Hybrid Sampling/Optimization-based Planning for Agile Jumping Robots on Challenging Terrains

Yanran Ding[1], Mengchao Zhang[1], Chuanzheng Li[1], Hae-Won Park[2], and Kris Hauser[3]

***Abstract*— This paper proposes a hybrid planning framework that generates complex dynamic motion plans for jumping legged robots to traverse challenging terrains. By employing a motion primitive, the original problem is decoupled as path planning followed by a trajectory optimization (TO) module that handles dynamics. A variant of a kinodynamic Rapidly-exploring Random Trees (RRT) planner finds a path as a parabola sequence between stance phases. To make this fast, a reachability informed control sampling scheme leverages a precomputed velocity reachability map. The path is post-processed to eliminate redundant jumps and passed to the TO module to find a dynamically feasible trajectory. Simulation results are presented where the proposed hybrid planner solves challenging terrains by executing multiple consecutive jumps, producing novel strategies to leap over large gaps by leveraging dynamics. In a physical experiment, the hybrid planner is tested on a real robot successfully traversing a challenging terrain.**

I. Introduction

Legged systems have the unique capability of making jumps to overcome challenging terrains with large height differences and wide gaps. Agile animals such as squirrels can plan complex dynamic maneuvers that fully utilize their inherent dynamics to jump over extremely difficult obstacle tracks. To rival nimble animals, many legged robots with high jumping capability have been developed [1], [2], [3], [4]. However, due to the differential constraints imposed by the robot dynamics and the hybrid nature inherent to locomotion, motion planning algorithms that can enable these jumping robots to traverse complex terrains are not as well developed.

Campana and Laumond proposed a ballistic motion planning algorithm [5] that can find a path with friction cone and velocity constraints in a complex 3D environment using Probabilistic Roadmaps (PRM) [6]. Inspired by [5], this work aims to improve upon the assumption of impulsive stance phase by explicitly addressing the stance dynamics, so that more innovative trajectory can be discovered. To achieve this goal, this work adopts the kinodynamic motion planning [7] view point, which respects the dynamics of the robot by imposing it as differential constraints. Sampling-based methods such as PRM and Rapidly-exploring Random Trees (RRT) [8] are widely used to solve large planning problems. However, the presence of certain differential constraints can severely compromise the efficiency of these algorithms. Reachability-guided RRT (RG-RRT) [9] increases the sampling efficiency by taking into consideration of local reachability. Implementing RG-RRT in task space and utilizing motion primitives enabled the LittleDog to bound over rough terrain [10]. To increase the efficiency of kinodynamic planners, Bézier curves [11] are used since they parametrize a trajectory with fewer parameters. On the other extreme, solving the whole trajectory with full dynamics results in a large trajectory optimization (TO) problem. TO has been widely used to generate dynamic motion plans for humanoids [12], [13] and quadrupeds [14], [4] since it handles the state and control constraints in a nonlinear program (NLP) formulation. However, the computation time increases drastically as the planning horizon increases.

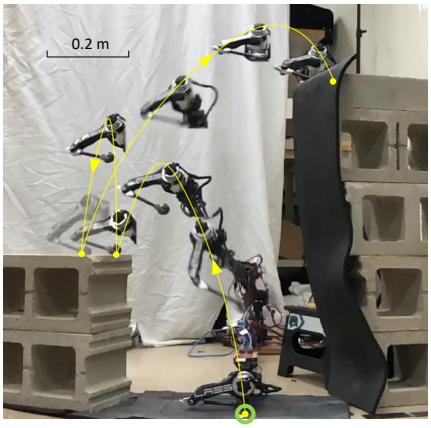

Fig. 1. Snapshots of the experiment where the robot executed the motion produced by the proposed hybrid planner, which re-directed the momentum of the robot in the second jump and surmounted the 0.9 m high platform. The robot is mounted on a boom system.

To exploit the advantages of both sampling and optimization-based methods, this paper proposes a hybrid sampling/optimization-based planner for generating dynamic motions for single-legged jumping robots to traverse challenging terrains. We decouple the original problem into sampling-based planning followed by a module that solves for the full dynamics using optimization. Similar to [5], aerial phases are constructed as parabolas connected by stance phases. Since the relationship between the touchdown and liftoff state is complex, a velocity reachability map is precomputed and used in the kinodynamic RRT. After a feasible path is found by the kinodynamic RRT and post-processed, trajectory optimization is performed at each stance to find

[1]Yanran Ding, Mengchao Zhang and Chuanzheng Li are with the Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, IL - 61801, USA. email: {yding35,mz17,cli67}@illinois.edu. Yanran Ding and Chuanzheng Li are supported by NSF 1752262 and NAVER LABS under grant 087387; Mengchao Zhang is supported by NSF 1911087.

[2]Hae-Won Park is with the Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology, Daejeon- 34141, South Korea. email: haewonpark@kaist.ac.kr.

[3]Kris Hauser is with the Department of Computer Science, University of Illinois at Urbana-Champaign, IL - 61801, USA. email: kkhauser@illinois.edu.

the state and control trajectories.

The proposed hybrid planner is benchmarked against a quasi-static planner and a mixed-integer convex program (MICP) based planner. Compared with the quasi-static planner, our method is momentum aware in the sense that it can find strategies where the robot makes consecutive jumps to gain momentum to clear a wide gap. Compared with the MICP-based planner, the solve time of the hybrid planner scales much better as the number of step increases. To validate the trajectory produced by the hybrid planner, a physical experiment is conducted on robot hardware and snapshots of the experiment are presented in Fig. 1.

II. Method

A. Dynamical Model

A point-mass model is employed since the physical robot of interest [15] has most of its mass lumped at the base, and the leg contributes to less than 10% of the total mass. The base of the robot is fixed at the end of a boom system and there is no torso pitch motion. Hence, the configuration space of the robot is its center of mass (CoM) position $\boldsymbol{p} = [x, z]^T \in \mathbb{R}^2$. The equation of motion (EoM) of the system is

$$\ddot{\boldsymbol{p}} = \frac{\boldsymbol{F}}{m} + \boldsymbol{a}_g, \tag{1}$$

where m is the mass of the robot, $\boldsymbol{a}_g = [0, -g]^T$ is the gravitational acceleration vector. The ground reaction force (GRF) $\boldsymbol{F} \in \mathbb{R}^2$ is parameterized by a Bézier spline with coefficients $\boldsymbol{\alpha}_F = [\boldsymbol{\alpha}_{Fx}, \boldsymbol{\alpha}_{Fz}]^T$. Then the state trajectory $\boldsymbol{x}(t) = [\boldsymbol{p}(t), \dot{\boldsymbol{p}}(t)]^T$ is also parametrized by a Bézier spline with coefficients $\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_{\dot{p}}$, which can be calculated by linear operations

$$\boldsymbol{\alpha}_{\dot{p}} = \mathcal{L}(\boldsymbol{\alpha}_F, \dot{\boldsymbol{p}}_0), \boldsymbol{\alpha}_p = \mathcal{L}(\boldsymbol{\alpha}_{\dot{p}}, \boldsymbol{p}_0), \tag{2}$$

where $\boldsymbol{p}_0, \dot{\boldsymbol{p}}_0$ are the initial CoM position and velocity, and the definition of the linear operator $\mathcal{L}(\cdot)$ can be found in [16]. The duration of the stance is denoted T_{st}.

During the stance phase, the CoM of the robot is constrained to lie within the workspace Ω, which is shown in Fig. 2 (b). The following inequalities delineate Ω,

$$\begin{aligned} \Omega := \{\boldsymbol{p} \mid\ & 0 \le p_z && \text{(3a)} \\ & l \le \|\boldsymbol{p} - \boldsymbol{p}_c\|_2 && \text{(3b)} \\ & r_{min} \le \|\boldsymbol{p}\|_2 \le r_{max} && \text{(3c)} \\ & \theta_{min} \le arg(\boldsymbol{p}) \le \pi - \theta_{min}\}, && \text{(3d)} \end{aligned}$$

where l is the thigh/shank linkage length; $\boldsymbol{p}_c$ is the point where the knee joint would be if it contacted the ground; r_{min} and r_{max} are the radial limits; θ_{min} is the minimum angular offset from the ground; $arg(\cdot)$ calculates the angle between vector $\boldsymbol{p}$ and the positive x axis.

Dynamic constraints such as torque limit and GRF constraints are also enforced. The joint torque $\boldsymbol{\tau} = [\tau_{hip}, \tau_{knee}]^T$ is mapped from GRF through $\boldsymbol{\tau} = \boldsymbol{J}^T\boldsymbol{F}$, where $\boldsymbol{J}$ is the Jacobian matrix; τ_{min} and τ_{max} are the torque limits. The friction cone $\mathcal{C}(\mu)$ is the set of GRF such that $|F_x| \le \mu F_z, F_z > 0$, where μ is the friction coefficient.

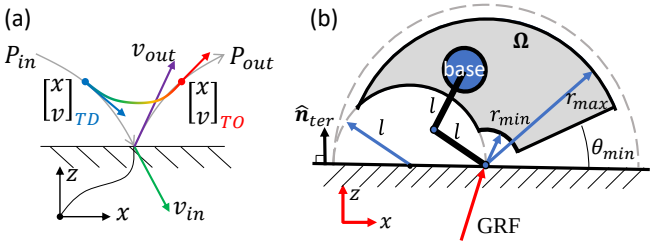

Fig. 2. (a) Illustration of the motion primitive. To connect two neighboring parabolas, the boundary states of TO are constrained on their corresponding parabola (gray curve). (b) The schematic of the single legged robot when it is in stance phase. The workspace Ω is the gray shaded area.

B. Hybrid Planning

The hybrid planning approach decomposes the complex kinodynamic motion planning problem into two stages. The first stage utilizes a variant of the kinodynamic RRT to find a sequence of parabolas that connects the start and the goal. The second stage applies TO at each stance to solve for the full dynamics. The motion primitive [17] that enables the hybrid framework is shown in Fig. 2. The extensions of the incoming parabola $\boldsymbol{P}_{in}$ and outgoing parabola $\boldsymbol{P}_{out}$ intersect at the contact point and are parametrized by $\boldsymbol{v}_{in}$ and $\boldsymbol{v}_{out}$, respectively. Note that these are variables that only pertain to the kinematic path, which is generated by the sampling-based planning stage, coinciding with Campana and Laumond's planner. The touchdown states of the TO $\boldsymbol{x}_{TD}$ and the liftoff state $\boldsymbol{x}_{LO}$ are chosen on $\boldsymbol{P}_{in}$ and $\boldsymbol{P}_{out}$, respectively. Therefore, the aerial phase determined by the first stage is preserved in the second stage. The added benefit is that since each stance is isolated, the multiple TOs could be parallelized.

If TO fails in the second stage, the failed jump is removed from the tree, and the sampling-based planning resumes to generate more TO candidates. We observe that failures are infrequent due to our use of a *velocity reachability map* within the sampling-based planning to generate feasible jumps with high likelihood.

C. Stance Trajectory Existence

A fundamental subproblem in our approach is a boundary value problem to determine whether a feasible trajectory at a stance can connect prescribed incoming and outgoing states. It can be formulated as a trajectory optimization problem, called TO[1], as follows:

$$\begin{aligned} \underset{\boldsymbol{\alpha}_F, T_{st}}{\text{minimize}} \quad & \sum_{k=1}^{N} \|\boldsymbol{\tau}_k\| \cdot T_{st} && \text{(4a)} \\ \text{subject to} \quad & \boldsymbol{p}_k \in \Omega && \text{(4b)} \\ & \tau_{min} \le \boldsymbol{\tau}_k \le \tau_{max} && \text{(4c)} \\ & \boldsymbol{x}_{TD} \in \boldsymbol{P}_{in}, \boldsymbol{x}_{LO} \in \boldsymbol{P}_{out} && \text{(4d)} \\ & \boldsymbol{F}_k \in \mathcal{C}(\mu). && \text{(4e)} \end{aligned}$$

Since the robot starts and ends at a static pose at the first and last jump, the initial position of the first jump and the final position of the last jump are only subject to the workspace constraint.

To make the problem finite-dimensional, the state and control trajectories are discretized at N sample points and subscript $(\cdot)_k$ indicates values at the k^{th} instance of the sampled time. The resulting optimization problem is a NLP. N is set to be a constant 20 in practice.

D. Velocity Reachability Map

Given an incoming velocity $\boldsymbol{v}_{in}$ at a stance, the velocity reachability map $\mathcal{R}(\boldsymbol{v}_{in})$ is defined as the set of $\boldsymbol{v}_{out}$ such that that $TO^1(\boldsymbol{v}_{in}, \boldsymbol{v}_{out})$ in (4) has a solution. The reverse reachability map $\mathcal{R}^{-1}(\boldsymbol{v}_{out})$ is defined similarly.

We precompute an approximation of the reachability map that is used in the sampling-based planner to greatly speed up planning by limiting connections so that they have a high probability of yielding a dynamically feasible trajectory. We approximate $\mathcal{R}(\boldsymbol{v}_{in})$ by running TO^1 over a 4D grid of $\boldsymbol{v}_{in}$, $\boldsymbol{v}_{out}$ and recording successes and failures. Even though an NLP could be trapped in local minima, TO^1 works sufficiently well for the small-scale problem here. For each $\boldsymbol{v}_{in}$, the successful $\boldsymbol{v}_{out}$ are approximated with a convex hull, which may under-approximate the true reachability at the margins but over-approximate it in convex regions. The same dataset is used to derive an approximation of $\mathcal{R}^{-1}$ in a similar fashion.

Fig. 3 presents an illustration of the velocity reachability map. The bottom area is the set of $\boldsymbol{v}_{in}$ with non-empty $\mathcal{R}(\boldsymbol{v}_{in})$. Each cell in the set of $\boldsymbol{v}_{in}$ is color coded by the area of $\mathcal{R}(\boldsymbol{v}_{in})$. Note that the set of valid $\boldsymbol{v}_{in}$ is asymmetric because the serial linkage leg of the robot bends towards one side. The $\mathcal{R}$ of two sample $\boldsymbol{v}_{in}$ are plotted on the top of Fig. 3, where the cross symbol represents that TO^1 can find a solution for the $\boldsymbol{v}_{in}, \boldsymbol{v}_{out}$ pair. The set $\mathcal{R}(\boldsymbol{v}_{in})$ is defined as the convex hull of the crossed points.

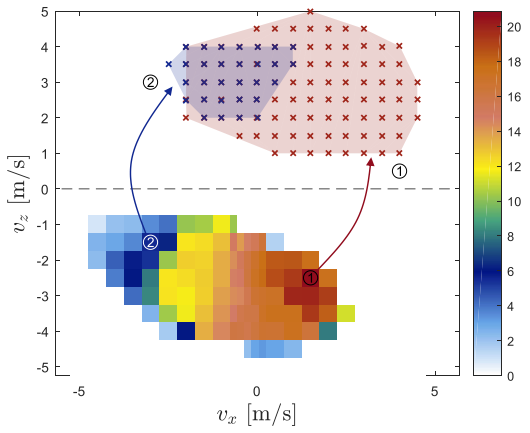

Fig. 3. An illustration of the reachability map $\mathcal{R}$. The bottom area is the set of $\boldsymbol{v}_{in}$ with a non-empty $\boldsymbol{v}_{out}$ set; the color at each $\boldsymbol{v}_{in}$ indicates the total area of the corresponding $\boldsymbol{v}_{out}$ set. Two sample $\boldsymbol{v}_{out}$ sets are shown at the top.

E. Sampling-Based Planning

Sampling-based planning forms the outer loop of the hybrid motion planner as shown in Algorithm 1. We use a variant of the kinodynamic RRT algorithm, modified with a reachability-informed control sampling scheme.

Algorithm 1 `Hybrid Motion Planner`

Input: $\mathcal{R}, \boldsymbol{x}_0, \boldsymbol{x}_g, Terrain$
Output: $Traj$ ▷ Jumping trajectory

$\mathcal{T}.init(\boldsymbol{x}_0)$ ▷ Initialize the RRT tree
$finished = False$
while not $finished$ **do**
  $\boldsymbol{x}_{rand} \leftarrow RandomSample(Terrain)$
  $\boldsymbol{x}_{parent} \leftarrow FindParent(\mathcal{T}, \boldsymbol{x}_{rand})$
  $\boldsymbol{x}_{new} \leftarrow STEER(\mathcal{R}, \boldsymbol{x}_{rand}, \boldsymbol{x}_{parent}, Terrain)$
  if $\boldsymbol{x}_{new} \neq null$ **then**
    $\mathcal{T}.add(\boldsymbol{x}_{parent} \rightarrow \boldsymbol{x}_{new})$
    if $ReachGoal(\boldsymbol{x}_{new})$ **then**
      $path = \mathcal{T}.FindPath()$
      $path^* = path.Shortcut()$
      $\boldsymbol{x}_{fail}, traj = TrajOpt(path^*)$
      if $\boldsymbol{x}_{fail} = null$ **then return** $traj$;
      else $\mathcal{T}.Trim(\boldsymbol{x}_{fail})$

A tree $\mathcal{T}$ is built from the initial state $\boldsymbol{x}_0$ by taking a random sample $\boldsymbol{x}_{rand} \in \mathbb{R}^3$ in the state-space ($RandomSample$). The dimensionality of $\boldsymbol{x}_{rand}$ is 3 because it involves the x-position and the incoming velocity. The parent node $\boldsymbol{x}_{parent}$ of the random sample $\boldsymbol{x}_{rand}$ is found ($FindParent$) based on a distance metric described in Section II-F. A steer function ($STEER$) attempts to extend from $\boldsymbol{x}_{parent}$ to $\boldsymbol{x}_{rand}$, and a new node $\boldsymbol{x}_{new}$ will be added to the tree if no collision was detected during the $STEER$ step. Otherwise, a sample will be drawn. Details of the $STEER$ function is presented in Algorithm 2 and in Section II-F. The kinodynamic RRT is terminated once the goal region has been reached ($ReachGoal$). Then a path $path$ will be extracted from the tree ($\mathcal{T}.FindPath$).

The path is post-processed to eliminate redundant jumps to get $path^*$ ($path.Shortcut$) as described in Section II-G. Trajectory optimization $TrajOpt$ is performed on the jumps in $path^*$ as described in Section II-H. If it succeeds, we are done, but if this fails, the failed state is returned. In this case, the sub-tree rooted at the failure state will be deleted and the random sampling resumes.

F. Reachability-Informed Control Sampling

A distance metric $\rho(x_1, x_2)$ is used to find the nearest neighbor of the sample. We use a weighted Euclidean distance with weights $[10, 30, 0.1, 0.01]$ to account for the importance of horizontal and vertical components of position and velocity. The weight ratio of p_x and p_z affects how much the planner prefers to stay on the same height, and the weight on v_x is higher than v_z because v_x determines the direction of the jump and is sensitive to friction limits. The $STEER$ function finds a collision-free parabola that goes from $\boldsymbol{x}_{parent}$ towards $\boldsymbol{x}_{rand}$ while considering the velocity reachability map. First, an outgoing velocity $\boldsymbol{v}_{guess}$ is obtained by kinematically connecting $\boldsymbol{p}_{parent}$ and $\boldsymbol{p}_{rand}$ without velocity constraint ($Connect$). Second, $\boldsymbol{v}_{guess}$ is projected ($Project$) to the reachable set $\mathcal{R}(v_{parent})$, which is a convex polytope. Hence, the projection can be performed by solving a quadratic program (QP) [18].

The $STEER$ function tries at most N_{try} times to obtain a collision-free parabola. The random samples from $SampleWithBias$ follow a Gaussian distribution centered at $\boldsymbol{v}_{proj}$ with standard deviation of σ, where σ is the distance between $\boldsymbol{v}_{proj}$ and the farthest vertex of the polytope $\mathcal{R}(\boldsymbol{v}_{parent})$. Any sample outside of the reachable set is rejected. The sampled velocity $\boldsymbol{v}_{sample}$ is used to project $\boldsymbol{x}_{parent}$ to the landing state $\boldsymbol{x}_{new}$ ($Projectile$). $\boldsymbol{x}_{new}$ is returned if the parabola does not induce collision. Otherwise, $null$ is returned to indicate collision.

Algorithm 2 STEER

Input: $\mathcal{R}, \boldsymbol{x}_{rand}, \boldsymbol{x}_{parent}, Terrain$
Output: $\boldsymbol{x}_{new}$

$\boldsymbol{v}_{guess} \leftarrow Connect(\boldsymbol{p}_{rand}, \boldsymbol{p}_{parent})$
$\boldsymbol{v}_{proj} \leftarrow Project(\boldsymbol{v}_{guess}, \mathcal{R}(\boldsymbol{v}_{parent}))$
for i = 1 to N_{try} **do**
  $\boldsymbol{v}_{sample} \leftarrow SampleWithBias(\boldsymbol{v}_{proj}, \mathcal{R}(\boldsymbol{v}_{parent}))$
  $collision, \boldsymbol{x}_{new} \leftarrow Projectile(\boldsymbol{x}_{parent}, \boldsymbol{v}_{sample}, Terrain)$
  if not $collision$ **then**
    return $\boldsymbol{x}_{new}$
return $null$

G. Path Shortcut

Once the goal region has been reached, a kinematic $path$ is extracted from the tree $\mathcal{T}$ using the method $\mathcal{T}.FindPath$. Since it is possible that $path$ involves redundant jumps, the $path.Shortcut$ method is applied to shortcut the path and reduce the solve time for the subsequent TO. The $Shortcut$ method is summarized in Algorithm 3. The node $\boldsymbol{x}_i$ attempts to connect with a node $\boldsymbol{x}_j (j > i)$ whose index starts from the end of $path$. If $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are successfully connected by the function $ConnectWith\mathcal{R}$, then the $Projectile$ function is called to check for collision. If the parabola is collision free, then the new node $\boldsymbol{x}_{new}$ is added to the $path^*$ and $\boldsymbol{v}^i_{out}, \boldsymbol{v}^j_{in}$ will be updated. Then the index i is given the value of j to indicate a shortcut. In the outer loop, $\boldsymbol{x}_i$ marches from the start of $path$ sequentially, until it reaches the length of the $path$ given by the method $path.Length$.

The function $ConnectWith\mathcal{R}$ solves an small nonlinear program that accounts for the velocity reachability map,

$$\underset{\boldsymbol{v}^i_{out}, \boldsymbol{v}^j_{in}, T_{air}}{\text{minimize}} \quad J \tag{5a}$$
$$\text{subject to} \quad \boldsymbol{v}^i_{out} \in \mathcal{R}(\boldsymbol{v}^i_{in}) \tag{5b}$$
$$\boldsymbol{v}^j_{in} \in \mathcal{R}^{-1}(\boldsymbol{v}^j_{out}) \tag{5c}$$
$$\boldsymbol{x}^j_{in} = \boldsymbol{\Phi}(\boldsymbol{x}^i_{out}, T_{air}), \tag{5d}$$

where the objective function J is set to a constant to form a feasibility problem.

H. TrajOpt

The parabola sequence $path^*$ from the sampling-based planner is solved in sequence using calls to TO^1 to connect each subsequent parabola with a stance phase. If TO^1 cannot solve a connection, a new TO that considers two consecutive jumps (TO^2) will be solved. This attempts to connect the incoming velocity from the current stance phase to the outgoing velocity in the next stance phase, which involves trajectory optimization over two stance phases and an aerial phase.

Algorithm 3 Path Shortcut

Input: $path, \mathcal{R}, Terrain$
Output: $path^*$

$path^*.init(path[0])$
$N \leftarrow path.Length(), i \leftarrow 0$
while $i < N$ **do**
  for $j = N$ **to** $i + 1$ **do**
    $\boldsymbol{x}_i \leftarrow path[i], \boldsymbol{x}_j \leftarrow path[j]$
    $success = False$
    $connected, \boldsymbol{v}_{out} \leftarrow ConnectWith\mathcal{R}(\boldsymbol{x}_i, \boldsymbol{x}_j, \mathcal{R})$
    if $connected$ **then**
      $collision, \boldsymbol{x}_{new} \leftarrow Projectile(\boldsymbol{x}_i, \boldsymbol{v}_{out}, Terrain)$
      if not $collision$ **then**
        $success = True$
    if $success = True$ **then**
      $path^*.add(\boldsymbol{x}_{new})$
      $i \leftarrow j$
      break
    else if $j = i + 1$ **then**
      $path^*.add(\boldsymbol{x}_j)$
      $i++$
return $path^*$

$$\underset{\boldsymbol{\alpha}^i_F, T^i_{st}, T_{air}, \boldsymbol{x}^1_{LO}, \boldsymbol{x}^2_{TD}}{\text{minimize}} \quad \sum_{i=1}^{2} \sum_{k=1}^{N} \|\boldsymbol{\tau}^i_k\| \cdot T^i_{st} \tag{6a}$$
$$\text{subject to} \quad \boldsymbol{p}^i_k \in \Omega \tag{6b}$$
$$\tau_{min} \leq \boldsymbol{\tau}^i_k \leq \tau_{max} \tag{6c}$$
$$\boldsymbol{x}^1_{TD} \in \boldsymbol{P}^1_{in}, \boldsymbol{x}^2_{LO} \in \boldsymbol{P}^2_{out} \tag{6d}$$
$$\boldsymbol{x}^2_{TD} = \boldsymbol{\Phi}(\boldsymbol{x}^1_{LO}, T_{air}) \tag{6e}$$
$$\boldsymbol{F}^i_k \in \mathcal{C}(\mu), \tag{6f}$$

where the superscript $i \in \{1, 2\}$ indicates the stance sequence. The aerial phase with aerial time T_{air} that connects stance 1 and 2 has the kinematic relationship $\boldsymbol{\Phi}$,

$$\boldsymbol{\Phi}(\boldsymbol{x}, T) = \begin{bmatrix} \boldsymbol{p} + \boldsymbol{v}T + \frac{1}{2}\boldsymbol{a}_g T^2 \\ \boldsymbol{v} + \boldsymbol{a}_g T \end{bmatrix}. \tag{7}$$

TO^2 can sometimes find trajectories that cannot be discovered by TO^1 since TO^2 has more relaxed constraints. If TO^2 cannot find a solution, the current stance state is treated as causing the failure and will be returned as $\boldsymbol{x}_{fail}$ to be pruned from the RRT tree.

III. Results

A. Computation Setup

The hybrid planning framework is formulated in MATLAB, and the TO^1 (4) and TO^2 (6) are formulated in CasADi [19] using the multiple-shooting method. The resulting nonlinear program (NLP) is solved by the solver ipopt [20]. The computational geometry calculation is done using the Mutli-Parametric Toolbox 3 (MPT3) [21]. The QPs for projection as described in Section II-F are solved by qpSWIFT [22]. The MICP-based planner in Section III-C is implemented using YALMIP [23] and solved by gurobi [24]. All of the simulation examples are run on a desktop with Intel i7 at 3.40 GHz.

B. Performance on Various Terrains

The proposed hybrid motion planner is tested on 5 different terrains, as shown in Fig. 4 and Fig. 5. The terrains are assumed to be represented by piecewise constant functions. The robot starts from the initial foot location (green circle) and tries to find a jumping path to reach the goal region (yellow box). Within each stance phase, the touchdown velocity $\boldsymbol{v}_{TD}$ is indicated by a blue arrow and the liftoff velocity $\boldsymbol{v}_{LO}$ a red arrow. The gray region shown in terrains (a) and (b) are the workspaces. The initial and final robot configurations at each stance are shown for terrains (a)-(d). Note that in terrain (d) segment 4, the planner adopted the strategy of taking intermediate jumps to re-orient the momentum of the robot in order to jump over the wide gap. Fig. 5 shows an example solution where the robot takes 9 jumps to reach the goal region. At stance 2, the TO^1 failed to find a solution, which is indicated by the black dot. Nevertheless, TO^2 found a feasible trajectory by simultaneously solving for stance 2 and 3 on segment 3. A zoom-in view is presented to illustrate stance 2 and 3, where the aerial trajectory is shown in a black dotted line. Fig. 6 (a) summarizes the solve time decomposition for the example terrains. Solve time results from 20 trials are averaged and the standard deviation is represented by the error bar. Terrain (a) takes the shortest time (5 s) and the terrain in Fig. 5 takes the longest time to solve (26 s). Fig. 6 (b) shows the average node number of the RRT tree. Please note that the y-axis is in log scale.

Although our method solves these problems in tens of seconds, the performance of the current implementation can be significantly improved. First, it is coded in MATLAB for rapid prototyping, so its run time can be reduced once re-written in a compile language. In addition, the run time can be further decreased if the TO at each stance was parallelized since each stance can be solved independently.

C. Benchmark

The proposed hybrid motion planning framework is compared with a quasi-static planner and a mixed-integer convex program (MICP) based planner. The three methods are tested in the scenario presented in Fig. 7 (a). The platform height $h, 2h$ and the gap width w are varied and the solve time results are shown in Fig. 7 (b)(c).

*1) **Quasi-static Planning**:* This method is similar to the proposed one except that it does not utilize the velocity reachability map introduced in Section II-D in the sampling stage. The quasi-static planning assumes that each step starts with zero velocity and can achieve maximum velocity v_{max} in every direction. The second assumption entails the assumed reachable region to be a parabolic envelope parametrized by v_{max} [25].

*2) **Mixed-Integer Convex Programs**:* The MICP-based method here plans consecutive jumps while considering actuator limits [26]. It is a resolution complete algorithm whose worst-case solve time increases drastically as the number of jumps increases. The number of jumps is set to 2 for the MICP-based planner to limit the solve time to be within the same order of magnitude as the other two methods. To achieve convex formulation, the torque limit constraint is relaxed and the solution is more conservative.

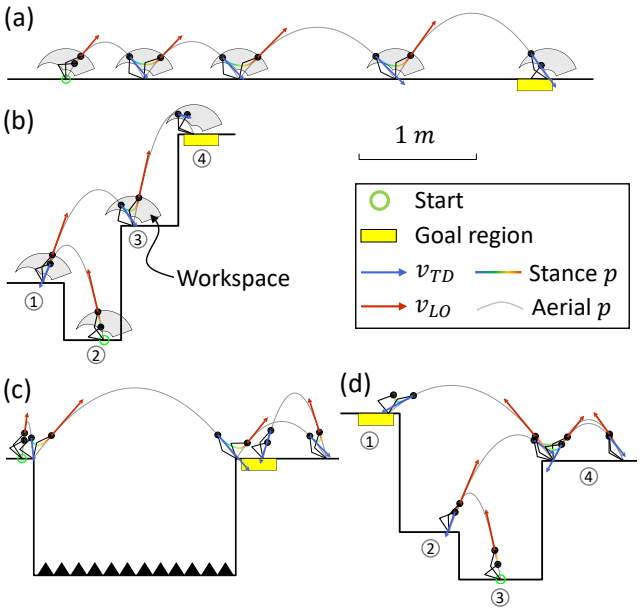

Fig. 4. Example terrains that are solved using the proposed hybrid motion planner. A scale is presented to show dimension. (a) a flat ground (b) three stairs: Left-Right-Right (c) a wide gap, where the region with saw teeth is forbidden (d) three stairs: Left-Right-Left.

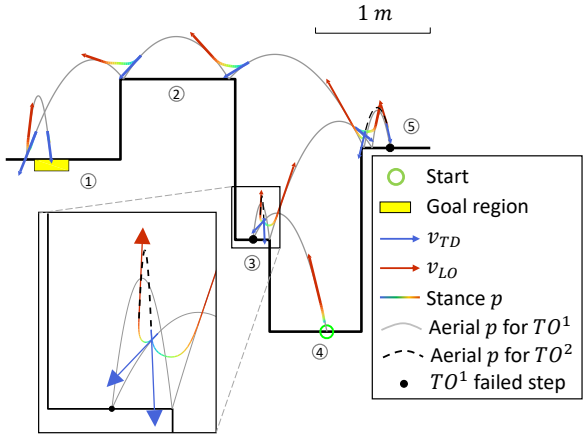

Fig. 5. The hybrid sampling/optimization-based planning algorithm can solve complex terrains as the one presented in this figure. The zoom-in part shows the case where TO^1 failed at stance 2, and TO^2 succeeded by solving for stance 2 and 3 simultaneously.

Fig. 7 (b) presents the solve time of the three methods where w is fixed at 0.4 m and h is varied from 0.2 m to 0.7 m. As can be observed from the figure, the MICP-based planner is slower than the other two methods and failed when h is higher than 0.45 m. In contrast, the quasi-static planner performs almost as well as the hybrid planner in the first scenario. Fig. 7 (c) shows the solve time comparison where h is fixed at 0.5 m and w is varied from 0.3 m to 0.6 m. The quasi-static planner failed at w =0.43 m due to its

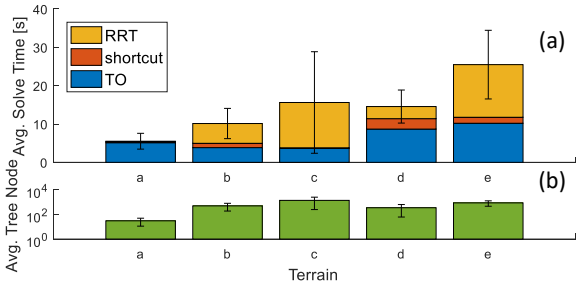

Fig. 6. Simulation results from 20 trials on the example terrains. The error bar represents one standard deviation. (a) The average solve time decomposed into three parts, RRT, shortcut, and TO. (b) The average node number of the tree.

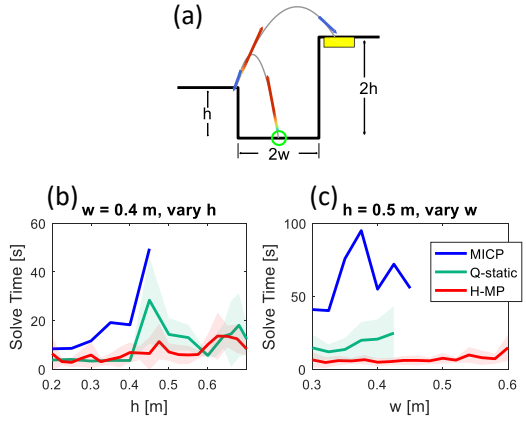

Fig. 7. Benchmark result of the hybrid planner, quasi-static planner and MICP-based planner. The former two are repeated for 10 times and the average is shown as solid line and shaded area represents one standard deviation. (a) The scenario for the benchmarking of the three methods (b) The testing case where w is fixed at 0.4 m and h is varied (c) The testing case where h is fixed at 0.5 m and w is varied. H-MP stands for the hybrid motion planner, and Q-static for quasi-static planner.

static assumption. MICP-based planner can solve a slightly wider gap at the cost of longer solve time. In comparison, the proposed hybrid planner finds solutions in all the tested scenarios using the shortest solve time.

These tests highlight the advantages of the proposed hybrid planning framework. Compared with the quasi-static planner, the proposed framework utilizes the velocity reachability map to reason about the momentum of the robot. Hence, the proposed method can come up with the strategy of taking intermediate jumps to re-direct its momentum to overcome wide gaps. MICP-based planner can plan for consecutive jumps but its solve time does not scale well as the number of jump increases due to the curse of dimensionality. Besides, the MICP-based planner produces conservative results due to the convex relaxation.

D. Experiment Setup

The 2 DoF robot leg [15] is fixed to the end of a boom system with a radius of 1.25 m. The position of the robot is measured by two encoders installed at the base of the boom. The moving mass is 1.1 kg and the link length of the robot is 0.14 m. The feedforward force profile from the hybrid planning algorithm is applied at the stance phases, and a PD controller is applied during the aerial phase to track foot swing trajectory. Proprioceptive contact detection was implemented to initiate stance phases.

E. Experiment Result

The terrain is set up such that the robot has to make use of the 0.4 m high platform on the left to reach the goal region on the 0.9 m high platform on the right. The snapshots of the experiment are presented in Fig. 1. The hybrid planner can come up with the strategy of making intermediate jumps on the left platform to re-direct its momentum to clear the wide gap and reach the goal region.

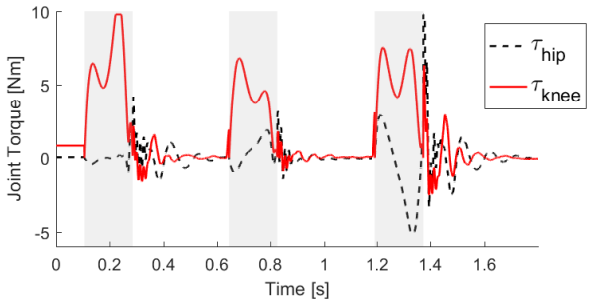

Fig. 8. The joint torque recording for the three consecutive jumps experiment. The shaded areas indicate stance phases.

The hip and knee joint torque trajectories for the three jumps experiment are shown in Fig. 8, where the stance phases are indicated by the gray areas. It can be observed that both hip and knee torques are within the torque limit (10 Nm). The oscillation after the stance phase is due to the rapid swing foot retraction to avoid collision with the environment.

IV. Conclusion

This paper presents a hybrid sampling/optimization motion planning algorithm for an agile single-legged robot to jump over challenging terrains. Under appropriate assumptions, the original kinodynamic motion planning problem could be decoupled into sampling and optimization stages. In the sampling stage, a variant of the kinodynamic RRT algorithm is employed to search for a kinematically feasible path as a sequence of parabolas. The pre-computed velocity reachability map is utilized to restrain the samples to be within a subset of the state space, which accelerates the algorithm by increasing the success rate of the subsequent TO. After a path shortcutting procedure, the optimization stage solves a TO problem at each jump for the dynamically feasible trajectory. The performance of the proposed hybrid motion planning algorithm is shown on various example terrains, and the advantage of this method is highlighted through benchmarking with two other methods. A trajectory generated by the proposed method is applied on a physical robot, which successfully traversed a challenging terrain by executing 3 consecutive jumps. The proposed hybrid planner is applicable to other single-legged robots such as SALTO to traverse more complex terrains.

REFERENCES

[1] J. K. Yim and R. S. Fearing, "Precision jumping limits from flight-phase control in salto-1p," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2229–2236.

[2] D. W. Haldane, M. M. Plecnik, J. K. Yim, and R. S. Fearing, "Robotic vertical jumping agility via series-elastic power modulation," *Science Robotics*, vol. 1, no. 1, 2016.

[3] J. Hwangbo, V. Tsounis, H. Kolvenbach, and M. Hutter, "Cable-driven actuation for highly dynamic robotic systems," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 8543–8550.

[4] B. Katz, J. Di Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6295–6301.

[5] M. Campana and J.-P. Laumond, "Ballistic motion planning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 1410–1416.

[6] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[7] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[8] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Report No. TR 98-11, Computer Science Department, Iowa State University.*, 1998. [Online]. Available: http://janowiec.cs.iastate.edu/papers/rrt.ps

[9] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 2859–2865.

[10] A. Shkolnik, M. Levashov, I. R. Manchester, and R. Tedrake, "Bounding on rough terrain with the littledog robot," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 192–215, 2011.

[11] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 2427–2433.

[12] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous robots*, vol. 40, no. 3, pp. 429–455, 2016.

[13] S. Dafarra, S. Bertrand, R. J. Griffin, G. Metta, D. Pucci, and J. Pratt, "Non-linear trajectory optimization for large step-ups: Application to the humanoid robot atlas," *arXiv preprint arXiv:2004.12083*, 2020.

[14] H.-W. Park, P. M. Wensing, S. Kim *et al.*, "Online planning for autonomous running jumps over obstacles in high-speed quadrupeds," *Robotics: Science and Systems*, 2015.

[15] Y. Ding and H.-W. Park, "Design and experimental implementation of a quasi-direct-drive leg for optimized jumping," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 300–305.

[16] 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)*.

[17] K. Hauser, T. Bretl, K. Harada, and J.-C. Latombe, "Using motion primitives in probabilistic sample-based planning for humanoid robots," in *Algorithmic foundation of robotics VII*. Springer, 2008, pp. 507–522.

[18] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[19] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[20] L. T. Biegler and V. M. Zavala, "Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization," *Computers & Chemical Engineering*, vol. 33, no. 3, pp. 575–582, 2009.

[21] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 502–510.

[22] A. G. Pandala, Y. Ding, and H. Park, "qpswift: A real-time sparse quadratic program solver for robotic applications," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3355–3362, 2019.

[23] J. Lofberg, "YALMIP: A toolbox for modeling and optimization in matlab," in *2004 IEEE international conference on robotics and automation (IEEE Cat. No. 04CH37508)*. IEEE, 2004, pp. 284–289.

[24] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[25] D. Donnelly, "The parabolic envelope of constant initial speed trajectories," *AmJPh*, vol. 60, no. 12, pp. 1149–1150, 1992.

[26] Y. Ding, C. Li, and H.-W. Park, "Single leg dynamic motion planning with mixed-integer convex optimization," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–6.