Learning to steer a locomotion contact planner

Jason Chemin, Pierre Fernbach, Daeun Song, Guilhem Saurel, Nicolas Mansard, Steve Tonneau

Learning to steer a locomotion contact planner

Jason Chemin[1,*], Pierre Fernbach[2], Daeun Song[3], Guilhem Saurel[1], Nicolas Mansard[1,4] and Steve Tonneau[5,1]

***Abstract*— The combinatorics inherent to the issue of planning legged locomotion can be addressed by decomposing the problem: first, select a guide path abstracting the contacts with a heuristic model; then compute the contact sequence to balance the robot gait along the guide path. While several models have been proposed to compute such a path, none have yet managed to efficiently capture the complexity of legged locomotion on arbitrary terrain. In this paper, we present a novel method to automatically build a local controller, or steering method, to generate a guide path along which a feasible contact sequence can be built. Our reinforcement learning approach is coupled with a geometric condition for feasibility during the training, which improves the convergence rate without incurring a loss in generality. We have designed a dedicated environment along with an associated reward function to run a classical reinforcement learning algorithm that computes the steering method. The policy takes a target velocity and a local heightmap of the terrain around the robot as inputs, and steers the path where new contacts should be created. It is then coupled with a contact generator that creates the contacts to support the robot movement. We show that the trained policy has an improved generalization and higher success rate at generating feasible contact plans than previous approaches. As a result, this policy can be used with a path planning algorithm to navigate complex environments.**

I. Introduction

The motivation for building legged robots lies in their ability to navigate across arbitrarily complex environments, as well or better than humans or other animals. To this date, this superiority is only potential: efficient methods exist to control a robot over flat terrain [1], but in spite of significant successes [2]–[5], the quest for a method able to robustly and interactively generate complex motions in unforeseen contexts remains active [6]. The reasons for this struggle are well known: legged robots are high-dimensional systems, and their motion is subject to non-linear dynamic and geometric constraints that change discretely with the choice of contacts, thus introducing a combinatorial aspect to the problem.

A common approach to tackle the problem is to decompose it into simpler sub-problems sequentially solved [4], [7]–[9]. The first sub-problem consists of generating a low-dimensional guide path for the root of the robot to a desired goal, where collisions and dynamic constraints are approximated according to a heuristic for feasibility. The second sub-problem consists of extending this guide path into a contact plan, composed of a discrete sequence of key contact postures. This strategy reduces the search space for contact locations to the set of surfaces reachable from discrete positions along the path and results in a fast and interactive contact planning algorithm. Given the contact sequence, efficient methods exist to interpolate the sequence into a whole-body motion for the robot [5], [10], [11].

In such a sequential decomposition, the feasibility constraints for next sub-problem are merely approximated: this divide-and-conquer approach can thus result in failure to plan feasible contact sequences. To improve the success rate of such a framework, we must refine our answer to the question: *What is a "good" guide path for the contact planner*? Previous locomotion planners, such as [7], often used heuristic approximations of the necessary and sufficient condition for a guide path to be feasible.

In this paper, we propose to approximate the feasibility condition with reinforcement learning. Concretely, we train a policy using a local heightmap to navigate complex environments and generate a guide path. This guide path is given to the contact generator that outputs a sequence of contact configurations along with an evaluation of its success. The RL algorithm learns to maximize the success rate, thus learning what is a good guide path for this contact planner. We demonstrate the capability of our steering method, LeaS (LEArn to Steer), coupled with a contact planner. LeaS can generalize to various unknown terrains and we compare its performance with our previous methods. Our results show that guide paths generated by LeaS avoid obstacles and collisions, and are suitable for the contact planner to compute contact sequences.

II. Related work

Locomotion has been explored by graphics and robotics communities for different types of characters and robots [12]–[14]. In this paper, we focus on contact planning for legged-character locomotion to navigate complex terrains while avoiding collisions and ensuring the feasibility of the movement.

1) Contact planning in robotics: Common methods for solving the contact planning problem are introduced by [3]. The first is the so-called "contact-before-motion" approach that aims to produce a contact sequence before generating the whole-body movement on the robot. This category regroups work in which a sequence of discrete contacts can be given

[1]LAAS-CNRS, Univ. Toulouse, CNRS, France
[2]Toward, Toulouse, France
[3]Dept. of CSE, Ewha Womans University, Korea
[4]Artificial and Natural Intelligence Toulouse Institute, France
[5]IPAB, The University of Edinburgh, Scotland
*corresponding author: jchemin@laas.fr

This work has been supported by the MEMMO European Union project within the H2020 Program under Grant Agreement No. 780684.

D. Song is supported in part by the ITRC/IITP program (IITP-2021-0-01460) and the NRF(2017R1A2B3012701) in South Korea

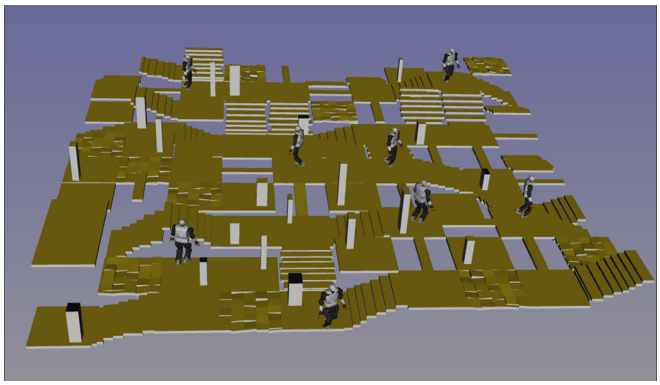


Fig. 1: The locomotion arena, a large stuctured simulation where our Talos agent learns in which direction to steer in order to generate stable contacts.

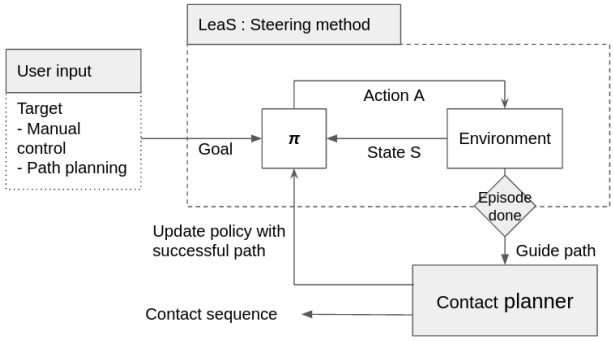


Fig. 2: System overview. The user inputs a target velocity to LeaS which computes the guide path. Once the episode is over (III-D), LeaS outputs the path to the contact planner to compute a contact sequence along it. During training, we update the policy with the successful path.

by the user [1], [15], retrieved from a database [16] or found using search algorithms or machine learning [4], [10], [17]–[21]. An alternative approach is to use continuous optimization instead of the discrete contact selection problem [22]–[24]. The second strategy is the "motion-before-contact" that first plans a guide path to follow (i.e. a root trajectory), then finds a contact sequence along it, and finally computes the whole-body motion from these contacts [8], [25], [26]. This approach greatly reduces the combinatorics on complex terrains at the cost of not guaranteeing a globally optimal solution. Reproducing complex movement behaviors, [27] finds the contacts and whole-body motion simultaneously thus improving the success of the motion, but at the cost of heavy computation.

2) Machine learning for legged-characters locomotion: Machine learning has recently brought interesting advances in locomotion. Data-driven approaches to learn whole-body controllers [28]–[31] produce natural and smooth locomotion (although non-dynamically consistent) that can generalize to unknown terrains using more commonly a height map for locomotion or a voxel map for environment interactions. Recent improvements in Reinforcement Learning (RL) [32], [33] have opened new possibilities in physics-based simulations to learn locomotion from scratch [34]–[36] and from motion data by imitation learning to improve the plausibility of movements [37]–[39]. However, learning a whole-body controller directly on a robot in the real world is not possible in practice due to the time of training required and the risk of breaking the robot. Learning in simulation and transferring the policy to the real robot, also called "sim-to-real", remains a challenging problem for quadrupeds [40] and bipeds [41].

More related to "contact-before-motion" approaches in robotics, DeepLoco [19] learns a contact planner that generates a sequence of footsteps fitting a learned whole-body controller to reach a goal. Following this architecture, DeepGait [42] learns two controllers separately, one for contact planning and the second to output target joint positions to move a quadruped robot.

In our work, we decide to focus on the contact planning problem and learn by reinforcement a higher-level controller as in [19], [42]. The difference with these works is our "motion-before-contact" approach where we learn to generate the guide path as the input to a given contact planner.

3) Steering methods: We define a *steering method* as a local planner which links an initial configuration to a goal in an obstacle-free space. Our previous work [26] uses a rapidly-exploring random tree for path planning to find waypoints leading to a goal and a linear interpolation as steering method to link these waypoints. In an extension of this work [8], a kinodynamic steering-method is introduced to produce dynamically feasible paths. In a similar spirit to our paper, [43] learns by reinforcement both the local steering method for obstacle avoidance and the control of a wheeled robot, then combines it with a sample-based path planning algorithm to navigate complex environments.

Our contribution is to learn such a local steering method and to couple it with a contact planner to improve their synergy, quantifiable in terms of success rate.

III. Learning the Steering Method

A. Overview

Reinforcement Learning can be defined as a Markov Decision Process (S, A, P, R, γ) where S is a set of states, A a set of discrete or continuous actions, $P(s_t, a_t, s_{t+1})$ is a transition probability $p(s_{t+1}|s_t, a_t)$, R is a reward function mapping $S \times A \rightarrow \Re$, and $\gamma \in [0, 1)$ a discount factor. The agent takes some actions according to the policy π in function of the actual state $a_t = \pi(s_t)$. We want to learn a policy that tries to maximize the future cumulative rewards $\mathbb{E}[\sum_{k=t}^{\infty} \gamma^k r_{t+k}]$. In this work, we use the RL algorithm "Proximal policy optimization" (PPO) [44].

During the training, at each step, we predict the next action to perform, execute it in the environment, and repeat the process until the episode is over as shown on Fig. 2. At the end of an episode, we obtain a sequence of 6D robot positions that represents the guide path, which is then given to a contact planner to compute contact configurations along it. We learn to provide guide path fitting the contact planner by discarding the part of the path that results in failure.

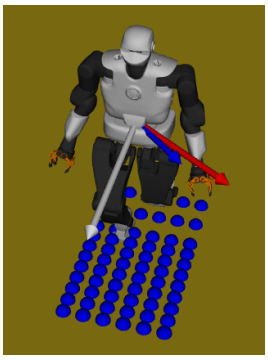


Fig. 3: Observable state with root orientation (**white arrow**), root velocity (**Blue arrow**), the desired target velocity and orientation (**Red arrow**) and the local height map (**Blue dots**).

B. States

The observable state is a set, $S = \{v_o, o_{target}, h_o\}$, where v_o is the velocity of the robot root relative to its orientation, o_{target} is the angle difference between its orientation and the desired target, and h_o is the local heightmap relative to its orientation. The states and dimensions of the heightmap can be seen on Fig. 3. The dimensions of the heightmap are: 9 values in front of the robot, 2 values in its back and 7 values from left to right with a discretization of 8cm, 20cm and 15cm respectively.

The heightmap is small on purpose as we desire a local steering method as described in II-.3. Increasing the height map size leads to over-fitting on our fixed training terrain and path planning instead of a local steering method.

C. Actions

Our policy returns a set of actions $A = \{a_x, a_y, a_z, \omega\}$ with a_x, a_y, a_z the accelerations of the robot on each axis and ω the angular velocity of its orientation. At each step i, the robot position R_{pos}, velocity R_{vel} and orientation R_{ori} are modified as follows :

(1) $R^i_{pos} = R^{i-1}_{pos} + R^{i-1}_{vel} * timestep$
(2) $R^i_{vel} = R^{i-1}_{vel} + [a_x, a_y, a_z] * timestep$
(3) $R^i_{ori} = R^{i-1}_{ori} + \omega * timestep$

where timestep is a constant (Table I). If the velocity is constant, the longer the timestep, the fewer configurations there are on the guide for given initial and goal configurations. The choice of this value depends on the contact planner used. Empirically, we know that the contact-generation algorithm [7] has higher success rates with timestep values between $[0.01s, 0.10s]$ for velocity norms inferior to $0.18m/s$.

D. Termination conditions

We consider an episode to be over when the maximum number of steps is reached or when the current configuration is invalid. When executing a trained policy, we also consider an episode to be finished when the root position of the robot has reached the target $d(root, target) < \epsilon$ with ϵ a constant. The validity of a configuration is done by checking if there is no obstacle in collision with the robot root and if it can touch the ground by looking at the *reachability condition* [7]. This condition is necessary but not sufficient for the contact planner to find a contact from this root configuration.

E. Rewards

We want a steering method able to (1) move the robot in the desired direction at the right velocity, (2) orientate the robot in this same direction, (3) keep a valid configuration even when blocked by obstacles, (4) generate a smooth path and (5) make this guide path feasible by the contact planner.

The reward for (1) punishes the agent for not going in the desired direction at the right velocity : $R_{dir} = -(||\overrightarrow{v} - \overrightarrow{v*}||/(2v_{max}))^2$ with $\overrightarrow{v}$ the vector velocity, $\overrightarrow{v*}$ the desired vector velocity and v_{max} its maximum norm. The reward for (2) punishes the agent for not being oriented toward the target : $R_{ori} = -(1 - \overrightarrow{u}_{ori} \cdot \overrightarrow{u}_{target})$ with $\overrightarrow{u}_{ori}$ the unit vector representing the orientation of the root and $\overrightarrow{u}_{target}$ the desired direction of movement. The reward for (3) encourages the agent to follow its path until the maximum number of steps in the episode is reached with $R_{alive} = 1$. The two rewards for (4) punishes the agent for taking too large actions : $effort_{orientation} = -|\omega/\omega_{max}|^2$ with ω the action on orientation and ω_{max} the maximum angular velocity, and $effort_{acceleration} = -(||[a_x, a_y, a_z]||/a_{max})^2$ with $[a_x, a_y, a_z]$ the action taken on acceleration and a_{max} its maximum norm. There is no reward for (5) as it corresponds to the discarding of the failed part of the guide path as explained in III-A . The resulting reward is : $R = R_{dir}w_{dir} + R_{ori}w_{ori} + effort_{ori}w_{eff\_ori} + effort_{acc}w_{eff\_acc} + R_{alive}w_{alive}$ with w some weights (see Table I).

The design of a reward for the behaviors (1), (2), (4) and (5) is straightforward and we could obtain it using a positive reward for (1) and negative rewards for the rest. The difficulty is to insert the condition (3) to make the robot keep a valid configuration, even when not able to go further. When the robot approaches an obstacle that it cannot avoid, we want the robot to stop moving, and thus, it needs to have a positive reward keeping it from terminating the episode. We opt for a simple way to solve this problem by adding a constant positive reward R_{alive} at each step.

F. Implementation

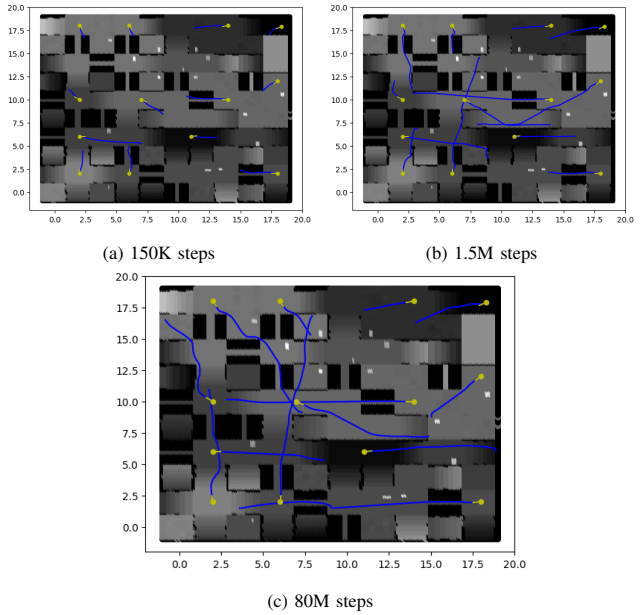
Optimized asynchronous versions of PPO, such as IMPALA [45], are available. In future work, we plan to integrate it or investigate the use of off-policy algorithms. In this paper, we use our own asynchronous version of PPO that separates contact planning from acting, using a Master-Minion architecture (`http://cpc.cx/ucS`).

IV. Results

We used the HPP planner [46], [47] and the Talos model [48]. Our algorithm was implemented in Python using the PPO implementation of Stable Baselines [49], modified for our parallel implementation (see Sec. III-F). The simulation was run on a PC with an Intel Core i7-8700 (12 cores, 3.20Ghz). Our network is fully-connected with two hidden layers of 64 units with Tanh activation function as was

TABLE I: Parameters

State dimensions	81	Actions dimension	4
Parallel actors	6	Number of minions	7
Batch size	$2048 * 6$	Mini-Batches	256
Noptepochs	10	Max Episode Length	400
Discount Factor	0.97	Clip range	0.2
Learning rate	$5.0e-4$	timestep	$0.10s$
$w_{direction}$	1.0	$w_{orientation}$	0.4
$w_{eff\_orientation}$	0.1	$w_{eff\_acceleration}$	0.1
w_{alive}	1.0		



(a) 150K steps (b) 1.5M steps

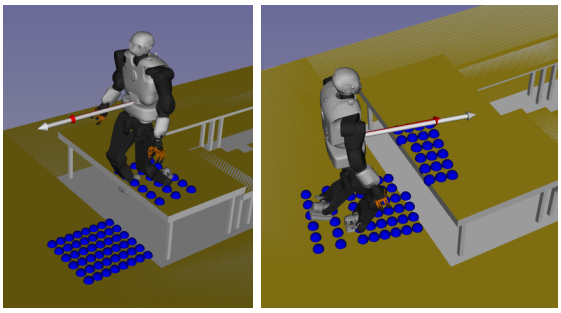
(c) 80M steps

Fig. 4: Paths after 150K, 1.5M and 80M steps of learning on a 10x10 grid training terrain with some initial positions and target directions (**Yellow**) and their trajectories (**Blue**).

used in [44]. We trained our policy for 53 hours (80 millions steps) using the parameters in Table I, with an architecture of 6 actors in parallel and 7 external minions validating the paths with the contact planner [7]. The training terrain is a 10x10 grid ($20 \times 20m$) where each cell corresponds to a type of terrain as depicted by Fig. 1. During the training, the initial position, orientation and velocity of the robot are set randomly on the training ground. The target direction is set randomly on the horizontal plane and remains constant for a path. For training and comparison purposes, we set the norm of the target velocity to $0.18m/s$ to balance the path lengths and the contact planner performances (see Sec. III-C). To perform our comparisons, LeaS takes deterministic actions on another terrain never met during the training. The dimension of stairs, obstacles and beams are slightly different from those used during the training.

A. Learning progression

We evaluate the learning progress by plotting trajectories obtained with a policy at different stages of the training as seen on Fig. 4. We want our policy to follow a direction at the desired velocity as far as possible, while ensuring the success of the contact planner on this guide path. The policy in the early phase of the training makes our robot go toward



(a) (b)

Fig. 5: The robot cannot go further as an obstacle is detected on its local height map (**Blue dots**) in the desired direction (**Red arrow**) so the policy makes the robot stop.

Parameters	RBLin	RBKino	LeaS
$-30°$ to $30°$			
$\|\|v\|\|$ = 0,01	99.8%	99.7%	99.8%
$\|\|v\|\|$ = 0,04	99.8%	99.7%	99.8%
$\|\|v\|\|$ = 0,07	99.8%	99.7%	99.8%
$60°$ to $120°$			
$\|\|v\|\|$ = 0,01	99.8%	99.7%	99.8%
$\|\|v\|\|$ = 0,04	97.3%	99.7%	99.8%
$\|\|v\|\|$ = 0,07	99.8%	99.7%	99.8%
$150°$ to $210°$			
$\|\|v\|\|$ = 0,01	22.3%	76.5%	98.4%
$\|\|v\|\|$ = 0,04	28.9%	77.6%	93.1%
$\|\|v\|\|$ = 0,07	34.0%	85.5%	98.2%

TABLE II: Success rate of the steering methods with the contact planner on flat ground trajectories of 4 meters with different initial orientations and velocities (60 trajectories for each test)

the target but does not consider the surrounding terrain and fails to produce trajectories with a contact sequence. During the training, the policy learns how to generate longer trajectories in this direction while detecting the different types of obstacles within its local heightmap. Our policy also learns how to stop when it cannot go further, as demonstrated in Fig. 5 in an unknown environment.

B. Comparison of steering methods

In this section, we compare LeaS to RBLin [7] and RBKino [8], to perform a local task: given an initial configuration, the goal is to reach a target position by producing a guide path, which is given to the contact planner in order to generate the contact sequence. As our policy was not trained for accurately reaching a target, we consider that the goal is reached if a contact configuration is within a 30cm radius around it. In all our scenarios, the initial velocity direction of the robot is the same as its orientation. The timestep and the norm of target velocities are the same for all three methods.

*1) **Ground**:* Given an initial and target position, we set different initial orientations and velocities to evaluate the success of all the steering methods with the contact planner. The success value in Table II corresponds to the percentage of the path covered by the contact sequence. The orientation values correspond to the angle between the initial target direction and the root orientation of the robot. Results show that for all orientations going from 0 to 120 degrees, all

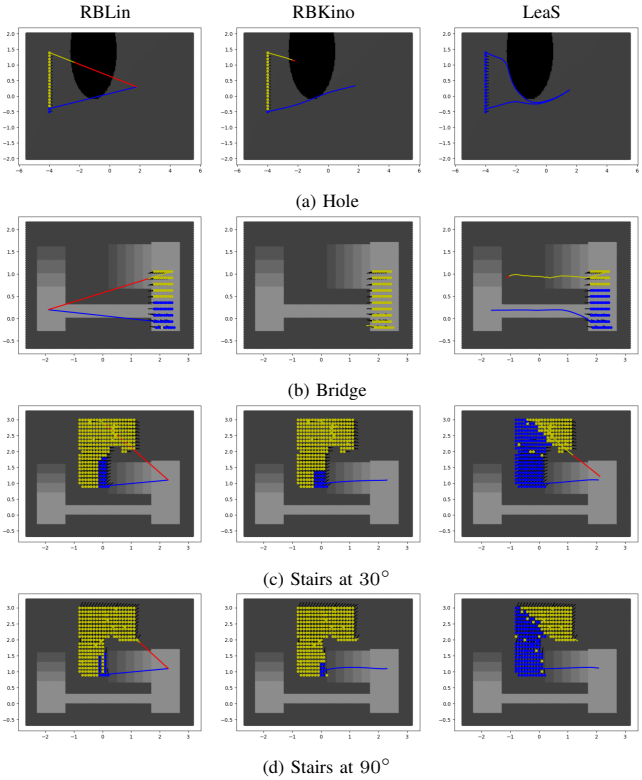


Fig. 6: Trajectories of steering methods with: valid path to target (**Blue**), failed path to target (**Yellow**), failed part of a path (**Red**) and initial orientation (**Black Arrow**). Pictures link : `http://cpc.cx/ubV`

three methods succeed in reaching the target. But when the robot has its back turned to the target, the success rates of both previous methods drop. RBLin rotates in a fixed number of steps that needs to be tuned (here 100 steps) and fails with rotations of high angular velocity. RBKino gets better results than RBLin when facing backward, but trajectories generated still have a low success rate with our contact planner. We also notice that with both methods, raising the initial velocity improves its success rate. In contrast, our method LeaS performs well in generating valid trajectories, for any given initial velocity and orientation.

*2) **Hole**:* In the hole scenario (Fig. 6a), some initial configurations are sampled online and the target is placed on the other side of the hole. The difficulty of this scenario is to have paths away from the edge of the hole to ensure the success of the contact planner. RBLin does not perform well in this scenario where most of linear paths lead inside the hole. The same goes for RBKino, which in most cases cannot generate a complete path leading to the target. In order to succeed in this scenario, both of these steering methods require path planning and intermediary waypoints to the target. In contrast, LeaS succeeds in generating a valid guide path leading to the target while keeping a safe distance from the hole. This means that configurations on the guide do not lead to a funambulist walk, leading to a low success rate of the contact planner.

*3) **Bridge**:* In this scenario (Fig. 6b), the goal is to make the robot cross a 30cm-wide bridge to reach a target placed on the other side. As for the hole scenario, one of the difficulties is to make the robot stay away from the edges. RBKino fails in finding a complete path. Thus the contact planner is unable to provide a contact sequence up to the target. RBLin succeeds in reaching the goal from all the blue dots, but the contact planner can have problems in finding contact steps on paths very close to the edges. By contrast, LeaS succeeds in crossing the bridge while staying in the middle as long as it appears in its vision field. As our policy has been trained to follow a direction with a reduced local heightmap, the yellow dots show the cases in which LeaS has the stairs in its vision field but not the bridge. In these initial configurations, LeaS moves in the target direction and goes down the stairs. Finally, it detects a platform too high to reach and makes the robot stop moving. This is an implementation choice as discussed in III-B.

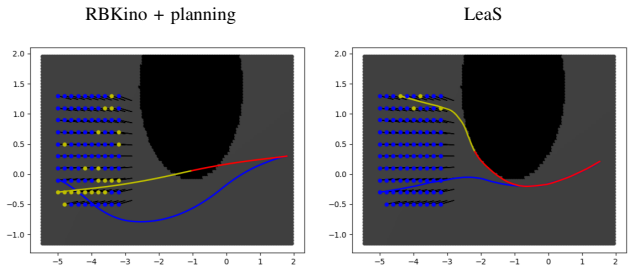
*4) **Stairs**:* In the stairs scenario (Fig. 6c and 6d), the goal is to climb the stairs to reach the target at the top. The difficulty of this scenario is to find the right height at which to engage the stairs. Empirically, we know that the contact planner has difficulties when guide paths climbing the stairs are too close or far from the ground. As a result, it cannot find a contact transition in equilibrium and fails the path. The result of RBLin shows the difficulty to climb the stairs while keeping the right height from the ground, thus leading to a reduced zone of initial configurations with valid paths. This zone is smaller for RBKino, which has difficulties in connecting the initial and target configuration. LeaS succeeds in engaging the stairs and climbing to the target as long as the beginning of the stairs is detected in its local heightmap.

We make another comparison with two different initial angles. When the robot starts with an orientation perpendicular to the target direction (Fig. 6d), the number of successful guide paths for all three methods is reduced. With our method, the policy has to rotate the robot before detecting the stairs, and that can explain the success difference between Fig. 6c and 6d.

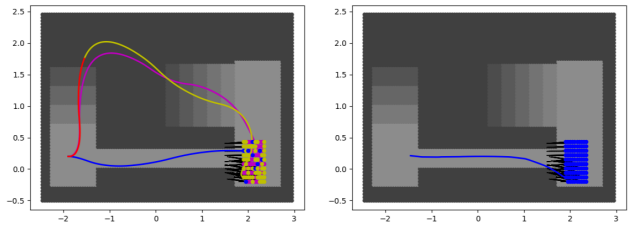
In conclusion, LeaS shows better results with the contact planner in all our scenarios. However, this does not guarantee its success, as we can see for example in Fig. 6d, where it can fail when starting perpendicular to the stairs. In this example, the guide path is valid and reaches the target, but we can see a limitation of the contact planner, which can fail to find a contact sequence.

C. Comparison to RBKino path planning

In the previous comparison, we were only using the steering methods and have shown that LeaS is always better than the heuristic controllers implemented in the planner [7] and [8]. Results in IV-B show that RBKino alone is too local to be fairly compared with other methods. That is why we use RBKino with its path planner, which stochastically explores waypoints to simplify the task of the local controller and maximizes its convergence. We perform a comparison with LeaS for the hole and bridge scenarios, where RBKino failed without path planning. No path planning strategy is used for



(a) Hole



(b) Bridge

Fig. 7: Trajectories of RBKino with path planning and LeaS with : valid trajectory to target (**Blue**), failed trajectory to target (**Yellow**), failed part of trajectory (**Red**) and path valid but not optimal by not taking the bridge (**Magenta**).

our policy, which remains a purely local method. Results for both scenarios are shown in Fig. 7.

*1) **Hole**:* The difficulty of this scenario is explained in IV-B.2: a trajectory too close to the hole would lead our robot to move like a tightrope walker on the edge of the hole. LeaS still performs better on average with fewer failures than RBKino, whose path planning finds some trajectories too close to the edge.

*2) **Bridge**:* In this scenario, RBKino with path planning reaches the target but most of these trajectories do not pass by the bridge and take the stairs instead. This scenario for RBKino shows both the difficulty to engage stairs with the right height and the non-optimality of its path planner. In comparison, our policy succeeds in taking the bridge and in reaching the target as in IV-B.3.

D. Path planning with LeaS

We evaluate the capability of our method to follow some waypoints. In the first test, we use our method combined with a path planning algorithm. Then we manually define some waypoints to make our robot cross an evaluation terrain. In both scenarios, we consider that a target waypoint is reached with LeaS when the distance separating our robot to this point is inferior to the threshold. We then set the target on the next waypoint.

1) Replace RBKino: The path planning of RBKino generates intermediary waypoints to be connected by the local steering method. In this test, we replace this steering method with ours. The results in Fig. 8 show that our local steering method can be used to follow these waypoints and compares well with the RBKino.

2) Manually defined circuit: Fig. 9 shows some manually placed waypoints that our robot has to follow. We start the path on the red dot with an orientation indicated by the black

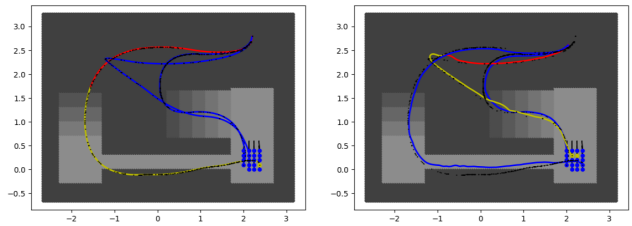


Fig. 8: Trajectories following waypoints made by the path planning of RBKino (**left**) and LeaS following it (**right**) with: valid path to target (**Blue**), failed path to target (**yellow**), failed part of the path (**Red**) and the waypoints to follow (**Black**).

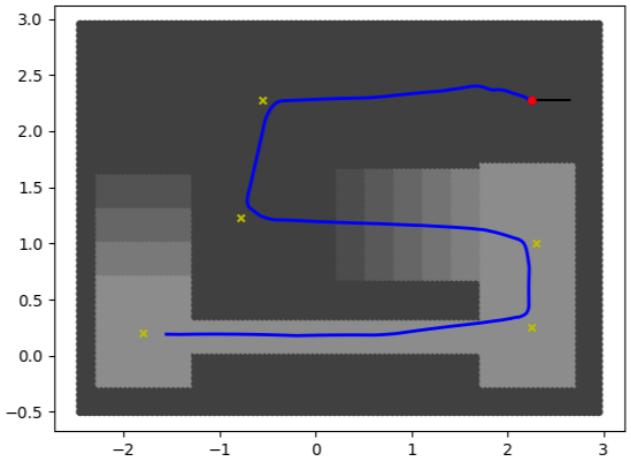


Fig. 9: LeaS following manually placed waypoints with : the successful path (**Blue**), the waypoints to reach (**Yellow**) and its initial orientation (**Black arrow**).

arrow. LeaS succeeds in rotating the robot toward the next waypoint and in reaching all of them one by one.

V. Conclusion

We proposed LeaS, a new approach for learning a steering method that generates a root path for humanoid robots. Our steering method is trained to fit a contact planner and to generate feasible paths for it. Our method can generalize to unknown environments and performs better in all scenarios than our previous techniques with the contact planner. It can be used both as an improved steering method in the global locomotion planner, or as a local method to control the robot locomotion in real-time, e.g. in a navigation task. A drawback of LeaS is that its success depend on the chosen contact planner and timestep value. We plan to investigate this issue in the future. Our algorithm can be extended in several ways. Currently, we only train the policy to satisfy the (implicit) working assumptions of the contact generator. It should be possible to implement more restrictive constraints required to generate a complete motion of the whole-body. We would also like to directly exploit sensor measurements (e.g. lidar data) for the policy input, and then validate the approach on a real robot.

References

[1] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, “Biped walking pattern generation by using preview

control of zero-moment point," *IEEE International Conference on Robotics and Automation*, 2003.

[2] K. Hauser, T. Bretl, and J.-C. Latombe, "Non-gaited humanoid locomotion planning," in *IEEE-RAS International Conference on Humanoid Robots*, 2005.

[3] T. Bretl, "Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem," *The International Journal of Robotics Research*, vol. 25, pp. 317 – 342, 2006.

[4] A. Escande, A. Kheddar, and S. Miossec, "Planning contact points for humanoid robots," *Robotics and Autonomous Systems*, vol. 61, no. 5, pp. 428–442, 2013.

[5] B. Ponton, M. Khadiv, A. Meduri, and L. Righetti, "Efficient multi-contact pattern generation with sequential convex approximations of the centroidal dynamics," 2020.

[6] C. G. Atkeson, B. Babu, N. Banerjee, D. Berenson, C. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin *et al.*, "What happened at the darpa robotics challenge, and why," *submitted to the DRC Finals Special Issue of the Journal of Field Robotics*, vol. 1, 2016.

[7] S. Tonneau, N. Mansard, C. Park, D. Manocha, F. Multon, and J. Pettré, "A Reachability-based planner for sequences of acyclic contacts in cluttered environments," in *International Symposium on Robotics Research (ISSR)*, Sestri Levante, Italy, Sep. 2015.

[8] P. Fernbach, S. Tonneau, A. Del Prete, and M. Taïx, "A Kinodynamic steering-method for legged multi-contact locomotion," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vancouver, Canada, Sep. 2017, p. 7p.

[9] I. Kumagai, M. Morisawa, S. Hattori, M. Benallegue, and F. Kanehiro, "Multi-contact locomotion planning for humanoid robot based on sustainable contact graph with local contact modification," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6379–6387, 2020.

[10] J. Carpentier, S. Tonneau, M. Naveau, O. Stasse, and N. Mansard, "A versatile and efficient pattern generator for generalized legged locomotion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[11] P. Fernbach, S. Tonneau, and M. Taïx, "Croc: Convex resolution of centroidal dynamics trajectories to provide a feasibility criterion for the multi contact planning problem," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–9.

[12] Z. Qiu, A. Escande, A. Micaelli, and T. Robert, "Human motions analysis and simulation based on a general criterion of stability," in *International Symposium on Digital Human Modeling*, 2011.

[13] A. W. Winkler, F. Farshidian, M. Neunert, D. Pardo, and J. Buchli, "Online walking motion and foothold optimization for quadruped locomotion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[14] J. Won, J. Park, K. Kim, and J. Lee, "How to train your dragon: Example-guided control of flapping flight," *ACM Trans. Graph.*, vol. 36, no. 6, Nov. 2017.

[15] S. Caron, Q.-C. Pham, and Y. Nakamura, "Leveraging cone double description for multi-contact stability of humanoids with applications to statics and dynamics," in *Robotics: Science and System*, 2015.

[16] S. Agrawal and M. van de Panne, "Task-based locomotion," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 35, no. 4, 2016.

[17] K. Hauser, T. Bretl, J. Latombe, K. Harada, and B. Wilcox, "Motion planning for legged robots on varied terrain," *International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1325–1349, Nov. 2008.

[18] D. Kanoulas, A. Stumpf, V. S. Raghavan, C. Zhou, A. Toumpa, O. Von Stryk, D. G. Caldwell, and N. G. Tsagarakis, "Footstep planning in rough terrain for bipedal robots using curved contact patches," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[19] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 36, no. 4, 2017.

[20] S. Tonneau, P. Fernbach, A. D. Prete, J. Pettré, and N. Mansard, "2pac: Two-point attractors for center of mass trajectories in multi-contact scenarios," *ACM Trans. Graph.*, vol. 37, no. 5, Oct. 2018.

[21] J. Englsberger, C. Ott, and A. Albu-Schäffer, "Three-dimensional bipedal walking control based on divergent component of motion," *IEEE Transactions on Robotics*, 2015.

[22] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *IEEE-RAS International Conference on Humanoid Robots*, 2014.

[23] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, 2018.

[24] S. Tonneau, D. Song, P. Fernbach, N. Mansard, M. Taïx, and A. D. Prete, "Sl1m: Sparse l1-norm minimization for contact planning on uneven terrain," *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[25] K. Bouyarmane, A. Escande, F. Lamiraux, and A. Kheddar, "Potential field guide for humanoid multicontacts acyclic motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[26] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 586–601, Jun. 2018.

[27] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, no. 4, Jul. 2012.

[28] S. Starke, H. Zhang, T. Komura, and J. Saito, "Neural state machine for character-scene interactions," *ACM Trans. Graph.*, vol. 38, no. 6, Nov. 2019.

[29] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017.

[30] H. Zhang, S. Starke, T. Komura, and J. Saito, "Mode-adaptive neural networks for quadruped motion control," *ACM Trans. Graph.*, vol. 37, no. 4, Jul. 2018.

[31] F. G. Harvey and C. Pal, "Recurrent transition networks for character locomotion," in *SIGGRAPH Asia Technical Briefs*. Association for Computing Machinery, 2018.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, 2013.

[33] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, 2015.

[34] X. B. Peng, G. Berseth, and M. van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 35, no. 4, 2016.

[35] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *CoRR*, vol. abs/1707.02286, 2017.

[36] Z. Xie, H. Y. Ling, N. H. Kim, and M. van de Panne, "Allsteps: Curriculum-driven learning of stepping stone skills," in *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2020.

[37] Y.-S. Luo, J. H. Soeseno, T. P.-C. Chen, and W.-C. Chen, "Carl: Controllable agent with reinforcement learning for quadruped locomotion," *ACM Trans. Graph.*, vol. 39, no. 4, Jul. 2020.

[38] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 143:1–143:14, Jul. 2018.

[39] K. Bergamin, S. Clavet, D. Holden, and J. R. Forbes, "Drecon: Data-driven responsive control of physics-based characters," *ACM Trans. Graph.*, vol. 38, no. 6, Nov. 2019.

[40] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," 2018.

[41] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. van de Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 317–329.

[42] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3699–3706, 2020.

[43] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 07 2019.

[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[45] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable dis-

tributed deep-rl with importance weighted actor-learner architectures," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[46] J. Mirabel, S. Tonneau, P. Fernbach, A.-K. Seppälä, M. Campana, N. Mansard, and F. Lamiraux, "HPP: a new software for constrained motion planning," in *International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea, Oct. 2016.

[47] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE/SICE International Symposium on System Integrations (SII)*, paris, France, Jan. 2019.

[48] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. Del Prete, P. Souères, N. Mansard, F. Lamiraux, J.-P. Laumond, L. Marchionni, H. Tome, and F. Ferro, "TALOS: A new humanoid research platform targeted for industrial applications," in *International Conference on Humanoid Robotics, ICHR, Birmingham*, ser. IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids),. Birmingham, United Kingdom: IEEE, Nov. 2017.

[49] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," 2018.