2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Path-Parameterised RRTs for Underactuated Systems

Damian Abood and Ian R. Manchester

Abstract— **We present a sample-based motion planning algorithm specialised to a class of underactuated systems using path parameterisation. The structure this class presents under a path parameterisation enables the trivial computation of dynamic feasibility along a path. Using this, a specialised state-based steering mechanism within an RRT motion planning algorithm is developed, enabling the generation of both geometric paths and their time parameterisations without introducing excessive computational overhead. We find with two systems that our algorithm computes feasible trajectories with higher rates of success and lower mean computation times compared to existing approaches.**

I. Introduction

Generating dynamically feasible motions is fundamental to the planning and control of robotic systems [1]. Underactuated systems such as flying and legged systems are fast becoming canonical to modern robotics, with growing applications towards remote inspection and surveillance of high-risk areas [2], [3]. A large challenge in motion planning for underactuated systems however is that their space of feasible motions is strongly constrained by their dynamics. Plans generated by motion planning algorithms must therefore exploit the natural dynamics of the system given this coupling [4].

A. Motion Planning Algorithms for Dynamic Systems

Local search strategies such as nonlinear programming involve the creation of mathematical programs from the associated kinodynamic constraints through transcription methods (e.g. direct collocation and multiple shooting [5]). The generality of transcription methods in handling kinodynamic constraints has enabled such programs to produce collision-free optimal trajectories for a range of mechanical systems such as vehicles [6], manipulators [7] and legged systems [8]. Generating plans within complex environments can however hinder these programs due to increased problem size and additional constraints, which lead to longer solve times and potentially an increase of local minima.

Sample-based methods are designed with complex environments in mind, with algorithms such as Rapidly-Exploring Random Trees (RRTs) [9] and Probabilistic Road Maps (PRMs) [10] demonstrating the ability to compute feasible solutions by exploring the available state space. Whilst feasible solutions are returned, they are almost always suboptimal unless additional routines are incorporated, as in RRT* [11].

Continuous optimisation approaches naturally include underactuated systems due to their generic handling of dynamic constraints. These dynamic constraints can however be more challenging to satisfy in comparison to fully actuated systems, particularly if poor initialisations are made [12]. Whilst sampling-based methods also extend to underactuated systems, the difficulty of creating feasible trajectories between sampled states can lead to longer computation times to arrive at a feasible solution [13]. To ensure branches within the tree respect the underactuated dynamics, these algorithms must have a well-designed steering routine.

B. Steering Methods

Control-based steering methods drive a system to a target state through the design of an appropriate control law, with the trajectory created by forward simulation becoming a new branch in the tree. Control laws can be as trivial as sampling from a fixed set of inputs and durations [13], encouraging exploration of the entire state space [14]. Selecting from a fixed pool of inputs can however make steering exactly to a target challenging. Optimal control methods such as LQR [15] and nonlinear programming [16] can be used to provide this exact steering to a target state. These methods do incur a much larger computational demand, which can become increasingly expensive to solve with higher dimensions [17].

State-based steering can also be used in the case of fully actuated systems, where instead a path is created between two points first and is assessed for its feasibility afterwards via inverse dynamics [18]. This path-driven approach has enabled path parameterisation, a technique that decomposes a trajectory into its geometric and temporal components [19], to be used for the dynamic evaluation of these created paths. Path parameterisation has been used extensively for creating time-optimal executions of fixed paths, with efficient algorithms catered to these problems using methods such as numerical integration [20], reachability analysis [21] and continuous optimisation [22]. The convexity of these fixed-path problems enables efficient implementations in conic programming [22] and interior-point methods [23]. Optimising the path in addition to its execution leads to a nonlinear program, which has been considered in [6] using a bilevel perspective. This method however requires paths to maintain feasibility at each iteration, which can be difficult for systems where their dynamics and geometric path are strongly coupled, such as those with underactuation.

The authors are with the Australian Centre for Robotics and the School of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, Australia. This research was supported in part by the ARC Research Hub for Intelligent Robotic Systems for Real-Time Asset Management (IH210100030). {damian.abood, ian.manchester}@sydney.edu.au

C. Steering Methods for Underactuated Systems

Compared to fully actuated cases, underactuated systems generally take longer computational time to compute solutions for when using control-based steering approaches, given most inputs for this class of system generate undesirable behaviours [13]. Optimal control methods can avoid this issue with their exact steering capabilities and have been shown to generate trajectories on low-dimensional models such as the acrobot [15].

Underactuated systems are far more challenging for state-based steering, as the designed paths must satisfy their underactuated dynamics exactly to be feasible. Dynamically infeasible paths are not suitable for path parameterisation methods, as these approaches are designed for fully or redundantly actuated systems [20]. Although underactuated systems were not considered, [14] presented an RRT-based algorithm *AVP-RRT* that demonstrated success in finding feasible trajectories for a double pendulum system with severe torque limitations. Branches in the tree were accepted depending on their success of their path parameterisations by admissible velocity propagation (AVP).

There are current limitations in adapting state-based steering approaches to underactuated systems, despite their prevalence in robotics. Path parameterisation has shown to be extendable to the class of underactuated systems with a single degree of underactuation, where dynamically feasible trajectories are those that follow their "decoupling vector fields" [24]. The focus of [24] was however to generate paths that were feasible for all time parameterisations, limiting the choice of system to those without potential effects such as gravity. This class of underactuated system is well renowned for its application in gait generation in planar walking systems [25], where [26] demonstrated an efficient and direct computation of the dynamic feasibility of generated paths through the path parameterisation of motion primitives. Primitive-based planning does however restrict the complexity of output motions, with existing works requiring transitioning between primitives at zero-velocity states [24] or ensuring the primitives are simple enough to enable efficient quadrature computations [26]. The same kinodynamic querying procedure of [26] could be used to assess the feasibility of motions with more dynamic behaviours such as those created under sample-based planning, as highlighted in [14].

D. Proposed Contributions

Under this motivation, we offer the following contributions within this paper

- We propose *UA1-RRT*, a path parameterisation-based random-search algorithm specialised for the motion planning of *underactuated degree-one* (UA1) systems.
- We present a state-based steering mechanism for the random sampling of geometric paths for UA1 systems, where no such method currently exists.
- We provide a configuration-space distance metric for nearest neighbours selection, which allows the distinguishing of points in configuration space with different velocities.

II. Preliminaries

A. Path Parameterising a Trajectory

Consider an n-dimensional mechanical system with configuration $\boldsymbol{q}$ and m inputs $\boldsymbol{u}$ with dynamics provided by

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{B}\boldsymbol{u} \tag{1}$$

where $\boldsymbol{M} \in \mathbb{R}^{n\times n}$, $\boldsymbol{C} \in \mathbb{R}^{n\times n}$, $\boldsymbol{G} \in \mathbb{R}^{n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times m}$. We assume that $\boldsymbol{q}$ contains only prismatic and revolute coordinates that can be readily mapped and "unwrapped" [27] respectively to $\mathbb{R}^n$. A trajectory $\boldsymbol{q}(t)$ can be path-parameterised such that an n-dimensional geometric path $\boldsymbol{\mathcal{P}}(s) : [0,1] \to \mathbb{R}^n$ and corresponding monotonic time parameterisation $s(t) : [0,T] \mapsto [0,1]$ are created to achieve $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$. Differentiating with respect to time, the generalised rates are consequently

$$\dot{\boldsymbol{q}} = \dot{s}\boldsymbol{\mathcal{P}}', \quad \ddot{\boldsymbol{q}} = \dot{s}^2\boldsymbol{\mathcal{P}}'' + \ddot{s}\boldsymbol{\mathcal{P}}' \tag{2}$$

Where $\boldsymbol{\mathcal{P}}' := \frac{d\boldsymbol{\mathcal{P}}}{ds}$ and $\dot{s} := \frac{ds}{dt}$ and likewise for other variables.

With the linearity of $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ in $\dot{\boldsymbol{q}}$ for mechanical systems [20], the substitution of (2) into (1) provides a representation of the dynamics under $s(t)$ that can be factored into an affine expression in $\ddot{s}$, $\dot{s}^2$ and $\boldsymbol{u}$

$$\boldsymbol{a}(s)\ddot{s} + \boldsymbol{b}(s)\dot{s}^2 + \boldsymbol{c}(s) = \boldsymbol{B}\boldsymbol{u} \tag{3}$$

with coefficient vectors $\boldsymbol{a}(s)$, $\boldsymbol{b}(s)$, $\boldsymbol{c}(s) \in \mathbb{R}^n$ given by

$$\begin{aligned}\boldsymbol{a}(s) &:= \boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'\\ \boldsymbol{b}(s) &:= \boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'' + \boldsymbol{C}(\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{P}}')\boldsymbol{\mathcal{P}}'\\ \boldsymbol{c}(s) &:= \boldsymbol{G}(\boldsymbol{\mathcal{P}})\end{aligned}$$

where dependencies on s and t have been removed for brevity.

We make the following definition regarding dynamic feasibility for systems under a time parameterisation $s(t)$:

Definition 2.1 (Dynamic Feasibility): A geometric path $\boldsymbol{\mathcal{P}}(s)$ is dynamically feasible for the system (1) if there exists a strictly monotonic time parameterisation $s(t) : [0,T] \mapsto [0,1]$ with $s(0) = 0$ and $s(T) = 1$ such that (3) is satisfied for the duration of the trajectory, an equivalent condition being $\dot{s}(t) > 0\ \forall t \in (0,T)$. If no such $s(t)$ exists, the path is *dynamically infeasible*.

B. Underactuated Degree-One Systems

Underactuated degree-one (UA1) systems are mechanical second-order systems where $\mathrm{rank}(\boldsymbol{B}) = n-1$ in (1). By an appropriate choice of coordinates, the dynamics of these systems can be written in the canonical form

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix}0\\ \boldsymbol{\tau}\end{bmatrix} \tag{4}$$

where we now consider the generalised inputs $\boldsymbol{\tau} \in \mathbb{R}^{n-1}$ to the system (i.e. $[0\ \boldsymbol{\tau}^T]^T = \boldsymbol{B}\boldsymbol{u}$)

III. Solving Path Parameterisations for UA1 Systems

We consider the problem of generating trajectories $\boldsymbol{q}(t) : [0, T] \to \mathbb{R}^n$ for UA1 systems within configuration space $\mathcal{Q} \subset \mathbb{R}^n$ connecting an initial state $(\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0)$ to a goal state $(\boldsymbol{q}_g, \dot{\boldsymbol{q}}_g)$. These trajectories must satisfy the dynamics (4) whilst respecting velocity and actuation bounds

$$\dot{\boldsymbol{q}}_L \leq \dot{\boldsymbol{q}}(t) \leq \dot{\boldsymbol{q}}_U, \ \boldsymbol{\tau}_L \leq \boldsymbol{\tau}(t) \leq \boldsymbol{\tau}_U \tag{5}$$

The approach taken finds geometric path $\boldsymbol{\mathcal{P}}(s) : [0, 1] \to \mathbb{R}^n$ joining $\boldsymbol{q}_0$ to $\boldsymbol{q}_g$ within $\mathcal{Q}$ with corresponding time parameterisation $s(t) : [0, T] \to [0, 1]$ such that $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$ satsifies (4) and (5).

Under a path parameterisation $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$, the dynamics (4) can be split as follows

$$\begin{bmatrix} a_u \\ \boldsymbol{a}_a \end{bmatrix} \ddot{s} + \begin{bmatrix} b_u \\ \boldsymbol{b}_a \end{bmatrix} \dot{s}^2 + \begin{bmatrix} c_u \\ \boldsymbol{c}_a \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{\tau} \end{bmatrix} \tag{6}$$

where the coefficient vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ are partitioned into their actuated $(\boldsymbol{a}_a, \boldsymbol{b}_a, \boldsymbol{c}_a \in \mathbb{R}^{n-1})$ and underactuated $(a_u, b_u, c_u \in \mathbb{R})$ components. The single passive degree of freedom in (6) forces the evolution of $s(t)$ to be driven purely by the underactuated dynamics, that is $s(t)$ must satisfy

$$a_u(s)\ddot{s} + b_u(s)\dot{s}^2 + c_u(s) = 0 \tag{7}$$

Drawing parallels to virtual constraints [25] can this be viewed as the resulting zero-dynamics of the system when applying the virtual constraints $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$. The constraint (7) imposed by underactuation implies $s(t)$ can be solved by numerical quadrature of (8), only requiring a single forward pass as opposed to the forwards and backwards passes necessary for classical TOPP methods [19], [20].

Given an initial path velocity $\dot{s}_0 \geq 0$, we choose to solve for the profile $\dot{s}^2(s) := \theta(s)$ over the path domain, which when applying the identity $\ddot{s} = \frac{1}{2}\frac{d}{ds}\dot{s}^2$ presents a desirable form of (7) as a first order linear differential equation

$$a_u(s)\theta' + 2b_u(s)\theta + 2c_u(s) = 0 \tag{8}$$

where we can numerically integrate this system from $\theta_0 = \dot{s}_0^2$ with a step size of Δs. At each point s_i can the kinodynamic quantities $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$ be recovered through (2) and the remaining $n-1$ equations of (6) using the computed $\theta(s_i)$, $\theta'(s_i)$ (using (8)) and $\boldsymbol{\mathcal{P}}(s_i)$.

A. Dynamic Feasibility of a Path

In UA1 systems, $\dot{s}(t) < 0$ presents a physical significance given $\dot{s}(t)$ is an indication of the available kinetic energy along the provided path [26]. If $\dot{s}(t) < 0$, then insufficient kinetic energy exists for the underactuated degree to complete motion along the path and instead "falls back" on itself. Dynamic feasibility conditions under $\theta(s)$ are identical to $\dot{s}(s)$ such that provided an initial path velocity $\dot{s}(0) = \dot{s}_0 \geq 0$, zero-crossings of $\theta(s)$ indicate the path is dynamically infeasible beyond the crossing. The forward integration scheme for θ enabled by (7) allows for zero-crossings in θ to be detected immediately, allowing rapid detection of path infeasibility and termination of the integration. As implied by (8), zero-inertia points s_z (i.e. $a_u(s_z) = 0$ in (8)) will always correspond to dynamic singularities where $\theta'(s_z)$ is undefined. To avoid unwanted behaviour in $\theta(s)$, we approximate $\theta'(s_z)$ at these points by finite-differencing local to $s = s_z$.

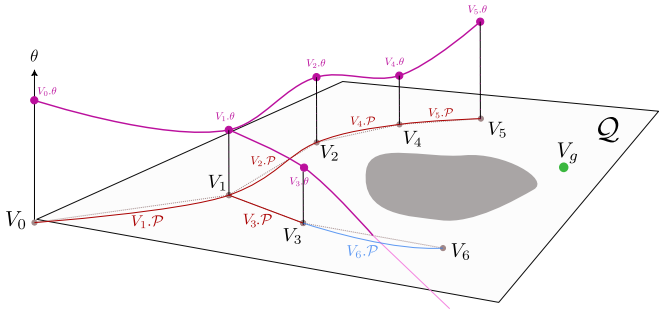


Fig. 1: Visualisation of tree-growth in UA1-RRT, with edges $V_i.\boldsymbol{\mathcal{P}}$ connecting vertices V_i in obstacle-free configuration space $\mathcal{Q}$. Path are added as edges to the tree if their path rate squared θ (magenta) remains positive (red) and discarded if a zero-crossing is encountered (blue).

IV. UA1-RRT Routine

In this section, we present the main contribution of this paper, the UA1-RRT algorithm. We adopt a path-parameterised perspective to the trajectory planning problem, breaking the problem into the coupled subproblems of configuration space planning and time parameterisation of the resulting paths.

Our algorithm follows the general structure of AVP-RRT [14], using a standard RRT procedure [9] to create a collision-free geometric path in configuration space by iteratively constructing the path in segments and computing the trajectory along each segment via path parameterisation as per Sec. III. Whilst optimal tree-based searches exist, the typical rewiring step of optimal tree search methods [15] poses difficulty with underactuated systems due to the challenge in generating feasible motions between states. The key difference of our proposed approach to AVP-RRT is that due to the constraint of underactuation in (6), the feasible velocity profile $\dot{s}(t)$ for a path segment $\boldsymbol{\mathcal{P}}(s)$ is unique as opposed to there being an interval of feasible path velocities [17]. UA1-RRT constructs configuration-space trees $\mathcal{T}$ comprised of vertices V that connect to other vertices with geometric paths (Fig. 1), with trajectories generated through the path parameterisation method of Sec. III. A vertex V therefore requires the following data: $V.\boldsymbol{q}$ - the location of the vertex in configuration space $\mathcal{Q}$; $V.\lambda$ - the parent vertex to V; $V.\boldsymbol{\mathcal{P}}$ - the connecting geometric path from $V.\lambda$ to V; $V.\theta$ - the path rate-squared at $V.\boldsymbol{q}$ when moving along $V.\boldsymbol{\mathcal{P}}$.

A. Algorithm Overview

We present our algorithm UA1-RRT in Algorithm 1, with a graphical depiction of the process shown in Fig. 1. The tree $\mathcal{T}$ is initialised with a root vertex V_0 at the initial configuration $\boldsymbol{q}_0$. With desired initial velocity $\dot{\boldsymbol{q}}_0$ and path

Algorithm 1 UA1-RRT

Input: $\boldsymbol{q}_0$, $\dot{\boldsymbol{q}}_0$, $\dot{s}_0$, $\boldsymbol{q}_g$, $\dot{\boldsymbol{q}}_g$
Output: Trajectory $\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)$ joining $\boldsymbol{q}_0$ to $\boldsymbol{q}_g$

1: $V_0.\boldsymbol{q} \leftarrow \boldsymbol{q}_0$, $V_0.\theta \leftarrow \dot{s}_0^2$, $V_0.\boldsymbol{\mathcal{P}}'(0) \leftarrow \dot{\boldsymbol{q}}_0/\dot{s}_0$
2: $\mathcal{T} = \{V_0\}$
3: **for** $i = 1, 2, \ldots, N$ **do**
4: $\quad \boldsymbol{q}_r = \text{RANDOMCONFIGURATION}()$
5: $\quad V_e = \text{EXTEND}(\mathcal{T}, \boldsymbol{q}_r)$ ▷ Alg. 2
6: $\quad$ **if** $V_e \neq \emptyset$ **then**
7: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{V_e\}$
8: $\quad$ **end if**
9: $\quad$ **if** $\text{REACHGOAL}(\mathcal{T}, \boldsymbol{q}_g, \dot{\boldsymbol{q}}_g)$ successful **then**
10: $\quad\quad (\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) = \text{COMPUTETRAJECTORY}(\mathcal{T})$
11: $\quad\quad$ **return** $(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))$
12: $\quad$ **end if**
13: **end for**

Algorithm 2 EXTEND

Input: Tree $\mathcal{T}$, configuration $\boldsymbol{q}_r$ to extend $\mathcal{T}$ towards
Output: A new vertex to be added to $\mathcal{T}$ or $\emptyset$

1: **function** $\text{EXTEND}(\mathcal{T}, \boldsymbol{q}_r)$
2: $\quad X = \text{NEAREST}_k(\mathcal{T}, \boldsymbol{q}_r), Y = \emptyset$
3: $\quad$ **for** $x_i \in X$ **do**
4: $\quad\quad \delta\boldsymbol{q} = \boldsymbol{q}_r - x_i.\boldsymbol{q}$, $\hat{\delta\boldsymbol{q}} = \delta\boldsymbol{q}/\left\|\delta\boldsymbol{q}\right\|$
5: $\quad\quad$ **if** $\left\|\delta\boldsymbol{q}\right\|_2 > D_{\max}$ **then**
6: $\quad\quad\quad \boldsymbol{q}_r \leftarrow x_i.\boldsymbol{q} + D_{\max}\hat{\delta\boldsymbol{q}}$
7: $\quad\quad$ **end if**
8: $\quad\quad V_s = \text{STEER}(x_i, \boldsymbol{q}_r)$ ▷ Alg. 3
9: $\quad\quad$ **if** $V_s \neq \emptyset$ and $\text{ISOBSTACLEFREE}(V_s.\boldsymbol{\mathcal{P}})$ **then**
10: $\quad\quad\quad Y = Y \cup \{V_s\}$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad$ **return** $\text{argmin}_{y \in Y} \left\|\boldsymbol{q}_r - y.\boldsymbol{\mathcal{P}}(1)\right\|_2$
14: **end function**

velocity $\dot{s}_0 \geq 0$, $V_0.\boldsymbol{\mathcal{P}}'(0)$ and rate-squared term $V_0.\theta$ are computed such that $\dot{\boldsymbol{q}}_0 = \sqrt{V_0.\theta}V_0.\boldsymbol{\mathcal{P}}'(0)$. We arbitrarily select $V_0.\theta = \dot{s}_0^2$ and $V_0.\boldsymbol{\mathcal{P}}'(0) = \dot{\boldsymbol{q}}/\dot{s}_0$ to satisfy this. In cases where $\dot{\boldsymbol{q}}_0 = \boldsymbol{0}$, we also have the option of $\dot{s}_0 = 0$ and $V_0.\boldsymbol{\mathcal{P}}'(0) \in \mathbb{R}^n$. RANDOMCONFIGURATION (Alg. 1 L4) generates a randomly sampled obstacle-free point $\boldsymbol{q}_r \in \mathcal{Q}$ for $\mathcal{T}$ to extend towards. REACHGOAL (Alg. 1 L9) checks whether a vertex $V \in \mathcal{T}$ is sufficiently close to the goal state, and if so, terminates the program. On termination, the final trajectory is computed by COMPUTETRAJECTORY (Alg. 1 L10) which successively moves through each vertex of the branch connecting root to goal, concatenating the edges of each vertex to create the path of the trajectory. $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ are then computed by solving (8) for the provided path and initial path velocity. We now detail the EXTEND and STEER routines of UA1-RRT.

B. Extension

The EXTEND procedure (Alg. 1 L5) attempts to create a new vertex which is as close as possible to target configuration $\boldsymbol{q}_r$ to be added to $\mathcal{T}$, with details of the routine shown in Alg. 2. To select suitable vertices within the tree to be extended from to the target point $\boldsymbol{q}_r$, NEAREST_k (Alg. 2 L2) returns a set of k vertices $X \subset \mathcal{T}$ which are closest to $\boldsymbol{q}_r$ by a distance metric. The distance metrics in AVP-RRT used the Euclidean distance within the configuration space, with the option to include the final path orientation [14].

Where repetitive motions or high-velocity manoeuvres are required, incorporating the orientation/velocity in these metrics is essential, given vertices that are close together in configuration space but with different velocities are indistinguishable. We propose the metric d_p that considers the difference between $\boldsymbol{q}_r$ and the projection of a vertex's configuration $V_i.\boldsymbol{q}$, $V_i \in \mathcal{T}$ under its current velocity for a user-defined duration $\gamma \in \mathbb{R}^+$. We define this projected point $\boldsymbol{q}_p \in \mathbb{R}^n$ as

$$\boldsymbol{q}_p = V_i.\boldsymbol{q} + \gamma\sqrt{V_i.\theta}V_i.\boldsymbol{\mathcal{P}}'(0) \tag{9}$$

which allows us to write the metric d_p (normalised) as

$$d_p = \sum_{i=1}^{n} \frac{\left\|q_{i,r} - q_{i,p}\right\|^2}{\left\|q_{i,\max}\right\|^2} \tag{10}$$

This metric is well suited towards state-based steering as it will prioritise vertices in $\mathcal{T}$ that have the most potential for the system to reach $\boldsymbol{q}_r$ from (or at least move in the direction of $\boldsymbol{q}_r$) based on their dead reckoning over horizon γ (i.e. $\boldsymbol{q}_p$).

Each vertex $x_i \in X$ is then steered towards the target point $\boldsymbol{q}_r$. A maximum extension distance $D_{\max}$ is enforced to regulate tree growth such that extensions for the tree do not exceed this distance (Alg. 2 L4-7). STEER (Alg. 2 L8) returns a vertex V_s with a feasible edge connected to $x_i \in \mathcal{T}$ extending as close as possible to $\boldsymbol{q}_r$ (details in Sec. IV-C).

ISOBSTACLEFREE (Alg. 2 L9) performs collision checking of the path $V_s.\boldsymbol{\mathcal{P}}$ with any configuration-space obstacles (shaded region of $\mathcal{Q}$ in Fig. 1). If no collisions are made, we add V_s to the candidate vertex set Y (Alg. 2 L10). After all vertices in X are steered towards, the closest vertex in Y to $\boldsymbol{q}_r$ by Euclidean distance is returned as the new vertex to be added to $\mathcal{T}$ (Alg. 2 L13).

To encourage growth towards the goal region, we periodically call EXTEND (Alg. 1 L5-8) with $\boldsymbol{q}_r = \boldsymbol{q}_g$ [14]. As the tree grows closer to the goal region over the duration of the program, the projected point $\boldsymbol{q}_p$ in (10) will likely overshoot the goal when using a fixed γ. To account for this, we set $\gamma = 0$ for d_p in (10) for this extension process to base distance purely on proximity to the goal in $\mathcal{Q}$. Together with the original EXTEND procedure, we offer a search method that encourages dynamically conscious tree growth in configuration space coupled with refined goal-reaching capabilities once significantly extended.

C. Steering

Our state-base steering approach motivated by the use of the path parameterisation allows us to create connecting paths $\boldsymbol{\mathcal{P}}(s)$ first and then determine whether they can be

Algorithm 3 STEER

Input: Vertex $x_0 \in \mathcal{T}$, target configuration $\boldsymbol{q}_r$
Output: A vertex at $\boldsymbol{q}_r$ with parent x_0 or $\emptyset$

function STEER(x_0, $\boldsymbol{q}_r$)
  $Y = \emptyset$
  for $i = 1, 2, \ldots, N_{rndm}$ **do**
    $\boldsymbol{\mathcal{P}}(s) = \text{GENERATEPATH}(x_0, \boldsymbol{q}_r)$
    $(s^\star, \theta^\star) = \text{PATHPROFILE}(\boldsymbol{\mathcal{P}}(s), x_0.\theta)$ ▷ Sec. III
    if $s^\star \geq s^\dagger$ **then**
      $y.\boldsymbol{\mathcal{P}} \leftarrow \{\boldsymbol{\mathcal{P}}(s) \mid \forall s \in [0, s^*]\}$
      $y.\theta = \theta^\star$
      $Y = Y \cup \{y\}$
    end if
  end for
  return $\text{argmin}_{y \in Y} \|\boldsymbol{q}_r - y.\boldsymbol{\mathcal{P}}(1)\|_2$
end function

added to $\mathcal{T}$ if their corresponding rate-squared profile $\theta(s)$ (by III) complies with kinodynamic constraints (1) and (5). Using a path parameterisation also has the additional freedom that the execution time of a path is encoded by the profile $\theta(s)$, normally a difficult parameter to tune in the RRT method [13] without resorting to a sophisticated control policy.

To create geometric paths $\boldsymbol{\mathcal{P}}(s)$, we require they be smooth with $\mathcal{C}^1$ continuity such that $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ are continuous. Adding a new vertex to the tree must therefore ensure its path created by STEER (Alg. 2 L8, Alg. 3) is $\mathcal{C}^1$-continuous to the path of its parent vertex. An exception to these continuity requirements is where $\dot{\boldsymbol{q}} \approx 0$ at an existing vertex in the tree. In these cases, the initial tangent vector of the new vertex's path can ignore this continuity requirement, provided we have $\theta = 0$ for this point.

GENERATEPATH (Alg. 3 L4) creates paths $\boldsymbol{\mathcal{P}}(s)$ which meet these continuity requirements. Paths are created that connect a vertex $x_0 \in \mathcal{T}$ to configuration $\boldsymbol{q}_r$ such that $\boldsymbol{\mathcal{P}}(s)$ is tangent to the tree at $x_0.\boldsymbol{q}$. Whilst any polynomial of degree three or higher meets this criterion, GENERATEPATH selects paths from the family of cubic polynomials with fixed end-point positions and initial gradients. These polynomials have one remaining degree of freedom which we use to parameterise this family of paths, which we choose to be the gradient at the end of the path. Randomly selecting the gradient within a user-defined range is performed on each call to GENERATEPATH, providing a randomised $\mathcal{C}^1$-continuous path $\boldsymbol{\mathcal{P}}(s)$ steering from $x_0.\boldsymbol{q}$ to $\boldsymbol{q}_r$.

As shown in III, the velocity profile $\dot{\boldsymbol{q}}(t)$ for UA1 systems is determined purely by the path $\boldsymbol{\mathcal{P}}(s)$ and initial velocity $\dot{s}_0$. Ensuring bounds such as (5) are respected therefore requires the appropriate $\boldsymbol{\mathcal{P}}(s)$ if $\dot{s}_0$ is already given. From our proposed GENERATEPATH, it is expected that most paths will only partially respect these bounds (5). Despite this limitation, we can exploit the uniqueness of $s(t)$ for a given path by truncating paths $\boldsymbol{\mathcal{P}}(s)$ beyond a point $s^\star > 0$ (i.e. $\boldsymbol{\mathcal{P}}(s) \leftarrow \{\boldsymbol{\mathcal{P}}(s) \mid \forall s \in [0, s^\star]\}$). We choose the point $s^\star$ such that the truncated path is dynamically feasible by (7) whilst adhering to the imposed kinodynamic bounds (5). Formally, we define the set of admissible path rates that respect (5) as $\mathcal{S}_{\boldsymbol{\mathcal{P}}}(s)$, given by

$$\mathcal{S}_{\boldsymbol{\mathcal{P}}}(s) = \{(\dot{s}, \ddot{s}) \mid \dot{\boldsymbol{q}}_L \leq \dot{\boldsymbol{q}}(s) \leq \dot{\boldsymbol{q}}_U, \boldsymbol{\tau}_L \leq \boldsymbol{\tau}(s) \leq \boldsymbol{\tau}_U\} \tag{11}$$

where $\dot{\boldsymbol{q}}(s)$ and $\boldsymbol{\tau}(s)$ are computed with (2) and (6) respectively. With this, we can define $s^\star$ as the largest value in $s \in [0, 1]$ such that $\boldsymbol{\mathcal{P}}(s)$ over the interval $[0, s^\star]$ is dynamically feasible by (7) and the rates of its corresponding time parameterisation $s(t)$ are contained within $S_{\boldsymbol{\mathcal{P}}}(s)$. Our definition for $s^\star$ does not include paths that encounter dynamic infeasibility (Def. 2.1), as we believe creating paths that terminate at zero crossings of θ lead to poor future extensions since they indicate paths of insufficient kinetic energy to complete future motion [26]. This is demonstrated in Fig. 1, where the branch from V_3 to V_6 is discarded given a zero crossing of θ is encountered along $V_6.\boldsymbol{\mathcal{P}}$. The search for $s^\star$ is performed within the numerical integration of $\theta(s)$ (PATHPROFILE) and is returned upon termination of the integration.

A cut-off threshold $0 < s^\dagger \leq 1$ is set so that paths with $s^\star < s^\dagger$ are discarded to avoid slow tree growth. In our implementation, we choose the fairly conservative value $s^\dagger = 5\Delta s$. To increase the likelihood of returning a vertex with a feasible path, we create N_{rndm} random paths to assess. After all N_{rndm} paths are computed and parameterised, STEER returns the vertex whose path terminates closest to $\boldsymbol{q}_r$ in the Euclidean sense.

D. Terminal Conditions

REACHGOAL (Alg. 1 L9) determines if any vertex in $\mathcal{T}$ is within a user-defined distance of the goal $(\boldsymbol{q}_g, \dot{\boldsymbol{q}}_g)$ and signals termination of the program if so. Given the terminal state $\boldsymbol{q} = V.\boldsymbol{q}$ and $\dot{\boldsymbol{q}} = \sqrt{V.\theta} V.\boldsymbol{\mathcal{P}}'(1)$ of a vertex $V \in \mathcal{T}$, normalised goal metrics for prismatic (T) and revolute (R) coordinates are computed as

$$d_{g,i} = \begin{cases} \frac{\|q_{g,i} - q_i\|}{\|q_{i,\max}\|} & i \in T \\ (1 - \cos(q_{g,i} - q_i)) & i \in R \end{cases} \tag{12}$$

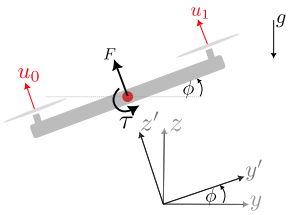
Together with the normalised velocity error, the overall distance-to-goal is then

$$d_{goal} = \frac{1}{2n}\left(\sqrt{\sum_{i=1}^{n} d_{g,i}^2} + \sqrt{\sum_{i=1}^{n} \frac{(\dot{q}_{g,i} - \dot{q}_i)^2}{\dot{q}_{i,\max}^2}}\right) \tag{13}$$

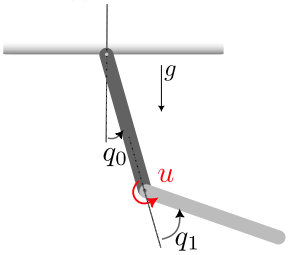
For a distance threshold ϵ_g, the tree has reached the goal if a vertex $V \in \mathcal{T}$ returns a goal metric $d_{goal} \leq \epsilon_g$. We select a value of $\epsilon_g = 10^{-2}$ to provide sufficient accuracy in the resulting trajectories akin to AVP-RRT [14].

V. SIMULATION EXPERIMENTS

We considered two UA1 systems for the demonstration of our method (Fig. 2), a planar UAV that must navigate a $4m$ obstacle-filled tunnel and an acrobot performing a swing-up procedure.



(a) UAV model.



(b) Acrobot model.

Fig. 2: Underactuated degree-one (UA1) example systems.

UAV: We chose the physical parameters $m = 0.1\,kg$, $I_{xx} = 1 \times 10^{-4}\,kgm^2$, with uni-directional thruster limits of $1\,N$. We imposed maximum translational and rotational velocities of $20\,ms^{-1}$ and $50\,rad/s$ respectively. The dynamics of the UAV within the inertial frame (y, z, ϕ) along with inputs (u_0, u_1) (Fig. 2a) does not readily produce dynamics of the form (6). Instead, we consider the dynamics in the body frame (y', z', ϕ) with the net translational and rotational forces on the body $(\boldsymbol{F}, \boldsymbol{\tau})$ as inputs to the system. From this, the passive degree of freedom is the lateral axis y' as no input can create additional acceleration in this direction. We specify the endpoints of the trajectory to be the start and end of the tunnel with target velocities $\dot{\boldsymbol{q}}_0 = \dot{\boldsymbol{q}}_g = \mathbf{0}$. For tree search parameters, we selected $D_{\max} = 1.0$ and $\gamma = 1.0$.

Acrobot: The physical parameters for the acrobot are based on those of [14] now with a purely passive shoulder and an actuated elbow with torque limit $|u| \leq 50\,Nm$. We have initial conditions $\boldsymbol{q}_0 = \dot{\boldsymbol{q}}_0 = \mathbf{0}$ and velocity bounds $|\dot{\boldsymbol{q}}_{0,1}| \leq 50\,rad/s$. Our acrobot can swing any number of rotations around its shoulder joint but with constraints on the elbow joint to reside in $q_2 \in [-\pi, \pi]$. Using the unwrapped space [27] for $\boldsymbol{q}$, we choose a finite number of goal states to reach, $\boldsymbol{q}_g = [\pi + 2\pi p, 0]^T$, $\dot{\boldsymbol{q}}_g = \mathbf{0}$ with $p = 0, \pm 1, -2$. For tree search parameters, we selected $D_{\max} = 2.0$ and $\gamma = 0.1$.

We implemented the UA1-RRT routine within C++. We compare our approach to two other methods, also implemented in C++. The first is an adaptation of the AVP-RRT algorithm [14] performing the same procedure in Algorithm 1 expect for the path profile step (Alg. 3 L5) being the AVP algorithm provided by the TOPP library [20]. The second method is a standard RRT routine with k nearest neighbours (*KNN-RRT*) [9]. As discussed earlier, standard state-based steering is only viable for fully or redundantly actuated systems as finding profiles $\boldsymbol{q}(t)$ for underactuated systems that satisfy the dynamics exactly is non-trivial. Due to this restriction, we instead adopt the control-based steering approach of [9] for our KNN-RRT implementation. Given UA1-RRT also uses a sample-based steering approach, we believe this choice enables a fair comparison.

All tests were run using a nearest neighbours value of $k = 10$ in $\textsc{Nearest}_k$ to avoid excessively long run times. For each method, we used the same random seed sequences for point selection in RANDOMCONFIGURATION. We performed 20 seeds for the UAV example and 10 for the acrobot. We set a maximum run time of 5000 seconds for each run. For the path-parameterised methods, the integration procedures in s used a resolution of $\Delta s = 10^{-3}$ and for KKN-RRT we used a time step $\Delta t = 10^{-2}s$. For each EXTEND procedure in UA1-RRT and KNN-RRT, we trialled $N_{rndm} = 200$ random paths/controls respectively, whereas in AVP-RRT we performed one extension similar to their original approach. We recorded the computation time for each STEER action within EXTEND, with average steering times $t_{\textsc{Steer}}$ recorded for each run of the examples.

VI. Results and Discussions

All tests were performed on an Intel i7-2600 3.40 GHz processor, with the statistics for the UAV and acrobot tabulated in Tables I and II respectively. It is clear that of the three methods, UA1-RRT achieves the lowest mean run time and highest success rates in computing feasible trajectories for both examples.

Evaluation of AVP-RRT's resulting trajectories using COMPUTEPROFILE (Alg. 3 L5) were found to be only partially feasible, with trees containing at most 54.8% feasible branches across all runs (Table II). Infeasible branches in these trees often admitted much higher values for θ in subsequent branches, leading to greater difficulty in reaching states of rest, typically reaching the maximum run time with very large iteration counts. Compared with the 100% branch feasibility from UA1-RRT and KNN-RRT, it appears that AVP-RRT in its current form is not well suited for UA1 systems despite accommodating such constraints [17].

Fig. 3 shows the percentage of successful attempts achieved in each example over time, with UA1-RRT achieving greater success than KNN-RRT in both Fig. 3a and 3b. This could be attributed to the path smoothness encouraged by UA1-RRT's steering, enabling the growth of steady and feasible motions towards the goal whereas KNN-RRT's more aggressive approach will often create branches from which motion to the goal can not be recovered, leading to the higher failure rate and greater computational time. Fig. 4 confirms this for the UAV, with the path-parameterised approaches generating visually smoother motions for the UAV in comparison to the sharper movements from the random-control steering of KNN-RRT, particularly in pitch angle.

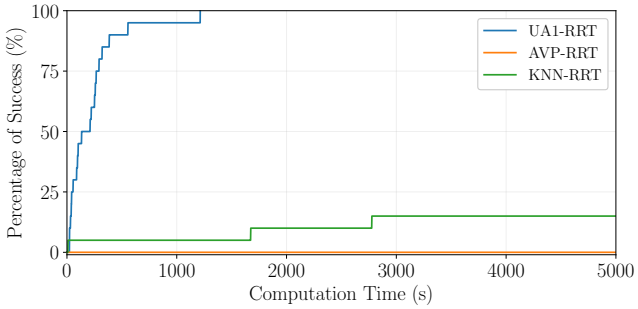
In the acrobot case (Fig. 3b), it is clear that KNN-RRT rapidly achieves 60% of program success before stalling, whereas UA1-RRT's rate of success grows more steadily and reaches 100% success. This difference also offers the insight that path smoothness could also impede tree growth if aggressive maneuvres are required of a system to continue tree expansion. Fig. 5 illustrates an example in which our

	Success (%)	Run Time (s)	Feasible Edges (%)	t_{STEER} (ms)	Iterations	Vertices
UA1-RRT	**100.0**	**229.9 / 1214.0**	**100.0 / 100.0**	1.00 / 1.29	11619 / 38571	10133 / 31619
AVP-RRT	0.0	3188.1 / 5000.0	10.5 / 16.8	1.1 / 1.18	72747 / 133692	22443 / 46262
KNN-RRT	15.0	4474.2 / 5000.0	**100.0 / 100.0**	**0.40 / 0.53**	102283 / 121742	122739 / 146090

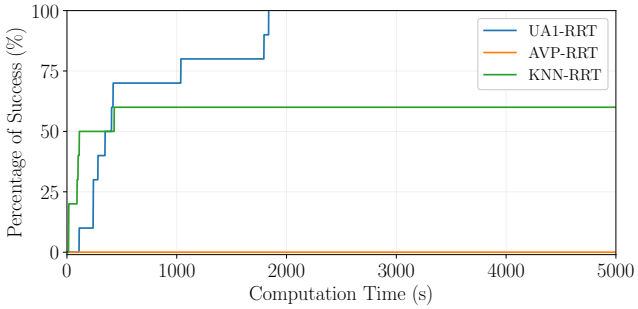
TABLE I: UAV fly-through results and statistics (mean / max).

	Success (%)	Run Time (s)	Feasible Edges (%)	t_{STEER} (ms)	Iterations	Vertices
UA1-RRT	**100.0**	**686.7 / 1886.7**	**100.0 / 100.0**	**0.10 / 0.15**	37282 / 69581	26933 / 50329
AVP-RRT	0.0	5000.0 / 5000.0	34.5 / 54.8	0.8 / 0.82	103991 / 291111	795 / 2305
KNN-RRT	60.0	2076.5 / 5000.0	**100.0 / 100.0**	2.0 / 2.3	29440 / 69351	52990 / 124831

TABLE II: Acrobot swing-up results and statistics (mean / max).



(a) UAV fly-through.



(b) Acrobot swing-up.

Fig. 3: Percentage of successful attempts vs. computation time for the UAV and acrobot examples.

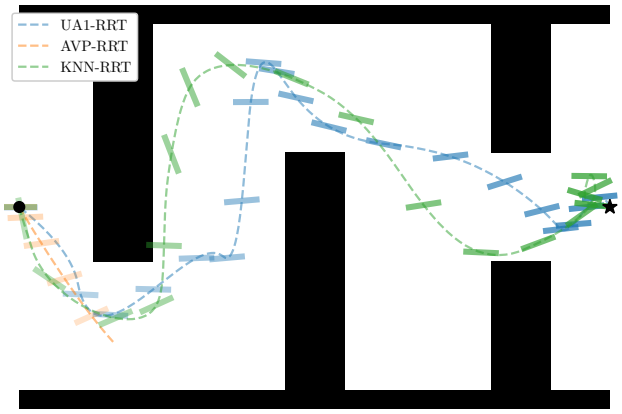


Fig. 4: UAV trajectories found by each method (feasible components of resulting trajectories shown).

smooth path construction leads to conservative motion, where we observe the requirement of several build-up swings before the final swing-up is achieved. Comparing this with KNN-RRT, more aggressive motions are generated by the random-control steering, where the upright state is achieved in shorter times through stronger torques being held for longer durations, as shown in Fig. 5. The ability of KNN-RRT to move much quicker through areas of its state space (e.g. the final second of motion in Fig. 5) may explain the shorter completion times to UA1-RRT in 3b, where the expansion of the tree through more varied motions can reach the goal in shorter times. To increase our approach's flexibility towards such erratic behaviour, $D_{\max}$ can be decreased appropriately, at the expense of denser and ultimately slower tree growth.

The rapid detection of infeasibility in PATHPROFILE allows UA1-RRT to discard paths with $s^\star < s^\dagger$ immediately in STEER (Alg. 3). As a result, little computation time is spent on paths created by GENERATEPATH that are not feasible. This is beneficial for systems with strong couplings between their paths and dynamics, as the majority of generated paths will be entirely infeasible and will not contribute to computational overhead. This is particularly the case for the acrobot, with steering times for UA1-RRT being the lowest by a considerable margin (Table II). AVP-RRT on the other hand performs several orders of magnitude slower for both examples, when considering UA1-RRT and KNN-RRT perform up to 200 trajectory evaluations per call of STEER whereas AVP-RRT only performs one. This is expected, however, since AVP must pre-compute several components such as limiting curves and switch points [20] to generate their velocity profiles, which UA1-RRT and KNN-RRT need not consider and instead compute theirs with a single forward pass.

VII. CONCLUSIONS

We have presented a sample-based motion planning algorithm using path parameterisation specialised to under-actuated degree-one systems. Using the structure these systems present under this parameterisation, an efficient state-based steering method was developed. For the examples considered in this paper, our proposed algorithm UA1-RRT demonstrated much higher rates of success and shorter mean

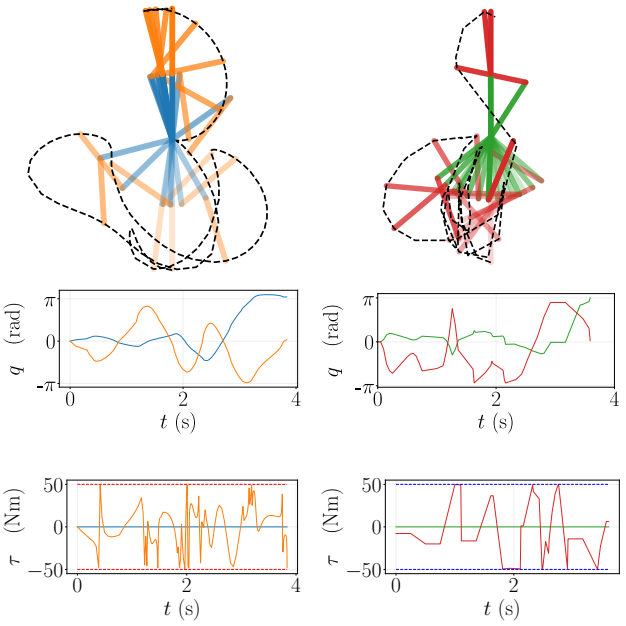


Fig. 5: Example acrobot motions for UA1-RRT (left) and KNN-RRT (right) with profiles $\boldsymbol{q}(t)$ and $\boldsymbol{\tau}(t)$.

run times in computing feasible trajectories in comparison to KNN-RRT and AVP-RRT, where these existing methods found difficulty. We highlighted that these achievements were largely attributed to our state-based steering approach, with efficient computations achieved without introducing excessive computational overhead.

Potential future avenues of research include an investigation into the scalability of UA1-RRT with models of higher dimension as well as those operating in more complex, obstacle-filled environments.

References

[1] S. M. LaValle, *Planning Algorithms*, 1st ed. Cambridge University Press, May 2006.

[2] V. L. Somers and I. R. Manchester, "Priority Maps for Surveillance and Intervention of Wildfires and other Spreading Processes," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 739–745.

[3] C. D. Bellicoso, M. Bjelonic, L. Wellhausen, K. Holtmann, F. Günther, M. Tranzatto, P. Fankhauser, and M. Hutter, "Advances in real-world applications for legged robots," *Journal of Field Robotics*, vol. 35, no. 8, pp. 1311–1326, 2018.

[4] A. Hereid, E. A. Cousineau, C. M. Hubicki, and A. D. Ames, "3D dynamic walking with underactuated humanoid robots: A direct collocation framework for optimizing hybrid zero dynamics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1447–1454.

[5] J. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, ser. Advances in Design and Control. Society for Industrial and Applied Mathematics, Jan. 2010.

[6] G. Tang, W. Sun, and K. Hauser, "Time-Optimal Trajectory Generation for Dynamic Vehicles: A Bilevel Optimization Approach," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China: IEEE, Nov. 2019, pp. 7644–7650.

[7] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A Fast Solver for Constrained Trajectory Optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China: IEEE, Nov. 2019, pp. 7674–7679.

[8] 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, Mar. 2016.

[9] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2. San Francisco, CA, USA: IEEE, 2000, pp. 995–1001.

[10] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation Planning with Probabilistic Roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, Aug. 2004.

[11] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[12] M. Kelly, "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, Jan. 2017.

[13] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[14] Q.-C. Pham, S. Caron, and Y. Nakamura, "Kinodynamic Planning in the Configuration Space via Admissible Velocity Propagation," in *Robotics: Science and Systems IX*. Robotics: Science and Systems Foundation, Jun. 2013.

[15] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *2012 IEEE International Conference on Robotics and Automation*. St Paul, MN, USA: IEEE, May 2012, pp. 2537–2542.

[16] S. Stoneman and R. Lampariello, "Embedding nonlinear optimization in RRT* for optimal kinodynamic planning," in *53rd IEEE Conference on Decision and Control*, Dec. 2014, pp. 3737–3744.

[17] Q.-C. Pham, S. Caron, P. Lertkultanon, and Y. Nakamura, "Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots," *The International Journal of Robotics Research*, vol. 36, no. 1, pp. 44–67, Jan. 2017.

[18] S. Caron, Q.-C. Pham, and Y. Nakamura, "Completeness of randomized kinodynamic planners with state-based steering," *Robotics and Autonomous Systems*, vol. 89, pp. 85–94, Mar. 2017.

[19] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-Optimal Control of Robotic Manipulators Along Specified Paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, Sep. 1985.

[20] Q.-C. Pham, "A General, Fast, and Robust Implementation of the Time-Optimal Path Parameterization Algorithm," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, Dec. 2014.

[21] H. Pham and Q.-C. Pham, "A New Approach to Time-Optimal Path Parameterization Based on Reachability Analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, Jun. 2018.

[22] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-Optimal Path Tracking for Robots: A Convex Optimization Approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, Oct. 2009.

[23] W. Suleiman, F. Kanehiro, E. Yoshida, J.-P. Laumond, and A. Monin, "Time Parameterization of Humanoid-Robot Paths," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 458–468, Jun. 2010.

[24] F. Bullo and K. Lynch, "Kinematic controllability for decoupled trajectory planning in underactuated mechanical systems," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 402–412, Aug. 2001.

[25] J. Grizzle, G. Abba, and F. Plestan, "Asymptotically stable walking for biped robots: analysis via systems with impulse effects," *IEEE Transactions on Automatic Control*, vol. 46, no. 1, pp. 51–64, Jan. 2001.

[26] I. R. Manchester and J. Umenberger, "Real-time planning with primitives for dynamic walking over uneven terrain," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 4639–4646.

[27] I. R. Manchester, J. Z. Tang, and J.-J. E. Slotine, "Unifying Robot Trajectory Tracking with Control Contraction Metrics," *Robotics Research: Volume 2*, pp. 403–418, 2018.