Rapid and Reliable Trajectory Planning Involving Omnidirectional Jumping of Quadruped Robots

Matthew Chignoli$^1$, Savva Morozov$^2$, and Sangbae Kim$^1$

Abstract—Dynamic jumping with multi-legged robots poses a challenging problem in planning and control. Formulating the jump optimization to allow fast online execution is difficult; efficiently using this capability to generate long-horizon trajectories further complicates the problem. In this work, we present a novel hierarchical planning framework to address this problem. We first formulate a real-time tractable trajectory optimization for performing omnidirectional jumping. We then embed the results of this optimization into a low-dimensional jump feasibility classifier. This classifier is leveraged by a high-level planner to produce paths that are both dynamically feasible and also robust to variability in hardware trajectory realizations. We deploy our framework on the Mini Cheetah Vision quadruped, demonstrating robot’s ability to generate and execute reliable, goal-oriented paths that involve forward, lateral, and rotational jumps onto surfaces 1.35 times taller than robot’s nominal hip height. The ability to plan through omnidirectional jumping greatly expands robot’s mobility relative to planners that restrict jumping to the sagittal or frontal planes.

I. INTRODUCTION

Legged animals are capable of many agile movements beyond basic walking and running. Jumping, especially omnidirectional jumping, allows robots to be deployed in environments with severely discontinuous elevation. The ability to reliably perform these jumps, like the one shown in Fig. 1, is a challenge in its own right. Planning when and which jump to perform adds another layer of complexity. This planning should happen quickly, at a rate that matches the dynamism of the robot, and should be aware of the robot’s physical capabilities to avoid planning infeasible jumps. We formulate a trajectory optimization that efficiently generates omnidirectional jumps, and then incorporate the results of this optimization into a hierarchical navigation framework. This allows us to rapidly plan paths through challenging environments while taking into account the disturbances and uncertainty related to executing a planned trajectory on a physical robot.

A. Related Work

In model-based control frameworks, the complexity of the considered model typically scales with the complexity of the task at hand. In the case of omnidirectional jumping, especially jumps with large rotational components, the model needs to be suitably detailed to capture these dynamics. Trajectory optimization has proven to be a powerful tool for reasoning about such complex models [1], [2] and generating impressive aerial motions [3]–[5]. Yet performing these optimizations in real-time onboard a mobile robot is still challenging, even for motions that consider short time horizons [6], [7].

For tasks that require planning over longer time horizons, such as navigating through terrain with obstacles, complicated models are computationally intractable and at times unnecessary. Consequently, hierarchical frameworks are often implemented to decouple short and long-horizon planning tasks [8]–[11]. Optimization-based methods like those described above tend to dominate low levels of the hierarchies, while high-level planning can be accomplished using optimization-based [12] or sampling-based methods [13]–[16] with lower-dimensional models.

Utilizing reduced-order models in high-level planning poses a challenging trade-off. Relaxing the complex robot dynamics may lead to high-level trajectories that are infeasible for a low-level controller to execute. On the other hand, further constraining the underlying dynamics may lead to overly-conservative high-level trajectories that fail to find a solution when one exists. To address this trade-off, techniques for embedding whole-body dynamics into...
the high-level reduced-order model have been proposed, but are limited to simulation [17] or slow-moving, quasi-static motions [18].

Additionally, uncertainties from external disturbances, imprecise tracking and estimation, and other unmodeled dynamics are inevitable when deploying control frameworks on hardware. Unless accounted for, this uncertainty may result in a failure mode, such as the robot colliding with obstacles. A well-designed high-level planner must therefore take into account the inevitable noise and variability that the robot will experience over the course of accomplishing its task.

To this end, we first formulate a trajectory optimization that is expressive enough to generate online jumping motions and efficient enough to plan omnidirectional jumps in real-time on the robot. We then incorporate this optimization in a novel, hierarchical framework for multi-modal locomotion. The framework uses a robustness metric to allow the high-level planner to rapidly generate goal-oriented trajectories that select the “safest” possible jumps - i.e., those that are farthest from the boundary between feasible and infeasible actions of the robot [19].

B. Contributions

The contributions of this paper are as follows:

1) A trajectory optimization that efficiently generates collision-free, omnidirectional jumps onto different heights.
2) A condensed representation of the robot’s jumping capabilities, suitable for use in fast high-level planning.
3) A real-time, high-level planner that uses jump feasibility to generate trajectories that are (A) able to cover terrain accessible only through omnidirectional jumping and (B) reliable against the effects of variability in execution.
4) Demonstration of the framework on the Mini Cheetah Vision [20]. The robot successfully plans and follows goal-oriented paths that involve forward, lateral, and rotational jumps onto surfaces as high as 35cm - 1.35 times taller than the robot’s nominal hip height.

The remainder of this paper adheres to the following structure. Section II provides an overview of the framework. Section III details the formulation of the kino-dynamic optimization that generates the jumping motions. Section IV discusses the jump feasibility classifier used by the high-level planner and how variability in jumping parameters, driven by the uncertainty in the hardware execution, informs jump selection. Section V details the specifics of a path planning algorithm used to generate reliable high-level trajectories involving both walking and jumping. Section VI discusses the results of deploying this framework on the MIT Mini Cheetah Vision, and Section VII concludes the work and looks towards future research.

II. FRAMEWORK

In this section, we provide a brief overview of our hierarchical planning framework that allows dynamic exploration over challenging environments. First, the robot uses its perception stack (brown block in Fig.2) to estimate its pose and build a robot-centric elevation map of the environment around it. The robot's operator then provides a desired position to the robot (yellow block in Fig.2). Taking into account the terrain elevation and the physical limits of the robot, the high-level sampling-based planner (blue block) generates a collision-free trajectory that includes walking and jumping modes. The generated trajectory produces an associated mode schedule, which is used by the high-level trajectory tracking controller (purple block) to inform the lower-level locomotion module (green block) when to walk or jump. Walking is accomplished with a hybrid convex model predictive control/whole-body impulse control scheme, discussed in [21]. The jumping controller, discussed in detail in Section III, is based on kino-dynamic trajectory optimization. Our framework allows the robot to quickly plan and execute dynamic omnidirectional trajectories that cannot be generated under common simplifications like planar jumping or quasi-static walking.

III. KINO-DYNAMIC JUMP OPTIMIZATION

Kino-dynamic optimization simultaneously optimizes over the centroidal dynamics and the joint-level kinematics of the robot. Such an optimization is well-suited for online 3D motions because the centroidal dynamics of the robot [22] are both computationally simple to work with as well as expressive enough to produce diverse motions. The kinematic aspect of the formulation is important because it ensures collision avoidance with the environment.

The centroidal dynamics of the robot can be described by state and control trajectories,

$$\mathbf{X}^{\text{dyn}}(t) = \begin{bmatrix} \mathbf{r}(t) \\ \dot{\mathbf{r}}(t) \\ \mathbf{h}(t) \\ \mathbf{c}_1(t) \\ \vdots \\ \mathbf{c}_{n_c}(t) \end{bmatrix}, \quad \mathbf{U}^{\text{dyn}}(t) = \begin{bmatrix} f_1(t) \\ \vdots \\ f_{n_c}(t) \end{bmatrix} \quad (1)$$

Fig. 2. Overview of the proposed navigation framework.
where \( r \in \mathbb{R}^3 \) is the center of mass trajectory, \( \dot{r} \in \mathbb{R}^3 \) is the center of mass velocity trajectory, \( h \in \mathbb{R}^3 \) is the centroid angular momentum, and \( c \in \mathbb{R}^{3n_c} \) and \( f \in \mathbb{R}^{3n_c} \) are the contact locations and ground reaction forces at the \( n_c \) contact points, respectively. The kinematic trajectory of the robot is given by

\[
X_{\text{kin}} = \begin{bmatrix} \mathbf{q}(t) \\ \dot{\mathbf{q}}(t) \end{bmatrix} \tag{2}
\]

where, for an \( n \) degree of freedom robot (including six floating-base degrees of freedom) \( \mathbf{q} \in \mathbb{R}^n \) are the joint positions and \( \dot{\mathbf{q}} \in \mathbb{R}^n \) are the joint velocities.

To formulate the problem as a trajectory optimization, the state and control trajectories are discretized into a set of \( N_{\text{TO}} \) timesteps corresponding to the takeoff period (all legs in contact) and a set of \( N_{\text{FL}} \) timesteps corresponding to the flight period (all legs out of contact). The total number of timesteps for the optimization is, therefore, \( N = N_{\text{TO}} + N_{\text{FL}} \). The duration of the takeoff period, \( T_{\text{TO}} \), is chosen beforehand, while the duration of the flight phase, \( T_{\text{FL}} \), is left as an optimization variable.

After discretization, the optimization variables (1) and (2) become

\[
X_{\text{dyn}} = \begin{bmatrix} X_{\text{dyn}}^1, & \ldots, & X_{\text{dyn}}^N \end{bmatrix},
\]

\[
U_{\text{dyn}} = \begin{bmatrix} U_{\text{dyn}}^1, & \ldots, & U_{\text{dyn}}^{N_{\text{TO}}-1} \end{bmatrix},
\]

\[
X_{\text{kin}} = \begin{bmatrix} X_{\text{kin}}^1, & \ldots, & X_{\text{kin}}^N \end{bmatrix}.
\]

The goal of the trajectory optimization problem in this work is to find a control policy \( U_{\text{dyn}} \) that results in the robot safely jumping onto an elevated surface. We formulate this optimization according to

\[
\min_{X_{\text{ref}}, U_{\text{dyn}}, X_{\text{kin}}} \left[ \| \dot{X}_{\text{dyn}} \|^2_{Q_d} + \| \dot{X}_{\text{kin}} \|^2_{Q_k} + \| U_{\text{dyn}} \|^2_{Q_u} \right] \quad \text{s.t.} \quad \begin{align*}
X_{i+1} &= X_i + f_{\text{dyn}}(X_i, U_i) \Delta t_i, \\
q_{i+1} &= q_i + \gamma \Delta t_i, \\
X_i &= \text{kinematics}(X_{\text{dyn}}), \\
U_i &= U_{\text{dyn}}, \quad i = 1, \ldots, N_{\text{TO}} - 1, \quad c_i \in C_i, \\
X_{\text{kin}}^i &= X_{\text{kin}}(0), \quad X_N^{\text{kin}} \in \mathcal{X}_{\text{land}}
\end{align*} \tag{4a}
\]

The cost function \( \mathcal{J} \) penalizes deviations from heuristically derived reference trajectories \( X_{\text{ref}} \) and \( X_{\text{kin}}^{\text{ref}} \), where

\[
\dot{X}_{\text{dyn}} = X_{\text{dyn}} - X_{\text{kin}}^{\text{ref}}, \quad \dot{X}_{\text{kin}} = X_{\text{kin}} - X_{\text{kin}}^{\text{ref}},
\]

and \( Q_d, Q_k, \) and \( Q_u \) are weighting matrices.

The centroidal dynamics of the system are encoded in (4b) via

\[
f_{\text{dyn}} = \frac{1}{m} \sum_{k=1}^{n_c} \sum_{i=1}^{n_e} \mathbf{f}_{k,i} (\mathbf{c}_{k,i} - \mathbf{r}_i) \times \mathbf{f}_{k,i},
\]

where \( m \) is the mass of the robot and \( \mathbf{f}_{k,i} \in \mathbb{R}^3 \) is the reaction force at the \( k \)th contact point at the \( i \)th timestep. Note that for (4b) and (4c), the timestep \( \Delta t_i \) is different for the takeoff and flight phases,

\[
\Delta t_i = \begin{cases} 
\frac{T_{\text{TO}}}{N_{\text{TO}} - 1}, & i < N_{\text{TO}} \\
\frac{T_{\text{FL}}}{N_{\text{FL}} - 1}, & N_{\text{TO}} \leq i < N
\end{cases},
\]

since \( T_{\text{TO}} \) is fixed, but \( T_{\text{FL}} \) is an optimization variable. This added nonlinearity to (4b) and (4c) is necessary because we cannot know a priori how long flight should last. Underestimation will not give the robot enough time to reach the required height, while overestimation will cause the robot to expend unnecessary energy or, worse yet, call for an overly aggressive jump that exceeds the robot’s torque limits.

Agreement between the centroidal dynamic and kinematic states of the robot is enforced via (4d). This constraint enforces that at each time step, applying forward kinematics to \( X_{\text{kin}} \) will yield a CoM and contact point locations that match \( X_{\text{dyn}} \). Furthermore, it enforces that the total angular momentum of the individual links of the robot is equal to the centroidal angular momentum [22].

Contact constraints are enforced via (4e). The set of feasible ground reaction forces \( U \) is given by a friction pyramid inscribed in the cone

\[
\left\{ (f_x, f_y, f_z) \in \mathbb{R}^3 | f_x^2 + f_y^2 \leq \mu f_z \right\}, \tag{8}
\]

where \( \mu \) is the coefficient of friction with the ground. The set of allowable footstep locations \( C \) depends on the phase. In takeoff, i.e., for \( i \leq N_{\text{TO}} \), \( C_i \) enforces that the feet remain in the position where they started. In flight, \( C_i \) is the set off all footstep positions above the terrain

\[
C_i = \left\{ (c_x, c_y, c_z) \in \mathbb{R}^3 | c_x > z_g(c_x, c_y) \right\}, \quad N_{\text{TO}} \leq i < N \tag{9}
\]

where \( z_g \) is a continuous function that maps the \( xy \) position of a foot to the height of the terrain at that point. In our case, we approximate discrete changes in elevation using the sigmoid function

\[
z_g(x, y) = \frac{a}{1 + e^{-\gamma(x-b-d_y)}}, \tag{10}
\]

where \( a, b, \) and \( d \) are parameters fit from height map data and \( \gamma \) is a manually selected parameter that dictates the steepness of the sigmoid.

Lastly, boundary conditions for the state of the robot are given by (4f). \( X_{\text{kin}}(0) \) is the initial state and \( \mathcal{X}_{\text{land}} \) is the allowable landing state on the elevated surface. The values associated with \( X_{\text{kin}}(0) \) come from the robot’s state estimator, while the values associated with \( \mathcal{X}_{\text{land}} \) come from the high-level path planner.

IV. FEASIBILITY CLASSIFIER: SELECTING RELIABLE JUMPS

We inform the high-level trajectory planner of the jumping controller’s capabilities via a feasibility classifier we hereafter refer to as a “jump feasibility classifier.” The JFC takes as input a description of the intended jump and outputs a binary classification of the jump as feasible or infeasible. The JFC allows the high-level path planner to be executed efficiently, since the feasibility of a jump can be checked as a cheap classifier evaluation instead of solving a full trajectory optimization. The JFC also offers a way to measure a given jump’s robustness to variability in jumping parameters due
to the uncertainty that arises when executing a planned path. This is important since, in practice, external disturbances and unmodeled dynamics will always prevent the robot from perfectly tracking its higher-level trajectory.

A. Jump Feasibility Classifier Generation

Our JFC is extracted from offline simulations of thousands of attempted jumps sampled from a 5D space of jumping parameters. To reduce the search space, we limit the planner to jumps where the robot’s center of mass velocity is perpendicular to the surface it is jumping onto. As a result, any jump can be described by a point in this 5D jump parameter space. The 5D space of jumping parameters, along which our JFC is defined, is illustrated in Fig 3. They include the height of the jump surface \( h \), the initial distance from the CoM to the jump surface edge \( d_i \), the initial yaw of the robot relative to the jump surface \( \psi_i \), the final distance of the CoM past the surface edge \( d_f \), and the final yaw of the robot \( \psi_f \).

Data collection of the offline jumps is performed by sampling over the jump parameter space, performing the optimization (4), executing the jump, and then classifying the jump as a success or failure. A successful jump is one where the robot ends with its CoM at the desired height and with an upright posture (i.e., the robot has not fallen over). A failure is any other outcome. We use the collected data, which maps jump parameters to a binary outcome, to train a support vector machine (SVM). This SVM is a compact, memory-efficient way to encode the robot’s observed jumping capabilities into a classifier that can be evaluated efficiently onboard the robot as it plans high-level paths.

B. Identifying the Maximally Robust Jumps

Given a continuous 5-dimensional space of jump parameters \( \mathcal{J}_f \), we aim to find the set of parameters that is maximally robust to the variable that will occur in the course of executing a planned trajectory. To pick such a jump parameterization, we first constrain \( \mathcal{J}_f \) according to the physical limits of our system. This produces a set of feasible jump parameters \( \mathcal{J}_f \). Next, we impose the collision avoidance constraints of the terrain, which produces the terrain-constrained set of feasible jumps \( \mathcal{J}_f \). Given the set \( \mathcal{J}_c \), we can determine the point that is maximally robust to the jump parameter variability that occurs throughout the trajectory realization on hardware. This point, also known as the pole of inaccessibility (PIA), is the point in the set \( \mathcal{J}_c \) that is furthest from the set boundary. We find this point using a PIA search algorithm, similar to [23].

V. HIGH-LEVEL PLANNING

We perform high-level planning using a multi-stage, sampling-based approach that quickly generates collision-free paths to a goal state. Our algorithm is multi-modal in that it plans motions that can alternate between segments of walking and jumping. The planner is practical for hardware implementation not only because of the speed at which it runs, but also because it selects a trajectory that is maximally robust according to the metric defined in Section IV.B.

A. Onboard perception and mapping

Onboard perception, mapping, and state estimation are handled via a suite of Intel RealSense cameras. Terrain data is obtained in the form of a point cloud using a D435 camera and is inserted into a height map [24]. This height map is robot-centric and is propagated using a 200Hz pose estimate produced by the T265 camera. The terrain is subsequently filtered and segmented into flat surfaces [25]. The reader is referred to [26] for a more detailed description of our setup.

B. Sampling-Based Coarse Trajectory Generation

Given the terrain, the current robot’s pose, and the desired location of the robot, we use RRT-connect [27] to find a geometric, collision-free pose trajectory to the goal. This coarse trajectory does not consider robot’s dynamics and uses a set of relaxed traversability heuristics to quickly obtain a rough trajectory estimate. Two heuristics are used: trajectory segments whose surroundings belong to a single flat surface are considered walkable, while the segments between different flat surfaces are accepted as jumps if they satisfy a set of constraints on height difference and initial and terminal locations.

C. Trajectory improvement

The goal of this next stage of the high-level planner is to refine the coarse geometric trajectory into a reliable, dynamically feasible trajectory. Segments of the trajectory that correspond to walking are modified by moving the trajectory vertices away from the obstacles. Subsequently, we find a collision-free region of space around the trajectory, which we call the safety tube. This tube is used by the high-level tracking controller, similar to [9], which attempts to keep the robot within the tube. We presume that the tracking and locomotion controllers can produce a collision-free realization of the trajectory, but this is not guaranteed. The effect of this assumption is mitigated by the fact that the degrees of freedom of the robot’s CoM are independently controllable in the transverse plane.

The coarse, high-level trajectory does not specify a particular jump for the robot to execute - rather, it constrains
the space of feasible jump parameters. Specifically, the terrain surrounding the intended jump location constrains the obstacle height \( h_{\text{obs}} \) to a particular value, while initial \( d_i \) and \( \psi_i \) and terminal \( d_t \) and \( \psi_t \) are constrained to ensure collision-free takeoff and landing. The most robust jump is then selected according to the process described in Section IV.B. If the morphing of the coarse trajectory to a dynamically feasible one is impossible, i.e., if the set \( J_{\text{cf}} \) is empty, then a new coarse trajectory is generated, and the process is repeated until convergence.

VI. RESULTS

In this section, we present the hardware implementation of our hierarchical planning framework outlined in this paper. We first provide solve-time statistics for the underlying planners to demonstrate the framework’s capacity to run online on a robot. We then discuss our ability to perform omnidirectional jumps, emphasizing the advantages for high-level planning. Lastly, we visualize the process of selecting maximally robust jump parameters and further motivate this design choice. A full demonstration of our proposed framework executing planned paths that involve omnidirectional jumps, including jumps with large aerial rotations, is included in the attached video.

A. Hardware Setup

The entire framework can run in real-time onboard a robot with only modest computational capabilities. In our case, we deploy our framework on a robot equipped with two onboard computers: an NVIDIA Jetson TX2 and an Intel UP Board. The more powerful TX2 is used for the more computationally intensive processes (perception, mapping, high-level trajectory planning, and nonlinear trajectory optimization). The UP Board handles simple, lower-level tasks such as locomotion control and communication with the robot’s motor controllers.

B. Fast Online Planning

A major contribution of the proposed framework is its fast onboard computation time. To reduce the solve time of our trajectory optimization, we use the technique of warm starting our optimization using a neural network trained from offline data [28], [29]. After 220 random jump optimizations sampled from the JFC were executed on the TX2, we observed that the optimization takes on average 1.91 s to solve (standard deviation 0.43 s) using KNITRO [30]. While this solve rate is too slow to be used for receding-horizon control, it is faster compared to other schemes that target this versatile behavior of omnidirectional jumps. These optimizations are typically so computationally expensive that they must be either computed offline [2] or on computers too bulky to be placed onboard the robot [6].

After 50 experiments over various terrain configurations, we observe that our high-level trajectory planner, which typically generates paths between 300-500 cm long, takes on average 0.55 s to find a plan (standard deviation 0.32 s). Since the robot is moving at approximately 0.25 m/s for the entire trajectory, the planner can, on average, generate trajectories at a rate 30 times faster than it takes to execute them.

C. Planning with Omnidirectional Jumping

The ability to perform omnidirectional jumping greatly expands the set of terrains the robot can traverse compared to a system that makes a conventional assumption such as planar jumping. Consider the lateral jumping motion animated in Fig. 1. If the robot tried to turn and face the jump surface head-on - a commonly made heuristic - it would collide with either the jump surface or the wall on the right side of the image.

This point is further emphasized by the rotational jump animated in Fig. 4. The high-level trajectory through along with the terrain height map is illustrated in Fig. 5. The constraints of this terrain are such that the robot must rotate anywhere from 60° to 100° while it is in the air to produce a safe, collision-free landing.

Given the terrain in Fig. 5, the initial yaw of the robot is bounded between -30° and 20° while the yaw upon landing is bounded between -100° and -70°. Due to the challenge posed by a difficult aerial rotation, it would be wrong to assume that the robot can execute any jump regardless of its approach. This classification of feasibility as a function of jump conditions is efficiently captured by our JFC. Other
common heuristics, such as purely sagittal or frontal plane jumping, therefore, would fail here.

D. Robust Jumping: Planning Under Uncertainty

The process of finding the maximally reliable initial conditions, assuming fixed terminal states and height, is depicted in Figure 6. The orange polygon, obtained from the offline-generated JFC, captures the set of feasible initial jumping conditions for a given desired terminal state. The red polygon, obtained online in one of our hardware experiments of the robot, represents the terrain-constrained set of jump parameters. The terrain-constrained in Fig 6 for example, corresponds to a jump surface that must be approached at an angle between $-80^\circ$ and $40^\circ$. The intersection of the feasible parameter set and the terrain-constrained parameter set is depicted in blue. The PIA of this polygon, shown in green, represents the point that is maximally robust to variability in the initial conditions of the jump. By performing this process in the 5-dimensional space of jump parameters, we are able to obtain a jump that can handle the largest deviation between the desired trajectory and its realization.

The advantage of our framework is illustrated in Fig 6. Simply picking a jump in the terrain-constrained set (red) is insufficient since, for some terminal conditions $(d_f, \psi_f)$ (bottom plot), most of the jumps in the terrain-constrained set are infeasible. Furthermore, picking an arbitrary point from the constrained-feasible set may also be insufficient, as perturbations in the jumping parameters may result in a collision or render the jump optimization infeasible. By finding the PIA of the entire 5-dimensional constrained feasible set, we ensure that the selected jump is the most robust to variability in jump parameters caused by uncertainty, disturbances, or imprecise tracking and estimation.

VII. CONCLUSIONS & FUTURE WORK

This work first presented a novel trajectory optimization formulation for generating online, omnidirectional jumps. The jumping capabilities endowed by this trajectory optimization are then leveraged by a high-level path planner. The planner considers the physical limits of the robot as well as external constraints of the terrain to select maximally reliable jumps that navigate the robot to a goal position. We synthesize these planners into a hierarchical framework that allows the Mini Cheetah Vision to rapidly plan and execute reliable trajectories through challenging terrain.

Future work will focus on leveraging the jump feasibility classifier to produce provable robustness guarantees. Our current high-level planner selects a jump that is maximally robust to jump parameter variability, given a candidate trajectory. This practical approach ensures that if a set of safe jumps does exist, our algorithm will have found one such jump. However, presently we do not make any claims about the existence of such a safe set - providing certificates of robust trajectory execution is beyond the scope of this paper.

ACKNOWLEDGMENTS

This work was supported by NAVER Labs, Toyota Research Institute, the Centers for ME Research and Education at MIT and SUSTech, and the National Science Foundation Graduate Research Fellowship Program under Grant No. 400092301.

REFERENCES

[1] M. Posa and R. Tedrake, “Direct trajectory optimization of rigid body dynamical systems through contact,” in Algorithmic foundations of robotics X. Springer, 2013, pp. 527–542.

[2] 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. IEEE, 2014, pp. 295–302.
[3] V. Zordan, D. Brown, A. Macchietto, and K. Yin, “Control of rotational dynamics for ground and aerial behavior,” *IEEE transactions on visualization and computer graphics*, vol. 20, no. 10, pp. 1356–1366, 2014.

[4] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, “An efficient optimal planning and control framework for quadrupedal locomotion,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 93–100.

[5] 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.

[6] B. Ponton, M. Khadiv, A. Meduri, and L. Righetti, “Efficient multi-contact pattern generation with sequential convex approximations of the centroidal dynamics,” *IEEE Transactions on Robotics*, 2021.

[7] Y. Ding, C. Li, and H.-W. Park, “Kinodynamic motion planning for multi-legged robot jumping via mixed-integer convex program,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 3998–4005.

[8] H. Li and P. M. Wensing, “Hybrid systems differential dynamic programming for whole-body motion planning of legged robots,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5448–5455, 2020.

[9] D. Wooden, M. Malchano, K. Blakespool, A. Howard, A. A. Rizzi, and M. Raibert, “Autonomous navigation for bigdog,” in *2010 IEEE international conference on robotics and automation*. Ieee, 2010, pp. 4736–4741.

[10] M. Ester, H.-P. Kriegel, J. Sander, X. Xu et al., “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[11] T. Dudzik, M. Chignoli, G. Bledt, B. Lim, A. Miller, D. Kim, and S. Kim, “Robust autonomous navigation of a small-scale quadruped robot in real-world environments,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 3664–3671.

[12] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposium Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[13] O. Melon, M. Geisert, D. Surovik, I. Havoutis, and M. Fallon, “Reliable trajectories for dynamic quadrupeds using analytical costs and learned initializations,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1410–1416.

[14] D. Surovik, O. Melon, M. Geisert, M. Fallon, and I. Havoutis, “Learning an expert skill-space for replanning dynamic quadruped locomotion over obstacles,” in *2020 Conference on Robot Learning (CoRL)*, 2020.

[15] R. H. Byrd, J. Nocedal, and R. A. Waltz, “Knitro: An integrated package for nonlinear optimization,” in *Large-scale nonlinear optimization*. Springer, 2006, pp. 35–59.

[16] M. Geisert, T. Yates, A. Orgen, P. Fernbach, and I. Havoutis, “Contact optimization and use case for rough terrain navigation,” in *Robot Operating System (ROS)*. Springer, 2016, pp. 99–120.

[17] J. H. Byrd, P. Lu, and J. Nocedal, “A limited memory algorithm for bound constrained optimization,” in *SIOPT*.

[18] J. Norby and A. M. Johnson, “Fast global motion planning for legged robots,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 3829–3836.

[19] J. Carpentier and N. Mansard, “Multicontact locomotion of legged robots,” *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1441–1460, 2018.

[20] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposium Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[21] D. Garcia-Castellanos and U. Lombardo, “Poles of inaccessibility: A calculation algorithm for the remotest places on earth,” *Scottish Geographical Journal*, vol. 123, no. 3, pp. 227–233, 2007.