisco Perez-Grau, Ricardo Ragel, Fernando Caballero, Antidio Viguria, and Anibal Ollero. "An architecture for robust UAV navigation in GPS-denied areas". In: *Journal of Field Robotics* (2017) (cit. on p. 19).

[Per+12] Nicolas Perrin, Olivier Stasse, Léo Baudouin, Florent Lamiraux, and Eiichi Yoshida. "Fast Humanoid Robot Collision-Free Footstep Planning Using Swept Volume Approximations". In: *IEEE TRO* (2012) (cit. on p. 14).

[PLS03] Julien Pettré, Jean-Paul Laumond, and Thierry Simeon. "A 2-stages locomotion planner for digital actors". In: *ACM SIGGRAPH/Eurographics symposium on Computer animation* (2003) (cit. on pp. 15, 23).

[PS05] Roland Philippsen and Roland Siegwart. "An Interpolated Dynamic Navigation Function". In: (2005) (cit. on p. 19).

[PCT14] Michael Posa, Cecilia Cantu, and Russ Tedrake. "A direct method for trajectory optimization of rigid bodies through contact". In: *The International Journal of Robotics Research* (2014) (cit. on p. 9).

[Pre+16] Andrea del Prete, Nicolas Mansard, Oscar Efrain Ramos Ponce, Olivier Stasse, and Francesco Nori. "Implementing Torque Control with High-Ratio Gear Boxes and without Joint-Torque Sensors". In: *International Journal of Humanoid Robotics* (2016) (cit. on pp. 7, 30).

[Rai86] Marc Raibert. *Legged Robots That Balance*. MIT Press, 1986 (cit. on pp. 14, 94).

[Raj+17] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *CoRR* (2017) (cit. on pp. 69, 70).

[Ras+17] Yadollah Rasekhipour, Amir Khajepour, Shih-Ken Chen, and Bakhtiar Litkouhi. "A Potential Field-Based Model Predictive Path-Planning Controller for Autonomous Road Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* (2017) (cit. on p. 21).

[Reb+07] John R. Rebula, Peter D. Neuhaus, Brian V. Bonnlander, Matthew J. Johnson, and Jerry E. Pratt. "A Controller for the LittleDog Quadruped Walking on Rough Terrain". In: *IEEE ICRA* (2007) (cit. on p. 14).

[RH20] Siavash Rezazadeh and Jonathan W Hurst. "Control of ATRIAS in three dimensions: Walking as a forced-oscillation problem". In: *The International Journal of Robotics Research* (2020) (cit. on p. 8).

[Rie+18] Martin Riedmiller et al. "Learning by Playing Solving Sparse Reward Tasks from Scratch". In: *PMLR*. 2018 (cit. on p. 69).

[Ris+22] Fanny Risbourg, Thomas Corbères, Pierre-Alexandre Léziart, Thomas Flayols, Nicolas Mansard, and Steve Tonneau. "Real time footstep planning and control of the Solo quadruped robot in 3D environments". In: (2022) (cit. on pp. 6, 16, 79, 94).

[Rös] Christoph Rösmann. "Difference between DWA and TEB local planners". In: () (cit. on p. 21).

[Rös+13] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. "Efficient trajectory optimization using a sparse model". In: *2013 European Conference on Mobile Robots* (2013) (cit. on p. 21).

[Rub] Paul A. Rubin. *Perils of "Big M"*. Accessed: 2022-09-01. URL: https://orinanobworld.blogspot.com/2011/07/perils-of-big-m.html (cit. on p. 73).

[SC12] Chelsea Sabo and Kelly Cohen. "Fuzzy Logic UAV (Unmanned Aerial Vehicle) Motion Planning". In: *Advances in Fuzzy Systems* (2012) (cit. on p. 20).

[SRR20] R.A. Saeed, Diego Reforgiato Recupero, and Paolo Remagnino. "A Boundary Node Method for path planning of mobile robots". In: *Robotics and Autonomous Systems* (2020) (cit. on p. 19).

[Sal+17] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: (2017) (cit. on p. 10).

[Sch+10] Erik Schuitema, Martijn Wisse, Thijs Ramakers, and Pieter Jonker. "The design of LEO: A 2D bipedal walking robot for online autonomous Reinforcement Learning". In: *IEEE-RSJ* (2010) (cit. on p. 12).

[Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* (2017) (cit. on pp. 10, 36, 39).

[Sci+19] Nicola Scianca, Daniele De Simone, Leonardo Lanari, and Giuseppe Oriolo. "MPC for Humanoid Gait Generation: Stability and Feasibility". In: *CoRR* (2019) (cit. on p. 14).

[SLL01] Thierry Siméon, Jean-Paul Laumond, and Florent Lamiraux. "Move3D: A generic platform for motion planning". In: (2001) (cit. on p. 20).

[Sin+22] Rohan Pratap Singh, Mehdi Benallegue, Mitsuharu Morisawa, Rafael Cisneros, and Fumio Kanehiro. *Learning Bipedal Walking On Planned Footsteps For Humanoid Robots*. 2022 (cit. on p. 7).

[Sno00] Greg Snook. *Simplified 3D Movement and Pathfinding Using Navigation Meshes*. Ed. by Mark DeLoura. Charles River Media, 2000 (cit. on p. 19).

[SC] Daeun Song and Jason Chemin. *Random scene generator: Playground*. URL: https://github.com/daeunSong/random-scene-gen/tree/playground (cit. on p. 38).

[Son+20] Daeun Song et al. "Solving Footstep Planning as a Feasibility Problem using L1-norm Minimization". In: *IEEE-RAL* (2020) (cit. on pp. 3, 14, 16, 23, 29, 38, 72, 73, 75, 76, 83–85, 89, 91).

[Sta+] O. Stasse et al. "TALOS: A new humanoid research platform targeted for industrial applications". In: *IEEE Humanoids 2017* () (cit. on pp. 36, 41).

[Ste94] A. Stentz. “Optimal and efficient path planning for partially-known environments”. In: (1994) (cit. on p. 19).

[Suh+22] Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. “Do Differentiable Simulators Give Better Policy Gradients?” In: (2022) (cit. on p. 10).

[Tan+18] Jie Tan et al. “Sim-to-Real: Learning Agile Locomotion For Quadruped Robots”. In: *CoRR* (2018) (cit. on p. 11).

[TZS+05] Russ Tedrake, Teresa Weirui Zhang, H Sebastian Seung, et al. “Learning to walk in 20 minutes”. In: 2005 (cit. on p. 11).

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: (2012) (cit. on p. 10).

[TCG12] Wouter G. van Toll, Atlas F. Cook IV, and Roland Geraerts. “Real-time density-based crowd simulation”. In: *Computer Animation and Virtual Worlds* (2012) (cit. on pp. 19–21).

[Tol+18] Wouter Van Toll, Atlas F. Cook Iv, Marc J. Van Kreveld, and Roland Geraerts. “The Medial Axis of a Multi-Layered Environment and Its Application as a Navigation Mesh”. In: *ACM Trans. Spatial Algorithms Syst.* (2018) (cit. on p. 19).

[TCG11] Wouter van Toll, Atlas F. Cook, and Roland Geraerts. “Navigation meshes for realistic multi-layered environments”. In: (2011) (cit. on p. 19).

[TP21] Wouter van Toll and Julien Pettré. “Algorithms for Microscopic Crowd Simulation: Advancements in the 2010s”. In: *Computer Graphics Forum* (2021). URL: https://hal.inria.fr/hal-03197198 (cit. on pp. 18, 22).

[Tol+16] Wouter van Toll et al. “A Comparative Study of Navigation Meshes”. In: 2016 (cit. on p. 19).

[Ton+15] S. Tonneau, N. Mansard, C. Park, D. Manocha, F. Multon, and J. Pettre. “A Reachability-Based Planner for Sequences of Acyclic Contacts in Cluttered Environments”. In: *ISRR* (2015) (cit. on pp. 3, 16, 23, 25, 28, 29, 93).

[Ton15] Steve Tonneau. “Motion planning and synthesis for virtual characters in constrained environments”. PhD thesis. INSA de Rennes, 2015 (cit. on pp. 3, 15).

[Ton+18a] Steve Tonneau, Andrea Del Prete, Julien Pettré, Chonhyon Park, Dinesh Manocha, and Nicolas Mansard. “An Efficient Acyclic Contact Planner for Multiped Robots”. In: *IEEE Transactions on Robotics* (2018) (cit. on pp. 3, 16, 23, 25, 29, 30, 33, 43, 52, 53, 56, 61, 64, 65, 69).

[Ton+18b] Steve Tonneau, Pierre Fernbach, Andrea Del Prete, Julien Pettré, and Nicolas Mansard. “2PAC: Two-Point Attractors for Center Of Mass Trajectories in Multi-Contact Scenarios”. In: *ACM SIGGRAPH* (2018) (cit. on pp. 6, 97, 98).

[Ton+20] Steve Tonneau, Daeun Song, Pierre Fernbach, Nicolas Mansard, Michel Taïx, and Andrea Del Prete. “SL1M: Sparse L1-norm Minimization for contact planning on uneven terrain”. In: *IEEE ICRA* (2020) (cit. on pp. 9, 15, 17, 72, 84, 88, 97, 98).

[Tra+22] Marco Tranzatto et al. *Team CERBERUS Wins the DARPA Subterranean Challenge: Technical Overview and Lessons Learned*. 2022 (cit. on pp. 1, 24).

[Tso+20] Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. "DeepGait: Planning and Control of Quadrupedal Gaits Using Deep Reinforcement Learning". In: *IEEE-RAL* (2020) (cit. on pp. 7, 14, 32, 49).

[TTO01] K. Tsujita, K. Tsuchiya, and A. Onat. "Adaptive gait pattern control of a quadruped locomotion robot". In: (2001) (cit. on p. 8).

[UM21] Adarsh Jagan Sathyamoorthy Utsav Patel Nithish Sanjeev Kumar and Dinesh Manocha. "DWA-RL: Dynamically Feasible Deep Reinforcement Learning Policy for Robot Navigation among Mobile Obstacles". In: *IEEE ICRA* (2021) (cit. on pp. 21, 22).

[Vai+14] Joris Vaillant et al. "Vertical ladder climbing by the HRP-2 humanoid robot". In: *IEEE International Conference on Humanoid Robots* (2014) (cit. on p. 7).

[Vec+17] Matej Vecerik et al. "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards". In: *CoRR* (2017) (cit. on pp. 69, 70).

[Ver+09] Paul Vernaza, Maxim Likhachev, Subhrajit Bhattacharya, Sachin Chitta, Aleksandr Kushleyev, and Daniel D. Lee. "Search-based planning for a legged robot over rough terrain". In: *IEEE ICRA* (2009) (cit. on p. 14).

[VB04] Miomir Vukobratovic and Branislav Borovac. "Zero-Moment Point - Thirty Five Years of its Life." In: *International Conference on Humanoid Robotics (Humanoids)* (2004) (cit. on p. 8).

[Wal+07] Thomas J. Walsh, Ali Nouri, Lihong Li, and Michael L. Littman. "Planning and Learning in Environments with Delayed Feedback". In: 2007 (cit. on p. 34).

[WFH09] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. "Optimizing Walking Controllers". In: *ACM SIGGRAPH* (2009) (cit. on p. 10).

[Wan+12] Jack M. Wang, Samuel R. Hamner, Scott L. Delp, and Vladlen Koltun. "Optimizing Locomotion Controllers Using Biologically-Based Actuators and Objectives". In: *ACM SIGGRAPH* (2012) (cit. on p. 10).

[WH21] Lorenz Wellhausen and Marco Hutter. "Rough Terrain Navigation for Legged Robots Using Reachability Planning and Template Learning". In: *IEEE IROS* (2021) (cit. on pp. 16, 23, 24).

[Wer+21] Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C. Karen Liu. "Fast and Feature-Complete Differentiable Physics for Articulated Rigid Bodies with Contact". In: *CoRR* (2021) (cit. on p. 10).

[Wer+16] Martin Wermelinger, Péter Fankhauser, Remo Diethelm, Philipp Krüsi, Roland Siegwart, and Marco Hutter. "Navigation planning for legged robots in challenging terrain". In: *IEEE IROS* (2016) (cit. on pp. 16, 24).

[Wes+07] E.R. Westervelt, J.W. Grizzle, C. Chevallereau, J.H. Choi, and B. Morris. *Feedback Control of Dynamic Bipedal Robot Locomotion*. Taylor and Francis/CRC, 2007 (cit. on p. 6).

[WTK16] Pierre-Brice Wieber, Russ Tedrake, and Scott Kuindersma. *Modeling and Control of Legged Robots*. Ed. by Bruno Siciliano and Oussama Khatib. 2016 (cit. on p. 8).

[Wij+19] Erik Wijmans et al. "Decentralized Distributed PPO: Solving PointGoal Navigation". In: *CoRR* (2019) (cit. on p. 41).

[Win+18] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli. "Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization". In: *IEEE-RAL* (2018) (cit. on p. 9).

[Win+17] Alexander W. Winkler, Farbod Farshidian, Diego Pardo, Michael Neunert, and Jonas Buchli. "Fast Trajectory Optimization for Legged Robots using Vertex-based ZMP Constraints". In: *CoRR* (2017) (cit. on pp. 9, 13).

[Win+15] Alexander W Winkler, Carlos Mastalli, Ioannis Havoutis, Michele Focchi, Darwin G. Caldwell, and Claudio Semini. "Planning and Execution of Dynamic Whole-Body Locomotion for a Hydraulic Quadruped on Challenging Terrain". In: *IEEE ICRA* (2015) (cit. on p. 16).

[Win+14] Alexander Winkler et al. "Path planning with force-based foothold adaptation and virtual model control for torque controlled quadruped robots". In: (2014) (cit. on pp. 16, 24, 25).

[WGH22] Jungdam Won, Deepak Gopinath, and Jessica Hodgins. "Physics-Based Character Controllers Using Conditional VAEs". In: *ACM SIGGRAPH* (2022) (cit. on pp. 10, 12).

[Wu+18] Keyu Wu, Mahdi Abolfazli Esfahani, Shenghai Yuan, and Han Wang. "Learn to Steer through Deep Reinforcement Learning". In: *Sensors* (2018) (cit. on p. 22).

[Xie+21] Zhaoming Xie, Xingye Da, Buck Babich, Animesh Garg, and Michiel van de Panne. "GLiDE: Generalizable Quadrupedal Locomotion in Diverse Environments with a Centroidal Model". In: *CoRR* (2021) (cit. on p. 7).

[Xie+20] Zhaoming Xie, Hung Yu Ling, Nam Hee Kim, and Michiel van de Panne. "ALLSTEPS: Curriculum-driven Learning of Stepping Stone Skills". In: *ACM SIGGRAPH* (2020) (cit. on p. 7).

[Yan+16] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. "Survey of Robot 3D Path Planning Algorithms". In: *Journal of Control Science and Engineering* (2016) (cit. on p. 20).

[Yan+22] Tsung-Yen Yang, Tingnan Zhang, Linda Luu, Sehoon Ha, Jie Tan, and Wenhao Yu. "Safe Reinforcement Learning for Legged Locomotion". In: *arXiv* (2022) (cit. on pp. 11, 12).

[YLP07] KangKang Yin, Kevin Loken, and Michiel van de Panne. "SIMBICON: Simple Biped Locomotion Control". In: *ACM SIGGRAPH* (2007) (cit. on p. 10).

[Yos+05] Eiichi Yoshida, I. Belousov, Claudia Esteves, and Jean-Paul Laumond. In: *International Conference on Humanoid Robots* (2005) (cit. on p. 16).

[Yu+19] Wenhao Yu, Visak C. V. Kumar, Greg Turk, and C. Karen Liu. "Sim-to-Real Transfer for Biped Locomotion". In: *CoRR* (2019) (cit. on p. 11).

[ZK18] Christian Zammit and Erik-Jan van Kampen. “Comparison between A* and RRT Algorithms for UAV Path Planning”. In: *AIAA Guidance, Navigation, and Control Conference* (2018) (cit. on p. 20).

[Zhe17] Kaiyu Zheng. “ROS Navigation Tuning Guide”. In: *CoRR* (2017) (cit. on pp. 18, 19, 21).

[Zho+19] Allan Zhou et al. “Watch, Try, Learn: Meta-Learning from Demonstrations and Reward”. In: *CoRR* (2019) (cit. on p. 70).

[Zuc10] Matthew Zucker. “Learning and Optimization Methods for High Level Planning”. PhD thesis. Carnegie Mellon University, 2010 (cit. on p. 14).