Learning to steer a locomotion contact planner

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

Learning to steer a locomotion contact planner

Jason Chemin[1], Pierre Fernbach[2], Daeun Song[3], 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 models; then compute the contact sequence to balance the robot gait along the guide path. While several models have been proposed to compute such 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, able 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 inducing a loss in generality. We have designed a dedicated environment and the associated reward function where a classical reinforcement learning algorithm can be run to compute the steering method. The policy takes as inputs a target direction and a local heightmap of the terrain around the robot, to steer 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 demonstrate that the trained policy is able to generate feasible contact plans with a higher success rates than previous approaches and that it generalises to terrains not considered during the training. As a result, the policy can be used with a path planning algorithm to navigate in 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 dynamics and geometric constraints that change discretely with the choice of contacts, thus introducing a combinatorial aspect to the problem.

As a result a common approach to tackle the problem is to decompose it into simpler problems sequentially solved [4], [7]. The most common simplification aims at avoiding the combinatorics induced by the choice of contact locations. Indeed whether they consider the full body representation [8], [9] of the robot, or a simplified one focused on the Center Of Mass [10]–[14], most trajectory generation methods assume that the contact positions are given, or relax the discrete contact selection problem into a continuous one [15], [16].

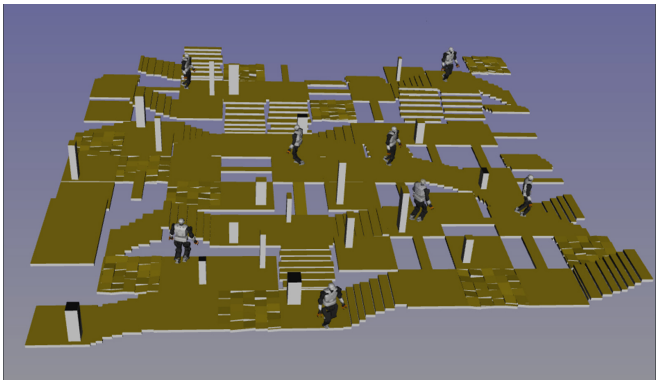

Fig. 1: The locomotion arena, a large stuctured simulation where our Talos agent learns in which direction to steer in order to generate stable contacts.

Though efficient techniques exist for planning gaited locomotion [17]–[20], one challenge of planning contacts in arbitrary contexts results from the fact that the contact manifold has a null measure in the configuration space[1]. Several contact planners that have proven their robustness in challenging cases are also known for their prohibitive computation time [4]. Using a guide path for the root of the robot to reduce the search space significantly improves the computation time (from hours to a few minutes) [21]. Thus at the top of this cascade of problems that lead to the generation of a motion, lies the issue of computing efficiently a guide path.

In [7], [22] low dimensional guide-path planners have been proposed to interactively compute a relevant guide path, further reducing the total computation time to a few seconds. However, the low-dimensional approximation potentially leads to the computation of unfeasible guide paths and result in failure of the complete motion generation pipeline.

To find a compromise between accuracy [21] and efficiency [7], we propose to improve the quality of the low-dimensional approximation using a reinforcement learning approach. [7] introduces the **reachability condition**, which approximates the combination of a necessary condition and a sufficient condition for a guide-path plan to be feasible. We use Reinforcement Learning (**RL**) to tighten the condition and obtain a better approximation of a sufficient constraint

[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

This work has been supported by the MEMMO European Union project within the H2020 Program under Grant Agreement.

[1]Informally, the set of postures in contact and not in the collision with the environment is so small with respect to the set of all possible postures that it can't be sampled.

without losing the generality of the approach. The necessary condition, required to efficiently explore the search space, remains unchanged.

Specifically, we propose to learn the steering method that builds the guide path, that is the local method which, given a current 6D position for the root of the robot and a desired target position, computes a trajectory for the root that defines a feasible contact planning problem. During the training, we use feedback from the contact planner (**CP**) which tries to generate a sequence of foot contact configurations on this trajectory and returns an evaluation of its success. The RL algorithm maximizes this success rate, thus learning what is a good trajectory for this contact planner. Our agent is trained in a large but fixed environment. We demonstrate that our Learned Steering method **LeaS**, is able to generalize in unknown environments where it can locally navigate to follow a target direction, avoid collisions, and make a stop before an unavoidable fall. While our results focus on legged locomotion, the generic formulation of the contact planning problem allows us to consider an extension to multi-contact motions that will be investigated for future work.

II. Learning

A. Machine learning in locomotion

Recent improvements in machine learning [23], [24] have opened new possibilities for learning locomotion on legged mechanisms. Data-driven approaches learn how to move in a kinematic domain [25]–[27], and to interact with its environment [28]. In computer graphics, RL has been used in physics-based simulation to learn locomotion through trial and error for whole body control [29], [30]. Finally, mixed approaches melting supervised and reinforcement learning have also been explored [31]–[33] to combine naturalness and robustness of locomotion. In [19], [20] it has been proposed to learn contact planning. The learned policy predicts the next contact, along with the whole body movement to reach it, using a heightmap to represent the surrounding terrain. In a similar spirit to our paper, [34] learns by reinforcement the local steering method, applied to planning the movement a wheeled robot in an unknown environment. We take inspiration from these methods but focus on what we believe is the key problem in contact planning: defining the guide path along which contacts can be created.

B. Reinforcement Learning

Reinforcement Learning is 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 the transition probability $p(s_{t+1}|s_t, a_t)$, R a reward function mapping $S \times A \rightarrow \Re$ and $\gamma \in [0, 1)$ the 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 which 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) [35].

Algorithm 1: RL with Master-Minions architecture

```
/* τ = (S, A, S', R, done)                                   */
initialization of contact planner on minions;
id = 0;
listT = [];
batch = [];
for iteration = 1, 2, ... do
    for actor = 1, 2, ..., N_actors do
        A_actor = π_θold(S_actor);
        τ = step(S_actor, A_actor);
        Save τ in T_actor;
        if τ.done then
            Save T_actor in listT with id;
            Send T_actor to minions with id;
            id+ = 1 ;
        end
    end
    if result contact planner received then
        for each result do
            id = result.id ;
            lastIndex = result.lastIndex;
            Retrieve T_id from listT;
            /* discard fails                                 */
            T_id = T_id[0 : lastIndex];
            T_id[lastIndex − 1].done = True;
            Add T_id to batch;
        end
    end
    if batch is full then
        Learn from batch and update θ;
        θ_old ← θ;
        Discard all T to minions;
        Reset minions;
    end
end
```

C. Learning from the contact planner

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. At the end of an episode, the sequence of robot 6D positions represents the trajectory. This trajectory is then given to the contact planner which calculates the contact configurations along it. We learn to improve the contact planner by discarding all the unsuccessful part of the path.

However, the calculation time for a contact sequence can vary from one to several seconds depending on the nature of path and terrain. If we have an architecture with several actors, a problem of time inefficiency appears as explained in [36]. Indeed, with a synchronous steps implementation, if an actor terminates its episode, the calculation of contacts on its trajectory will make all the others wait for it. The issue remains with a synchronous trajectory implementation, as each actor will have different calculation times. To solve this problem, we separate contact planning from acting, using a

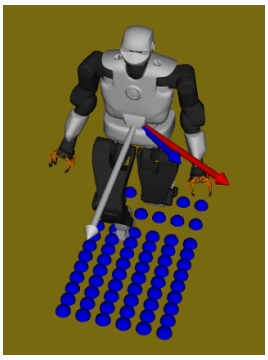

Fig. 2: Observable state with root orientation (**white arrow**), root velocity (**Blue arrow**), the desired target velocity and orientation (**Red arrow**) and the local height map (**Blue dots**).

Master-Minion architecture as shown on Algorithm 1. When an actor is done, it saves its trajectory and sends it on a server. Then a minion receives the trajectory and uses its contact planner. After calculation, it returns the index to locate the failed part of the path. Meanwhile, all the actors keep on generating trajectories and at the end of each step, check if a result is received. If it is the case, we discard the failed part of the trajectory before learning.

III. Environment

A. States

The observable state is a set $S = \{v_{ori}, ori_{target}, h_{ori}\}$ where v_{ori} is the velocity of the robot root relative to its orientation, ori_{target} is the angle difference between its orientation and the direction to target, and h_{ori} is the local heightmap relative to its orientation. The states and dimensions of the heightmap can be seen in Figure 2. The heightmap dimensions are : 9 values in front of the robot, 2 values in its back and 7 values from left to right with a discretization of respectively 8cm, 20cm and 15cm. The heightmap is small on purpose as we want to have a local steering method following a target direction. Increasing its size would lead the policy to perform path planning in some scenarios, by not moving the robot toward the desired direction but in another one that could maximize its reward later on.

B. Actions

Our policy returns a set of action $A = \{a_x, a_y, a_z, \omega\}$ with a_x, a_y, a_z the accelerations of our robot on each axis and ω the angular velocity of its orientation. At each step, the robot configuration is modified as follows : (1) $Robot_{pos} += Robot_{vel} * timestep$, (2) $Robot_{vel} += [a_x, a_y, a_z] * timestep$ and (3) $Robot_{orientation} += \omega * timestep$ where timestep is a constant (Table I). A low timestep value means more configurations on our trajectory to reach a target. On the opposite, if we raise this value, the contact planner will have less configurations to work from. By experience, we know [7] works better with timestep values between $[0.10s, 0.01s]$.

C. Termination conditions

We consider that an episode is over when the maximum number of steps is reached or the actual configuration is invalid. When playing the trained policy, we also consider that an episode can be done when the root position of the robot has reached the target $d(root, target) < epsilon$ with $epsilon$ a constant. The validity of a configuration is done by checking if there is an obstacle in collision with the robot root and if it can touch the ground by looking at the *reachability condition* [7]. The validity of this condition is necessary but not sufficient for the contact planner to find a contact from this root configuration.

D. 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 not able to go further, (4) generate a smooth path and (5) make this trajectory feasible by the contact planner.

The reward for (1) punishes the agent for not going in the desired direction at the right velocity : $R_{dir} = -(||\overrightarrow{v} - \overrightarrow{v*}||/(2 * v_{max}))^2$ 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 continue its trajectory 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 trajectory as explained in II-C . The resulting reward is : $R = R_{dir} * w_{dir} + R_{ori} * w_{ori} + effort_{ori} * w_{eff\_ori} + effort_{acc} * w_{eff\_acc} + R_{alive} * w_{alive}$

The design of a reward for the behaviors (1),(2),(4) and (5) is straightforward and we could obtain it using positive rewards 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 impossible to overcome, he needs to have a reason for not terminating the episode on purpose and to avoid accumulating negative rewards. We opt for a simple way to solve this problem by adding a constant positive value R_{alive} at each step. By selecting proper weights for all the rewards, this method learns the policy with the behavior (3) along with all the others.

IV. Results

We used the planner HPP [37] and the Talos model [38]. Our algorithm was implemented in python using the PPO implementation of Stable Baselines, modified for our Master-Minion architecture as in Algorithm 1. The simulation was

TABLE I: Parameters

States dimension	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		

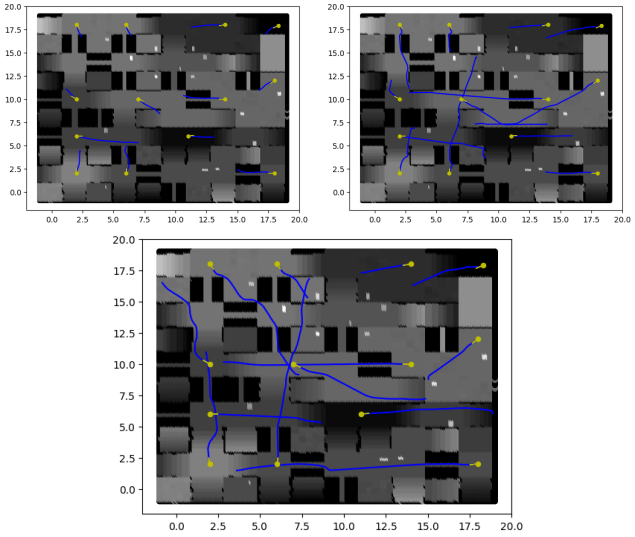

Fig. 3: Trajectories after 150K, 1M5 and 80M steps of learning on 10x10 grid training terrain with some initial positions and target directions (**Yellow**) and their trajectories (**Blue**).

run on a PC with Intel Core i7-8700 (12 cores, 3.20Ghz). Our network is fully-connected with two hidden layers of 64 units as done in [35]. We trained our policy for 53 hours using parameters in Table I, with an architecture of 6 actors in parallel and 7 external minions validating the trajectories with the contact planner [7]. The training terrain is a 10x10 grid ($20 \times 20m$) where each grid corresponds to a type of terrain as depicted by Fig. 1. We then used another terrain, that the agent never met during the training, to perform our comparisons, and where the dimension of stairs, obstacles and beam are slightly different from any of those encountered during the training.

A. Learning progression

We evaluate the progress of the learning by plotting trajectories obtained with a policy at different stages of the training on Figure 3. We want our policy to follow a direction at the right velocity as far as possible, while ensuring the success of the contact planner on this root trajectory. The policy in the early phase of the training makes our robot go toward the target, but does not take into account the surrounding terrain and fails to produce trajectories with 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 is not able to go further, as demonstrated in Figure 4 in an unknown environment by our policy.

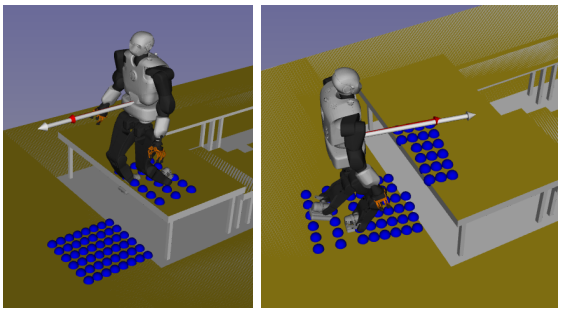

Fig. 4: The robot can not go further and the obstacle is detected on its local height map (**Blue dots**) in the desired direction (**Red arrow**) so the policy makes the robot stop.

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 of steering methods with the CP on flat ground trajectories of 4 meters with different initial orientations and velocities (60 trajectories for each test)

B. Comparison of steering methods

In this section, we compare LeaS to RBLin [7] and RBKino [22], to perform a local task: given an initial configuration, the goal is to reach a target by producing a root trajectory, 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.

*1) **Ground**:* Given a initial and target position, we set different initial orientations and velocities to evaluate the success of all the steering methods with the contact planner in Table II. The success value corresponds to the percentage of the path covered by the contact sequence. The orientation value corresponds to the angle between the initial target direction and the root orientation of the robot. Results shows that for all orientation going from 0 to 120 degrees, all three methods succeed in reaching the target. But when the robot is facing backward, the success of both previous methods drops. 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 performs better but fails once in a four to generate a valid trajectory. We also notice that with both methods, raising the initial velocity improve its success

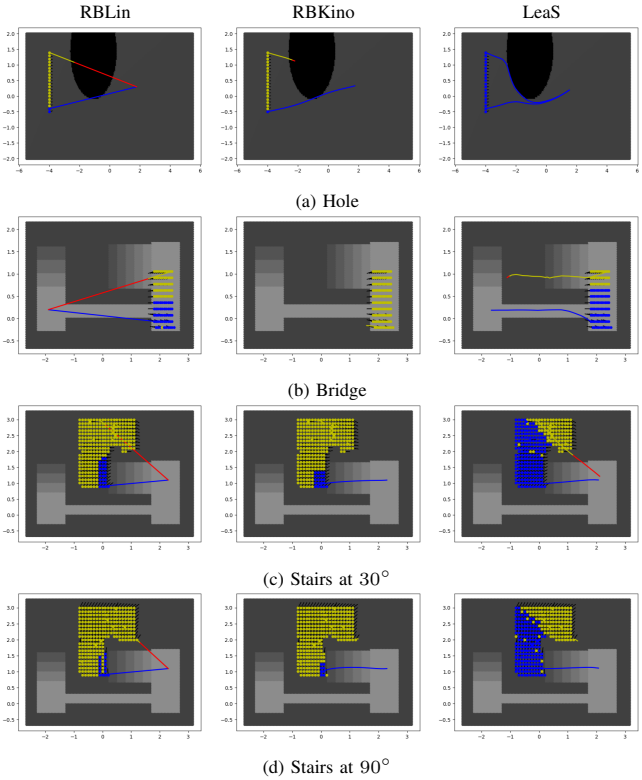

Fig. 5: Trajectories of steering methods with : valid trajectory to target (**Blue**), failed trajectory to target (**Yellow**), failed part of a trajectory (**Red**) and initial orientation (**Black Arrow**).

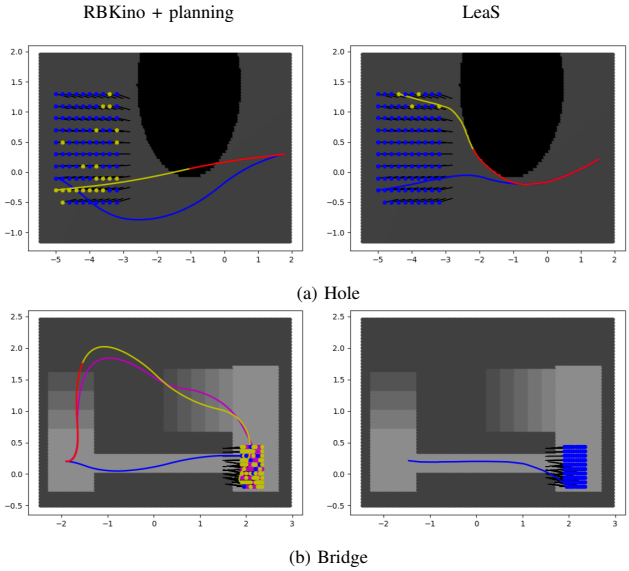

Fig. 6: Trajectories of RBKino with path planning and LeaS with : valid trajectory to target (**Blue**), failed trajectory to target (**Yellow**), failed part of trajectory (**Red**) and path valid but not optimal by not taking the bridge (**Magenta**).

rate. On the other hand, our method LeaS performs well in generating valid trajectories whatever the initial velocity and orientation.

*2) **Hole**:* In the hole scenario (Fig. 5a), some initial configurations are sampled in line and the target is fixed on the other side of the hole. The difficulty of this scenario is to have trajectories 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 trajectories lead inside the hole. The same goes for RBKino which in most cases can not generate a complete trajectory leading to the target. In order to succeed this scenario, both of these steering methods require path planning and intermediary waypoints to the target, whereas LeaS succeeds in generating a valid trajectory leading to the target while keeping a safe distance from the hole.

*3) **Bridge**:* In this scenario (Fig. 5b), the goal is to make the robot cross a bridge of 30cm width to reach a fixed target on the other side. As the hole scenario, one of the difficulties is to make the robot stay away from the edges. RBKino here 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 trajectories very close to the edges. On the other hand, LeaS succeeds to cross 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 local reduced heightmap, the yellow dots show cases where LeaS detects the stairs and not the bridge in the target direction, and decides to go down. After going down the stairs, it detects a platform too high to reach and makes the robot stop moving.

*4) **Stairs**:* In the stairs scenario (Fig. 5c and 5d), 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 trajectories climbing the stairs are too close or far from the ground. As a result, it can not find a contact transition in equilibrium and fails the trajectory. The result of RBLin shows the difficulty to climb the stairs while keeping a right height from the ground, thus leading to a reduced zone of initial configuration with valid trajectories. 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, the number of successful trajectories for all three methods is reduced. With our method, the policy detects the stairs later in the trajectory as the robot rotates while getting closer to the target, and thus can explain the success difference between Figures 5c and 5d.

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 [22]. Results in IV-B show that RBKino alone is too local to be fairly compared to other methods. That is why we use RBKino with its path planner, which explores waypoints

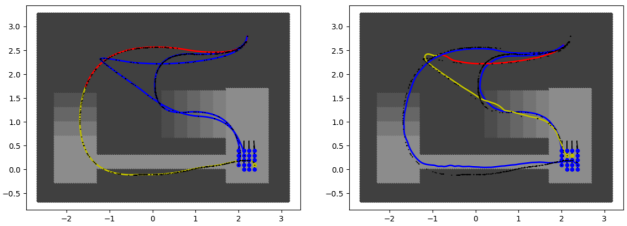

Fig. 7: Trajectories following waypoints made by the path planning of RBKino (**left**) and LeaS following it (**right**) with : valid trajectory to target (**Blue**), fail trajectory to target (**yellow**), failed part of trajectory (**Red**) and the waypoints to follow (**Black**).

stochastically to simplify the task of the local controller and maximizes its convergence. We perform a comparison with LeaS on hole and bridge scenarios, where RBKino alone was failing. No path planning strategy is used for our policy, which remains a pure local method. Results for both scenarios are shown in Figure 6.

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 as a tightrope walker on the edge of the hole. LeaS still performs better on average with fewer failure cases 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 shows for RBKino 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 a 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 for the robot to follow and connect it using its local SM. In this test, we replace this steering method with ours. The results in Fig. 7 show that our local steering method can be used to follow these waypoints and compares well with the RBKino.

2) Manually defined circuit: Figure 8 shows some manually placed waypoints that our robot has to follow. We start the trajectory on the red dot with an orientation indicated by the black arrow. Our agent succeeds in rotating the robot toward the next waypoint and in reaching all of them one by one.

V. Conclusion

We have proposed a new approach for training a steering method which generates a root trajectory for humanoids robots. Our steering method is trained to fit a contact planner and to generate trajectories feasible by it. Our method is able to generalize to unknown environments and performs better in all scenarios than our previous techniques with the contact planner. It could be used either as a replacement of the heuristic steering methods of the current planner, or as controllers to steer in real-time the robot locomotion e.g. in a navigation task.

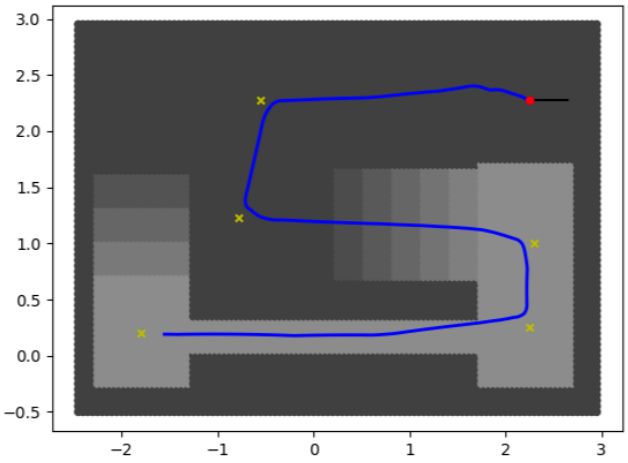

Fig. 8: LeaS following manually placed waypoints with : the successful trajectory (**Blue**), the waypoints to reach (**Yellow**) and its initial orientation (**Black arrow**).

Several extensions naturally arise. We yet only train the policy to satisfy the (implicit) working hypotheses of the contact generator, yet it should also enable the more restrictive constraints needed to generate a complete motion of the whole body. We also would like to take the policy input directly from sensor measurements (e.g. lidar data), and to validate the approach on the 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," *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 2, pp. 1620–1626 vol.2, 2003.

[2] K. Hauser, T. Bretl, and J.-C. Latombe, "Non-gaited humanoid locomotion planning," in *Humanoid Robots, 2005 5th IEEE-RAS Int. Conf. on*, 2005, pp. 7–12.

[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 2015)*, Sestri Levante, Italy, Sep. 2015. [Online]. Available: https://hal-lirmm.ccsd.cnrs.fr/lirmm-01149666

[8] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *2014 IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 295–302.

[9] M. Carlos, B. Rohan, M. Wolfgang, S. Guilhem, H. Bilal, N. Maximilien, R. L. Carpentier Justin, V. Sethu, and M. Nicolas, "Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[10] 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.

[11] J. Englsberger, C. Ott, and A. Albu-Schäffer, "Three-dimensional bipedal walking control based on divergent component of motion," *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 355–368, 2015.

[12] 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*, Jul. 2015.

[13] J. Carpentier, S. Tonneau, M. Naveau, O. Stasse, and N. Mansard, "A versatile and efficient pattern generator for generalized legged locomotion," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3555–3561.

[14] 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. [Online]. Available: https://doi.org/10.1145/3213773

[15] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, no. 4, Jul. 2012. [Online]. Available: https://doi.org/10.1145/2185520.2185539

[16] A. W. Winkler, F. Farshidian, M. Neunert, D. Pardo, and J. Buchli, "Online walking motion and foothold optimization for quadruped locomotion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5308–5313.

[17] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 279–286.

[18] 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," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6604–6610, 2020.

[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 2017)*, vol. 36, no. 4, 2017.

[20] 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.

[21] K. Bouyarmane, A. Escande, F. Lamiraux, and A. Kheddar, "Potential field guide for humanoid multicontacts acyclic motion planning," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 1165–1170.

[22] 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. [Online]. Available: https://hal.laas.fr/hal-01486933

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[24] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[25] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017. [Online]. Available: https://doi.org/10.1145/3072959.3073663

[26] 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. [Online]. Available: https://doi.org/10.1145/3197517.3201366

[27] F. G. Harvey and C. Pal, "Recurrent transition networks for character locomotion," in *SIGGRAPH Asia 2018 Technical Briefs*, ser. SA '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3283254.3283277

[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. [Online]. Available: https://doi.org/10.1145/3355089.3356505

[29] 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. [Online]. Available: http://arxiv.org/abs/1707.02286

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

[31] 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. [Online]. Available: https://doi.org/10.1145/3386569.3392433

[32] 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. [Online]. Available: http://doi.acm.org/10.1145/3197517.3201311

[33] 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. [Online]. Available: https://doi.org/10.1145/3355089.3356536

[34] 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.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[36] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[37] 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 2016)*, Daejeon, South Korea, Oct. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01290850

[38] 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 2017*, ser. IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids),. Birmingham, United Kingdom: IEEE, Nov. 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01485519