DIAG
Dipartimento di Ingegneria
informatica, automatica e gestionale
Antonio Ruberti

R3T: Optimal kinodynamic planning for nonlinear hybrid systems

Autonomous and Mobile Robotics
Master program in
Artificial Intelligence and Robotics

Professor:
Giuseppe Oriolo

Students:
Patrizio Perugini 1844358
Fabio Scaparro 1834913
Adriano Pacciarelli 1816493

Academic Year 2022/2023

Contents

Abstract

In this project we present the application of **optimal kinodynamic planning** algorithms for nonlinear hybrid systems subject to kinodynamic constraints. The analysis will be also validated on a pendulum. Sampling-based motion planning algorithms such as PRM and RRT are commonly used to solve planning problems due to their ability to efficiently find solutions. Nevertheless, when applied to kinodynamic and hybrid systems, they typically perform poorly and they do not guarantee probabilistic completeness. Consequently, we implemented the **R3T** algorithm [1], a probabilistic complete and asymptotically optimal variant of RRT for kinodynamic planning of nonlinear hybrid systems, and compared it with **RG-RRT** [2] and **RRT** [3] considering different systems, such as the *pendulum*, the *1D hopper* and the *2D hopper* (in an obstacle-free environment).

1 Introduction

Sampling-based motion planning algorithms, including probabilistic roadmaps (PRMs) [4] and rapidly-exploring random trees (RRTs) [3, 2, 1], have demonstrated their effectiveness in a wide range of planning problems. Despite their proven power, these methods have certain limitations such as:

- **Non-Optimality**: In the standard version these algorithms find a suitable motion regardless of the optimal solution.
- **Dependence on Sampling Strategy**: since performances heavily depends on the sampling strategy sometimes more elaborated sampling strategies might be needed (we have encountered this problem in one of our experiments).
- **Sensitivity to Obstacle Density**: in our experiments we didn't have obstacles but still the number of iterations needed to find a feasible motion might increase drastically in these scenarios.
- **Lack of Robustness**: the main disadvantage in this case is that if the initial conditions slightly change then a new plan needs to be found from scratch (this is indeed a *single query* algorithm).

RRT algorithms are based on the exploration of the state space and the connection of new states to previously explored states. When kinodynamic constraints are present, a two point boundary value problem must be solved to find an admissible trajectory for the connection, which can be expensive, because it may not be admissible to connect two states using a straight line (the trajectory must obey the system's dynamics). For general systems, the most common approach involves simulating trajectories forward and expanding the explored states with the nearest produced point to the sample state. However, this method may not be probabilistically complete in kinodynamic settings.

Additionally, those RRT approaches often perform poorly in hybrid systems and the choice of extension strategy and distance metrics can exacerbate these difficulties [2].

Numerous variations of the RRT algorithm have been created to enhance planning efficiency in kinodynamic and hybrid systems. In such systems, the nearest state in the tree to a sample may not be connected to the nearest reachable state to the sample (Fig. 1). Reachability-guided approaches, like the RG-RRT algorithm, tackle this issue by exploring reachable states ahead of time and then expanding the tree towards the nearest reachable state to the sample. However, this method does not define an universal method to extract reachable sets.

R3T provides a general framework to represent the reachable set of a state as (the union of) polytope(s) using the linearized local dynamics. The algorithm is probabilistically complete in kinodynamic settings with approximated reachable sets. R3T*, which is the version with rewiring, retains the asymptotic optimality of RRT*.

R3T provides the means for a general solution to the problem, which is guided by polytopic reachable sets. In this report we'll start by analyzing the standard RRT. We'll then focus on the novelty of RG-RRT and finally extend to the more general R3T .

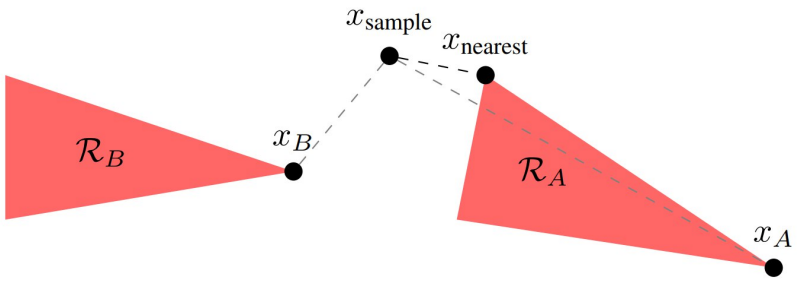

Figure 1: RRT extension and system reachability. While x_B is closer to x_{sample} than x_A , the reachable set of x_A , R_A , has states closer to x_{sample} than any point in R_B . A reachability guided algorithm would extend the tree to $x_{nearest}$, whereas traditional RRT would either extend in R_B or fail

2 Background

2.1 Kinodynamic motion planning

Kinodynamic motion planning focuses on generating motion plans for robots that take into account both their kinematics (i.e., the study of motion without regard to the forces and torques that cause the motion) and their dynamics (i.e., the study of motion with regard to the forces and torques that cause the motion).

We formulate the kinodynamic planning problem as motion planning in a 2n-dimensional state space [2] that has first-order nonholonomic constraints. Let $\mathcal{C}$ denote the configuration space ($\mathcal{C}$-space) that arises from a rigid or articulated robot that moves in a 2D or 3D world. Let χ denote the state space, where $x \in \chi$ is defined as $x = (q, \dot{q})$ for $q \in \mathcal{C}$.

In kinodynamic motion planning, the goal is to plan a motion that not only takes the robot to its desired destination, but also ensures that the robot moves smoothly and safely along the motion, without violating any physical constraints such as maximum velocity or acceleration limits. The resulting motion is often referred to as a *feasible trajectory* as it represents a motion that the robot can actually follow in the real world.

Specifically, we consider the general problem of motion planning for a system subject to differential constraints. We are interested in finding a control input $u(t) \in U$ that drives the system from the initial state x_{init} to the final state x_{goal} in a finite time T.

The resulting trajectory $x(t)$ must be collision free and satisfy the differential constraints according to the system dynamics:

$$\dot{x} = f(x, u) \tag{1}$$

This can make the problem computationally expensive, especially for complex robots and environments.

In order to describe the algorithms that solve this problem, the definition of *reachable set* is needed.

For a generic state $x_0 \in \chi$ we define the **reachable set** $\mathcal{R}_{\Delta t}(x_0)$ to be the set of all states that can be reached, starting from x_0 according to the system dynamics (Formula 1) through the set of all available control inputs in less than a finite time interval Δt. Overall, kinodynamic motion planning is a critical component of many robotic applications, and ongoing research in this field is aimed at improving the efficiency, safety, and robustness of the planning algorithms for a variety of robot platforms and environments.

2.2 Hybrid systems

A hybrid system is a type of mathematical model that describes the behavior of a system that exhibits a set of both continuous and discrete dynamics.

The continuous dynamics describe the system's behavior in the absence of any discrete events, while the discrete dynamics describe the system's behavior when a discrete event occurs. The discrete events can include actions such as mode switches which are changes in the differential equations that describe the continuous dynamic, instantaneous state transitions (also called *jumps*), or other changes in the system's behavior.

2.3 Polytopes

Polytopes are mathematical objects which are commonly used in mathematical programming and that are well suited to approximate reachable sets. We will follow the definitions of [5] and [6].An *H-polytope* in $\mathbb{R}^n$ is a **bounded** set in this form: $\mathbb{H} = \{x \in \mathbb{R}^n | Hx \leq h\}$, where $H \in \mathbb{R}^{q \times n}$ is full column rank. This object denotes a region of space bounded by hyperplanes, and it has a dual representation called *V-polytope*, which instead highlights the vertexes of this region. We will focus on the former in this report. An *H-polytope* can be transformed through an affine transform into what is called an *AH-polytope* which has the general form: $\mathbb{A} = \bar{x} + G\mathbb{H}$. Note that in general $\mathbb{A}$ and $\mathbb{H}$ may live in euclidean spaces of different dimension. An *AH-polytope* admits a representation using hyperplanes but it is not efficient to compute it, and we will use the affine transformation form for the rest of the report.

One problem of interest that often comes up when dealing with *AH-polytopes* and in [1] is finding the distance between an *AH-polytope* P and a query point q. The way it is solved in [6] is formulating this mathematical problem:

$$\begin{aligned} d(q, P) = \min \quad & \|\delta\|_p \\ \text{s.t.} \quad & \bar{x} + Gx = q + \delta \\ & Hx \leq h \end{aligned} \tag{2}$$

Introducing the slack variables δ and minimizing their norm according to a given norm such as $L2$ or $L1$. Once you solve this problem you can also find the projection of the query on the polytope $x_c = \bar{x} + Gx^*$, where x^* is the value of x at optimum.

3 Planning algorithms

3.1 RRT

RRT (Rapidly-exploring Random Tree) [3] is a probabilistic algorithm commonly used in motion planning to generate a collision-free trajectory for a robot or other objects in a complex, high-dimensional environment. The algorithm works by incrementally constructing a tree that connects the initial state to the goal state. Specifically, at each iteration, the algorithm randomly samples a state and finds the nearest node in the tree to that configuration (according to a given metric). Once the closest node is found the tree grows towards the random configuration by a specified step size and checks if the resulting configuration collides with any obstacle. If it does not collide, the new configuration is added to the tree and connected to its closest neighbor. This process is repeated until the goal configuration is reached. In the standard formulation, this algorithm tends to expand nodes which have a large associated **Voronoi region**. The Voronoi region associated to a node is the subset of χ such that every point in it is closer to that node than any other node in the tree.

The RRT algorithm is particularly useful for complex, high-dimensional environments because it rapidly explores the space and can quickly generate feasible motions. However, the motion generated by RRT may not be the most optimal and other algorithms may be better suited for finding optimal motions.
In addition, the step size used in RRT plays an important role in balancing exploration and efficiency. If the step size is too large, the algorithm may overlook important regions of the space and produce sub optimal motions. On the other hand, if the step size is too small, may take longer to converge.

The uniform sampling of the state space is the general approach, however more sophisticated sampling strategies can be adopted in order to allow the algorithm to converge faster. One could, for instance, use a goal biased sampling strategy to shape the Voronoi bias expanding toward the goal.

One shortcoming of RRT is that its performance degrades in presence of complex kinodynamic constraints, as highlighted in [2], [1]. This is caused by the fact that the expansion of a node might not get the state closer to the random sample. As a matter of fact a straight line is often an invalid trajectory for the system. To solve this problem, the node is expanded simulating the system for a time interval equal to the step size with the *best known* input that steers the system as close as possible to the random sample. Finding this input requires in general to solve a two point boundary value problem, which is hard and computationally unfeasible (it has to be done for every expansion). While it is possible to find solutions to this using optimal control [7] or **motion primitives**, it is not guaranteed that the new state will actually be closer to the sample than its parent. This results in choosing many times the same nodes for expansions without actually growing the tree in unexplored regions, considerably

slowing down convergence.

To improve the performance of this algorithm on complex systems, one approach is to find a better distance metric that takes into account the dynamics of the system, but it turns out that finding a good metric is often just as hard as solving the original planning problem. In this report we will instead focus on approaches that look at *reachable sets* of states to guide the expansion of the tree. The variants we will look at are RG-RRT and R3T.

Algorithm 1 RRT

Require: $(x_{init}, x_{goal}, iterations)$
  $\mathcal{T} \leftarrow InitializeTree()$
  $\mathcal{T} \leftarrow InsertNode(x_{init}, \mathcal{T})$
  $N \leftarrow 0$
  while $N \neq iterations$ **do**
    $x_{rand} \leftarrow RandomState()$
    $(x_{near}) \leftarrow NearestState(x_{rand}, \mathcal{T})$
    $u \leftarrow SolveInput(x_{near}, x_{rand})$
    $x_{new} \leftarrow NewState(x_{near}, u)$
    $\mathcal{T} \leftarrow InsertNode(x_{new}, \mathcal{T})$
    if $GoalCheck(x_{new}, x_{goal})$ **then**
      Return $\mathcal{T}$
    end if
    $N \leftarrow N + 1$
  end while

3.2 RG-RRT

Reachability Guided RRT is a motion planning algorithm that builds upon the basic Rapidly-exploring Random Tree (RRT). It takes the form of a modified RRT that explicitly accounts for the limitations of the system dynamics to shape the Voronoi bias so as to emphasize nodes within the tree that exhibit the greatest contribution towards exploring the state space. Moreover any node is added to the tree only if it makes progresses toward a given sample. RG-RRT provides a means for choosing nodes that have a better chance of yielding consistent trajectories without having to redefine the metric function.
This is achieved by using a simple heuristic that quickly throws away random samples that otherwise would not have the effect of extending the tree into previously unexplored regions of the state space.

In order to further develop the algorithm we must simplify the definition of reachable set.

In our specific implementation we have decided to represent the reachable set of a state as an approximation of the true reachable set using some *control points* . Specifically, each of these control point is obtained by applying a different control (motion primitives) for a given time and from a given state. With this choice all these points are indeed reachable by construction. For instance the resulting reachable set of the pendulum rather then being an approximately linear segment is identified by endpoints of this segment which are defined by the points in state space that are achieved by applying the minimum and maximum torques at the shoulder (Fig. 2).

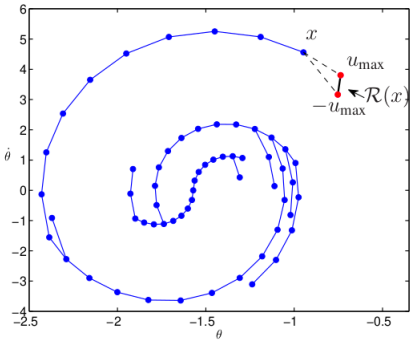

Figure 2: The reachable set $\mathcal{R}(x)$ for the underactuated pendulum.

Notice how with this assumption we have a much easier problem to solve since we have a small number of possible next states from each configuration. This choice will lead to fast runs of the algorithm.

Regarding the implementation of the algorithm, we followed the pseudo-code proposed in [2] (Alg. 2).

Until now we have only described the approximation of the reachable set, the great innovation of RG-RRT is in the insertion of a new node in the tree. With the node and its corresponding reachable set added to the tree, we draw a random sample x_{rand} from the state space, and use it to grow the tree. We do so through a variation on the nearest-neighbor matching strategy, that restricts its domain to nodes that are more likely to promote the expansion of the state space. Specifically, if the closest Reachable point is closer to the sample than the closest node of the tree, then x^r_{near}

Algorithm 2 RG-RRT

Require: $(x_{init}, x_{goal}, iterations)$
  $\mathcal{T} \leftarrow InitializeTree()$
  $\mathcal{T} \leftarrow InsertNode(x_{init}, \mathcal{T})$
  $N \leftarrow 0$
  while $N \neq iterations$ **do**
    $x_{rand} \leftarrow RandomState()$
    $(x_{near}, x^r_{near}) \leftarrow NearestState(x_{rand}, \mathcal{T})$
    if $Dist(x_{rand}, x^r_{near}) \leq Dist(x_{rand}, x_{near})$ **then**
      $u \leftarrow SolveInput(x_{near}, x^r_{near})$
      $x_{new} \leftarrow NewState(x_{near}, u)$
      $\mathcal{T} \leftarrow InsertNode(x_{new}, \mathcal{T})$
      if $GoalCheck(x_{new}, x_{goal})$ **then**
        Return $\mathcal{T}$
      end if
      $N \leftarrow N + 1$
    end if
  end while

will be added to the tree. Otherwise, if the closest node of the tree is nearer to the sample than any Reachable point no expansion is done and a new sample is taken. The result of this policy of throwing out samples for which the nearest node is closer than its reachable set is a change in the Voronoi bias. In the pseudo-code the function $Dist(\cdot, \cdot)$ can be arbitrarily modified to take into account any metric. The RG-RRT then allows samples only from Voronoi regions for which the differential constraints permit the expansion of the node towards the sample. While samples may be drawn from anywhere within the state space, only a subset of regions are actually used to grow the tree. This has the effect of modifying the Voronoi bias to emphasize nodes that are better suited to explore the state space while growing the tree.

3.3 R3T

R3T is a sampling based algorithm that builds on RG-RRT and addresses the problem of generalizing the representation of reachable sets. In particular it develops polytope reachable sets as an approximation of the true reachable set.

An *AH-polytope* (2.3) is well suited to approximate reachable set of non-linear systems. Consider the general expression of a hybrid dynamic system:

$$\begin{aligned} \dot{x} &= f(x, u, \sigma), (x, u) \notin \mathbb{G} \\ (x, \sigma)^+ &= r(x, u, \sigma), (x, u) \in \mathbb{G} \end{aligned} \tag{3}$$

Where σ is the system mode and $\mathbb{G}$ denotes the set where the system has discrete jumps or changes its dynamic mode. It can be discretized to obtain the following discrete dynamics:

$$x^+ = F_i(x,u), (x,u) \in \mathbb{S}_i \tag{4}$$

Where $\mathbb{S}_i = \{x,u|S_i(x,u) \leq \mathbf{0}\}$ is the set in which the system evolves according to the discretized dynamics $F_i(\cdot)$ and it also embeds state and input constraints. In the case of continuous time dynamics, $F_i(x,u)$ can be obtained via forward Euler method: $F_i(x,u) = x + dt \cdot f(x,u,\sigma_i)$. This dynamic can be linearized together with the set of constraints at $(x,u) = (\bar{x},\bar{u})$ to obtain the affine dynamics (5)

$$x^+ = A_i x + B_i u + c_i, \quad D_i x + E_i u \leq \zeta_i \tag{5}$$

where:

$A_i = \frac{\partial F_i}{\partial x}|^{x=\bar{x},}_{u=\bar{u}}$, $B_i = \frac{\partial F_i}{\partial u}|^{x=\bar{x},}_{u=\bar{u}}$, $c_i = F_i(\bar{x},\bar{u}) - A_i\bar{x} - B_i\bar{u}$,
$D_i = \frac{\partial S_i}{\partial x}|^{x=\bar{x},}_{u=\bar{u}}$, $E_i = \frac{\partial S_i}{\partial u}|^{x=\bar{x},}_{u=\bar{u}}$, $\zeta_i = S_i(\bar{x},\bar{u}) - D_i\bar{x} - E_i\bar{u}$

The reachable set in discrete time is then approximated by:

$$\mathcal{R}_{DT}(\bar{x}) = A_i\bar{x} + c_i + B\{u|E_i u \leq \zeta_i - D_i\bar{x}\} \tag{6}$$

which is an *AH-polytope*. This polytope approximates the set of all states reachable after a time interval *equal* to the planner's step size applying admissible inputs. This formulation is extremely general and holds for many kinds of systems subject to kinodynamic constraints and with hybrid dynamics.
To obtain an approximation for the states reachable in a time which is *less or equal* than the planner's step size, $\mathcal{R}_{CT}(\bar{x})$, it is possible to take the **convex hull** of $\bar{x}$ and $\mathcal{R}_{DT}(\bar{x})$, which is still an *AH-polytope* [5]. This allows to choose a large step size without overlooking reachable states.

This approximation is valid in systems which do not exhibit high discontinuities in state and for small integration time steps. For hybrid systems, the authors in [1] propose to linearize the system in each attainable mode from $\bar{x}$ and approximate the reachable set with the union of the corresponding polytopes. Having developed this framework, the actual **R3T** algorithm is a natural evolution of RG-RRT. Regarding the implementation of the algorithm, we followed the pseudo-code proposed in [1] (Alg. 3).

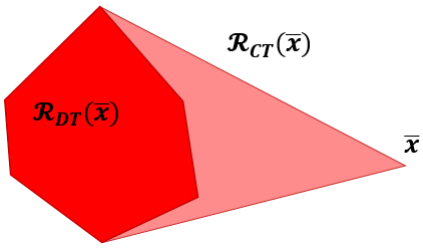

Figure 3: Reachable set in the continuous system

Every node in the tree stores its reachable set in a data structure called *Polytope Tree*. A random sample x_s is drawn from the phase space according to a sampling distribution. Then **R3T** finds the polytope which is closest to x_s in the tree and

Algorithm 3 R3T

Require: $(\mathcal{R}, x_{start}, x_{goal}, iterations, optimality)$
  $\mathcal{T} \leftarrow InitializeTree()$
  $\mathcal{T} \leftarrow InsertNode(x_{init}, \mathcal{R}(x_{init}), \mathcal{T})$
  $N \leftarrow 0$
  while $N \neq iterations$ **do**
    $x_s \leftarrow RandomState()$
    $(x_c, \mathcal{R}_c) \leftarrow FindNearest(x_s, \mathcal{T})$
    $x_{new} \leftarrow Extend(R_c, x_c, x_s)$
    if $CollisionFree(x_{near}, x_{new})$ **then**
      $\mathcal{T} \leftarrow InsertNode(x_{new}, \mathcal{R}(x_{new}), \mathcal{T})$
      if $optimality == True$ **then**
        $Rewire(\mathcal{T}, \mathcal{R}(x_{new}))$
      end if
      if $GoalCheck(x_{new}, x_{goal})$ **then**
        Return $\mathcal{T}$
      end if
      $N \leftarrow N + 1$
    end if
  end while

consequently it computes the projection of the random sample on this polytope x_c, which coincides with x_s if it is already in the polytope. A detailed explanation of $FindNearest(\cdot, \cdot)$, which is the function performing this operations, is given in the section 5.1. Then the node x_{near} corresponding to the closest reachable set is expanded toward x_c by the $Extend(\cdot, \cdot)$ routine, which is described in Alg 4. It is the most delicate part as it is not guaranteed that x_c is actually reachable, as the reachable set is just an approximation. To connect the two states, a suitable input must be found defining $CalcInput(\cdot, \cdot)$. In [1], the authors propose to invert the linearized model to find the best input (Formula 7).

$$u_{pinv} = \texttt{pinv}(B_i)(x_c - A_i x_{near} - c_i) \tag{7}$$

This particular choice might expand x_{near} toward points which are very far from x_c due to the linearization and possible mode switches in the neighborhood of x_{near}. After simulating the trajectory for a predetermined step size, the resulting state is added to the tree as a new node and its reachable set is computed and inserted in the *Polytope Tree*.

Finally the rewiring procedure assures asymptotic optimality. *Rewire* consists of finding the best parent of a newly added node, and using the new node as a potential parent. When a new node is added we look for potentially better parents for which

Algorithm 4 Extend

Require: (R_c, x_c, x_s)

Require: τ ▷ Planner step size

$u \leftarrow CalcInput(R_c, x_s)$

$x_n \leftarrow \int_{t=0,x(0)=x_s}^{t=\tau} f(x,u)\,dt$

if $CollisionFree(x_s, x_n)$ **then**

Return x_n

end if

Return $\emptyset$

the newly added node is within their reachable set and the parent with the lowest cost to go is selected. Once the node has been added with the proper parent, we look for nodes which are within the reachable set of this newly added node and we compare the new node against the original parent of the nodes that are found in its reachable set, selecting as a parent the one with the smallest cost to go.
Implementation details on rewiring will follow in section 5.4.

4 Models

The previously mention planning algorithm have been tested on common systems, both hybrid and non, in order to empirically determine their effectiveness. The robotic system taken into consideration are the *actuated Pendulum*, the *Hopper 1D* and the *Hopper 2D*.

4.1 Pendulum

The torque limited pendulum is a simple yet effective system often used to test algorithm and control laws. It is characterized by a link whose angular acceleration is actuated trough the use of a revolute joint.

To model this system, we have considered a single-link torque-limited system with damping, characterized by a 2-dimensional configuration state $(\theta, \dot{\theta})$, which refers to the current position and velocity of the system, and one input torque (τ). The dynamic model (Formula 8) have been proposed in the code associated to [1] as single link pendulum with mass at the end of a rod.

$$I\ \ddot{\theta} = \tau + \tau' + \tau_{damp} \tag{8}$$

where:

$$\tau_{damp} = -b\ \dot{\theta}$$

$$\tau' = -(mgl\sin(\theta) + m_l g\frac{l}{2}\sin(\theta))$$

$$I = ml^2 + (m_l l^2)\frac{1}{12}$$

with τ input torque, l length of the rod, m mass at the end of the pendulum, m_l mass of the rod and I inertia and b damping factor.

4.2 Hopper 1D

This is a one-dimensional hybrid system with a single DoF in the prismatic joint at the end of the "leg".

The model (the same as [1]) is characterized by a 2-dimensional configuration state comprehensive *current position* (x) and *velocity* $(\dot{x})$ of the system, and one input (f). It is a hybrid system, it presents continuous and discrete modes. Precisely, two **continuous dynamic modes** (*free flight* and *soft ground contact*) and one **discrete dynamic mode** (*inelastic collision* with the ground), which are interchanged depending on the current position of the robot.

$$\begin{cases} \ddot{x} = -g & \text{for } x > l + p \ \ (\textit{free flight}) \\ \ddot{x} = \frac{f}{m} - g & \text{for } l < x \leq l + p \ \ (\textit{ground contact}) \\ \dot{x}_{k+1} = -b\, \dot{x}_k & \text{otherwise} \ \ (\textit{inelastic collision}) \end{cases} \tag{9}$$

where f is the input to the joint, m is the hopper mass, l is the hopper length, p is the piston length, b is the damping factor for the ground and g is the gravity constant. To be noticed that the only moment in which an input command can be given is during the contact phase, while in the other modes the system can only passively follow the its natural dynamics.

4.3 Hopper 2D

The Hopper 2D (Fig. 4) is the last and more complex model we have considered, and only R3T has been able to consistently find a solution. It is a two-dimensional system composed by two parts (*body* and *leg*) and two actuators, one *revolute* located at the *hip*, which is the connection point of the body and the leg, and the other *prismatic* located in the leg.

The model [8] can be described with 5 generalized coordinates ($\in \mathbb{R}^5$), comprehensive of *2D Cartesian position* (x, y), *leg* and *body angle* (θ, ϕ) wrt the vertical axis and *leg extension* (r). Consequently, the dynamic model will consider a **10-dimensional state vector** ($\mathbf{x}$), comprehensive of the 5 generalized coordinates and their time derivatives ($\dot{x}$, $\dot{y}$, $\dot{\theta}$, $\dot{\phi}$, $\dot{r}$), and a **2-dimensional input vector** (τ) comprehensive of the revolute joint command (τ_{hip}) and the prismatic joint command (F_{leg}). In addition, like the 1D hopper, it is an hybrid system which presents four continuous dynamic modes(*contact ascent*, *contact descent*, *flying ascent* and *flying descent*) [8].

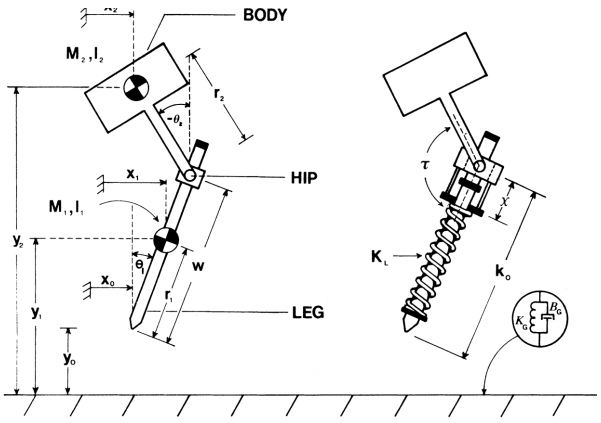

Figure 4: Visual illustration of a Hopper 2D model

The final dynamic model [8], is described by the following five differential equation:

$$
\begin{aligned}
(J_l - m_l R l_1)\,\ddot{\theta}\cos\theta - m_l R\ddot{x}_{ft} &= \cos\theta\left(l_1 F_y \sin\theta - l_1 F_x \cos\theta - \tau_{hip}\right) \\
&\quad - R\left(F_x - F_{leg}\sin\theta - m_l l_1 \dot{\theta}^2 \sin\theta\right) \\
(J_l - m_l R l_1)\,\ddot{\theta}\sin\theta + m_l R\ddot{y}_{ft} &= \sin\theta\left(l_1 F_y \sin\theta - l_1 F_x \cos\theta - \tau_{hip}\right) \\
&\quad + R\left(m_l l_1 \dot{\theta}^2 \cos\theta + F_y - F_{leg}\cos\theta - m_l\right) \\
(J_l + mRr)\,\ddot{\theta}\cos\theta + mR\ddot{x}_{ft} + mR\ddot{r}\sin\theta + mRl_2\ddot{\phi}\cos\phi &= \cos\theta\left(l_1 F_y \sin\theta - l_1 F_x \cos\theta - \tau_{hip}\right) \\
&\quad + RF_{leg}\sin\theta + mR\left(r\dot{\theta}^2 \sin\theta + l_2\dot{\phi}^2 \sin\phi - 2\dot{r}\dot{\theta}\cos\theta\right) \\
(J_l + mRr)\,\ddot{\theta}\sin\theta - mR\ddot{y}_{ft} - mR\ddot{r}\cos\theta + mRl_2\ddot{\phi}\sin\phi &= \sin\theta\left(l_1 F_y \sin\theta - l_1 F_x \cos\theta - \tau_{hip}\right) \\
&\quad - R\left(F_{leg}\cos\theta - mg\right) - mR\left(2\dot{r}\dot{\theta}\sin\theta + r\dot{\theta}^2\cos\theta + l_2\dot{\phi}^2\cos\phi\right) \\
J_l l_2 \ddot{\theta}\cos(\theta-\phi) - JR\ddot{\phi} &= l_2\cos(\theta-\phi)\left(l_1 F_y \sin\theta - l_1 F_x\cos\theta - \tau_{hip}\right) \\
&\quad - R\left(l_2 F_{leg}\sin(\phi-\theta) + \tau_{hip}\right)
\end{aligned}
$$

where $F_x(10)$ and F_y (11) are the reaction forces felt exclusively during contact (when the y-coordinate of the foot in negative=.

$$
F_x = \begin{cases} k_g\left(x_{ft} - x_{td}\right) - b_g\dot{x}_{ft} & \text{for } y_{ft} < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{10}
$$

$$
F_y = \begin{cases} k_g y_{ft} - b_g\dot{y}_{ft} & \text{for } y_{ft} < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{11}
$$

with k_g and b_g elastic constant and damping relative to the hopper's leg (and the ground), m is the hopper mass, l_1 is the distance from the foot to the leg's center of mass, l_2 is the distance from the hip to the body's center of mass, m_l is the leg mass, J is the body inertia and J_l is the leg inertia.

5 Implementation details

All the algorithms have been implemented in Python. The main libraries we used are: numpy, simPy, pypolycontain, osqp, rtree. SimPy has been used to define the model of the hopper 2D symbolically. To ease computation we also used lambdify, which is a module that provides convenient functions to transform symPy expressions to lambda functions. This reduces the expensive symbolic substitution to a numpy expression evaluation which is much faster.

In order to represent polytopes we used pypolycontain and osqp has been used to solve the minimization problem defined in Formula 2. Rtree has been used for fast nearest neighbor search and intersection search of bounding boxes.

In the following subsection we will give some insights on specific implementation choices that allowed us to get to the final results.

5.1 Find the closest polytope

The most important operation of R3T is $FindNearest$, that is finding the closest reachable polytope in the tree given a query point. The naive implementation is to brute search for the polytope whose distance to the query is minimal, computing the point-polytope distance defined in the Background section. This is clearly not feasible, as $FindNearest$ is called at every expansion and the number of polytopes grows linearly with the nodes. Instead we have implemented the `AABB-query` algorithm in [6]. To explain this algorithm we will introduce the concept of *axis aligned bounding box* (**AABB**).

An axis aligned bounding box is a bounded set described by inequalities in the form:

$$B(l, u) = \{x \in \mathbb{R}^n | x \geq l, x \leq u\}$$

Note that an AABB is also an *H-Polytope*. One important operation which involves polytopes and bounding boxes is finding the smallest bounding box which contains a given *AH-polytope*. This requires solving $2n$ linear problems where n is the dimension of the state:

$$l_i = \min_{Hx \leq h} g_i + [Gx]_i \quad u_i = \max_{Hx \leq h} g_i + [Gx]_i, \quad i = 1, ..., n \tag{12}$$

Where the suffix i denotes the i-th component of the vector. The reason why it might be useful to construct bounding boxes for polytopes is that AABBs can be indexed efficiently in a data structure called **R-tree**. Similarly to a KD-tree, it allows to efficiently compute nearest neighbor operations, but is splits the space in a totally different way: while a KD-tree partitions the space using hyperplanes, an R-tree divides the space in multidimensional rectangles (hence the R) which can overlap with each other. Moreover it allows the insertion of AABBs and finding the intersection

of AABBs, which is what enables to find the nearest polytope among a small set of candidates.

The `AABB-query` algorithm works by maintaining a *Polytope tree*, which is composed of two trees: one tree contains *keypoints* and can be for example a KD-tree, and a tree which contains *bounding boxes* and it is an R-tree. A keypoint associated to a polytope is such that it belongs to the polytope, while a bounding box associated to a polytope is just the one obtained solving (Formula 12). Inserting a new polytope in the tree is straightforward: the associated keypoint and bounding box are generated and put in the respective trees with an identifier which points to the polytope. Finding the closest polytope to a given query point is more complex: first, we find the closest keypoint to the query and consequently the associated polytope. Then, the distance from the query to this polytope d is computed. Since the distance to the keypoint upper bounds the distance to the polytope, it might be necessary to check other polytopes in an area surrounding the query point. In order to do this, an *heuristic box* is built $B_h(l = q - |\delta^*|, u = q + |\delta^*|)$, where $|\delta^*|$ are the absolute values of the slack variables at optimum in (Formula 2). This box is then used to query the AABB tree to find all the other intersecting boxes. For every intersecting box, we fetch the associated polytope and compute the distance to the query, if it is smaller than the current best alternative we update the minimum distance and we construct another heuristic box, repeating the process until there's no more bounding boxes to try, at which point we return the polytope associated to the minimum distance. This way we only solve the expensive problem (Formula 2) for a subset of polytopes which does not necessarily grow in size as the number of nodes.

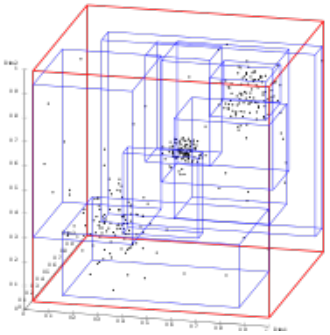

Figure 5: 3D R-tree graphical representation.

5.2 Fast-forward expansion

This is an expansion technique that is discussed in the optimization section of [1] and it speeds up convergence for hybrid systems which have modes with no input. This can happen, for example, in systems which can only apply inputs while they are in contact with the ground, like hoppers. While they're flying, their trajectory is decided at the time of takeoff. With this option enabled, the algorithms does not add nodes when the expansion is deterministic (no inputs). Instead, the dynamics is simulated until inputs are applicable again and the whole trajectory is appended to the node. This prevents the planner from expanding those states which always generate the same child resulting in a large amount of dropped samples (nodes which are already in the tree are discarded). This simple technique allows planners to increase their performance in

the benchmark tasks of hopper 1d and hopper 2d.

5.3 Sampling techniques

Uniformly sampling the space is the standard way to quickly explore the states in sampling based algorithms by leveraging the Voronoi bias. However, without adding any sort of bias toward the goal, convergence can be painfully slow. To solve this problem we implemented a simple modified sampling strategy for the pendulum benchmark task: with some probability $p = 0.2$ we use the goal state as a sample, and with probability $1 - p$ we fall back to uniform sampling. This adds a small bias to expand in the goal region and allows to get results in a reasonable time. It has been applied for all planning algorithms evaluated.

In addition, regarding only the Hopper 2D, we followed the implementation of the authors for a more advanced sampling strategy, since the standard uniform sampling was not able to converge (in a reasonable time) to a final solution. Our best guess why a standard uniform strategy was not effective is that 2D hoppers have a large state space and narrow reachable polytopes. As a result, the majority of the random samples were not able to make the system add any new promising nodes, excessively increasing the execution time. The solution to this problem has been to implement a *hip coordinates sampler*, which samples random configurations of the robot's hip and then converts them to hopper foot coordinates, allowing the system to converge.

5.4 Rewiring

Rewiring is the operation that allows to modify the structure of the tree and make it cost optimal. The choice of the cost function is up to the implementation, we chose to minimize the control effort along the trajectory. The implementation in [1] sets as the cost function the L_2 norm between subsequent nodes, which might result in shorter trajectories at the cost of more control effort. We believe that in constrained systems the Euclidean norm is not a good indication of cost to go, as is clearly highlighted in [2] in the case of a pendulum, and that the control effort is instead a more general objective to minimize regardless of the dynamics. Moreover, we approached the rewiring of the connections with special care: the new connections must still follow admissible motions, meaning that we need to solve for the input that connects the two nodes. In [1] this is done through a variant of $Extend(R_c, x_c, x_s)$, which also checks that x_n is approximately the same as x_c before connecting the two nodes. In practice we found out that in most of our simulations, finding inputs with Formula 7, almost never correctly connected the two states within acceptable tolerance.

6 Simulations

In this section we analyze the results that we have achieved applying the planning algorithms (when possible) to the previously mentioned systems. In the next chapters we present for each model the results obtained using the different algorithms, analyzing random runs (especially regarding the expanded nodes and the quality of the final solution) and comparing all results for 10 different runs.

6.1 Pendulum

Task: The task is the classic "*swing up*", starting at rest ($\dot{x}_{start} = 0\ [rad/s]$) in a configuration ($x_{start} = 0\ [rad]$) and aiming to the resting state ($\dot{x}_{goal} = 0\ [rad/s]$) at $x_{goal} = \pi\ [rad]$. The idea is that, due to its limited torque, the system is not able to directly accomplish the task. Consequently, the resulting behaviour is to "pump" energy into the system in order to be able to reach a state which requires more potential energy than the starting configuration.

Simulation parameters: we chose to model the pendulum with $m = 1\ [Kg]$, $m_l = 0\ [Kg]$, $l = 0.5\ [m], b = 0.1\ [Nms/rad]$, $g = 9.81\ [m/s^2]$ and $|\tau_{max}| = 1\ [Nm]$ (symbols defined in Section 4.1).
Regarding R3T and RG-RRT, we used a *reachable-set time horizon* of 0.2 s, while the *time step-size* for RRT and forward dynamics was $0.01[s]$.
Finally, the *task tolerance* is set to $||x_{start} - x_{goal}||_2 \leq 0.05$.

In the next sections will be presented the results from random runs from all three algorithms, showing for each of them the expanded nodes and the plot regarding the final solution.

RRT (Pendulum)

As shown in Figure 6, the algorithm has expanded quite a lot of nodes and most of them are close to each other. This is actually the expression of the main drawback of this algorithm: many external nodes are expanded multiple times without actually exploring the space resulting in many useless nodes clustered together.

Although RRT has shown to be inefficient, an admissible solution is still found (Fig. 7), which actually manages to complete the task with a short trajectory time ($5s$) and with only 3 oscillations.

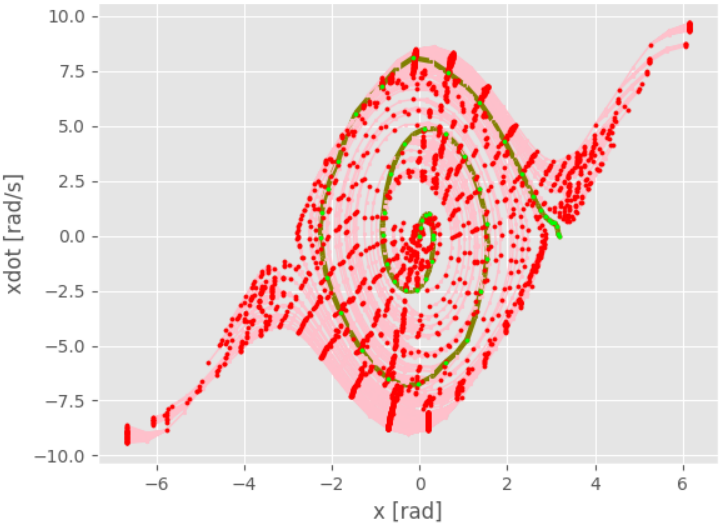

Figure 6: **2325 expanded nodes** with RRT.

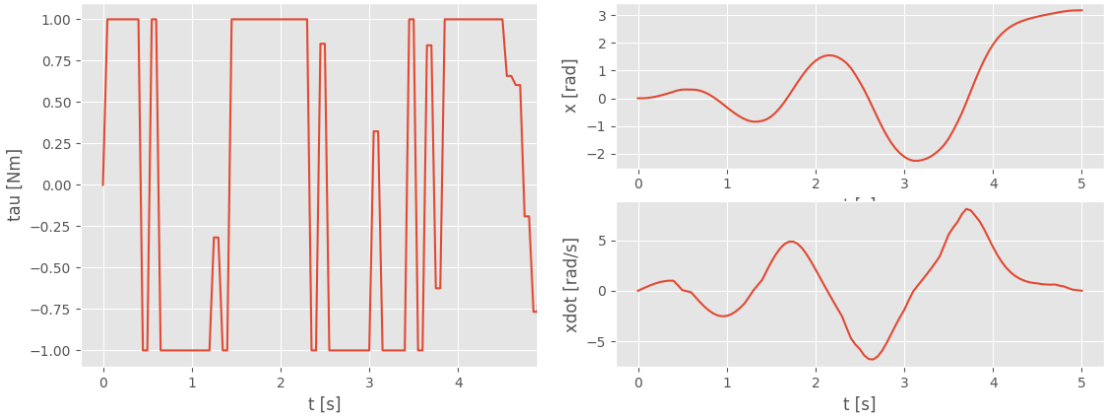

Figure 7: Inputs and trajectories in time

RG-RRT (Pendulum)

As shown in Figure 8, the algorithm has expanded less nodes and more uniformly, improving the exploration of χ. This is a positive results both in terms of computational cost and effectiveness of the algorithm itself, especially considering that we used only 3 motion primitives.

The resulting solution (Fig. 9) presents a trajectory time comparable to the solution provided by RRT, even thought the pendulum has to perform more swings.

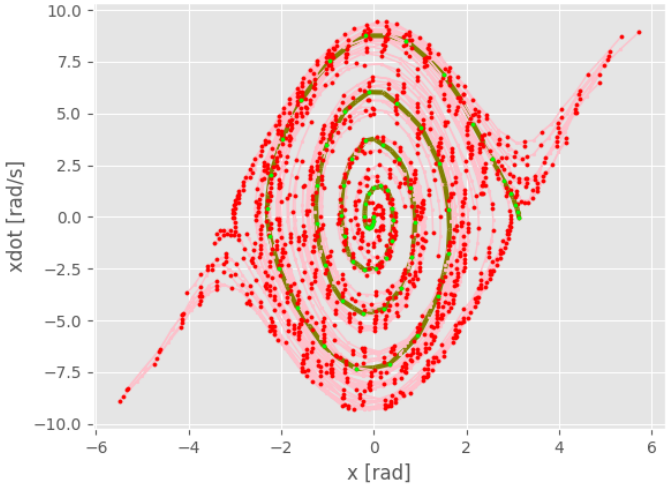

Figure 8: **1623 expanded nodes** with RG-RRT.

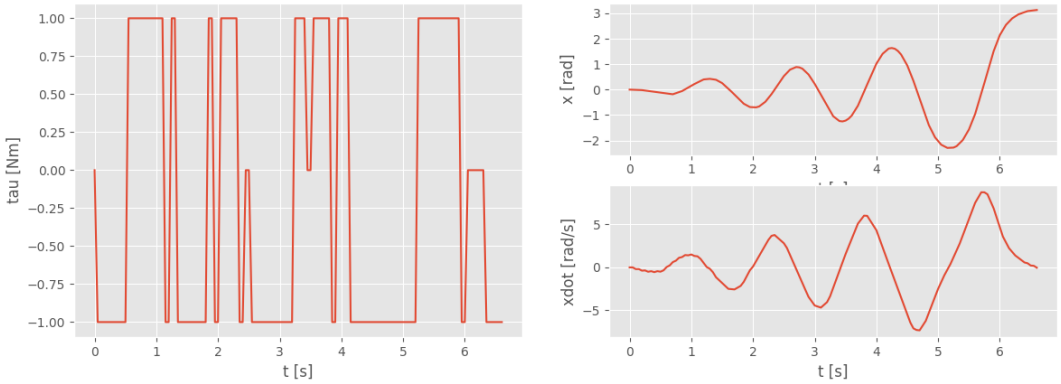

Figure 9: Inputs and trajectories in time.

R3T (Pendulum)

As shown in Figure 8, R3T with less nodes manages to explore the same region of space as the other algorithms thanks to the polytopic approximation for the reachable sets. The resulting solution (Fig. 9) is also improved, as it presents a lower trajectory time ($4s$) and only 3 oscillations.

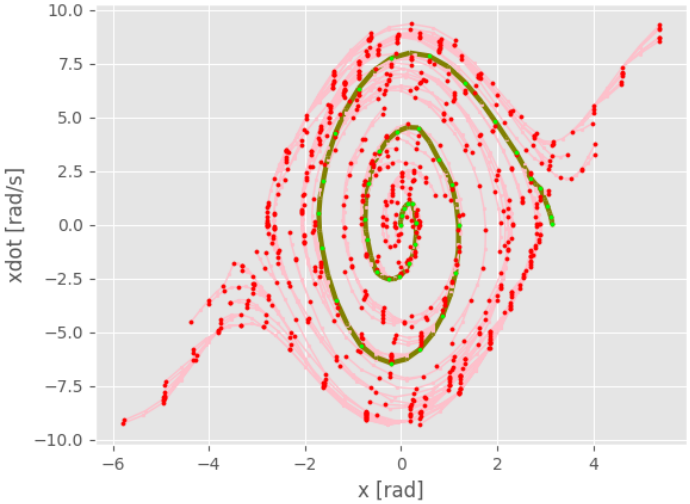

Figure 10: **916 expanded nodes** with R3T.

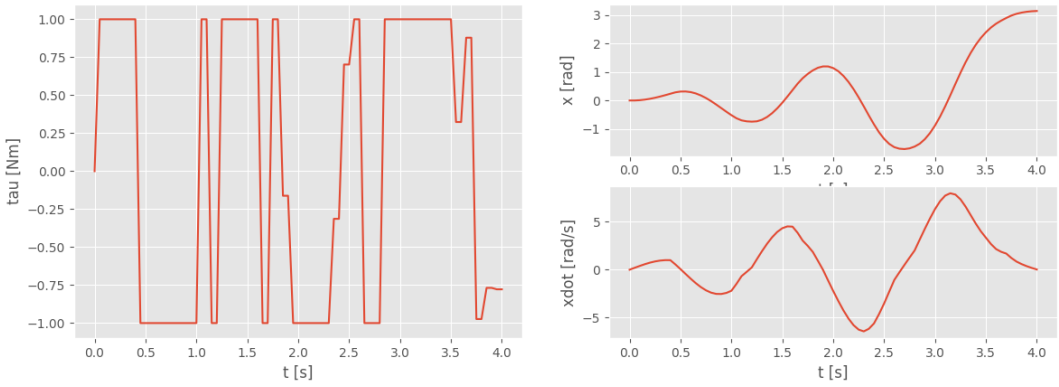

Figure 11: Inputs and trajectories in time.

Results comparison

10 consecutive planning tries have been performed for each algorithm and the results in terms of expanded nodes and average time are summarized in Table 1.

	RRT	RG-RRT	R3T
Mean	3430	2281	1050
Maximum	8996	5631	2004
Minimum	623	737	489
Mean time (s)	9.79	3.54	15.37

Table 1: motion Planning Statistics with Pendulum

As in [1], R3T has significantly outperformed RG-RRT and RRT. Specifically, the number of nodes expanded by RRT is much higher then the ones needed by the other two. Then, RG-RRT is very efficient in terms of computational time, but R3T is able to find a path in less nodes, proving its superior performances.

6.2 Hopper 1D

Task: For the hopper 1D the task is to "hop" from a lower resting state ($\dot{x}_{start} = 0\ [m]/s$) at ($x_{start} = 2\ [m]$) to an higher resting state ($\dot{x}_{goal} = 0\ [m/s]$) at ($x_{goal} = 3\ [m]$). As in in the pendulum case the idea is that, since the input is bounded, the system has to pump energy into the system in order to reach a state with higher energy. However, in this case the system actually switches dynamics modes, making the task unfeasible for RRT.

Simulation parameters: we chose to model the hopper 1D with $m = 1\ [Kg]$, $l = 1\ [m]$, $p = 0.1\ [m]$, $b = 0.9\ [Ns/m]$ and $g = 9.81\ [m/s^2]$ (symbols defined in

Section 4.2).The input boundary have been set to $f_{max} = 80\ [N]$.
Regarding R3T and RG-RRT we used a *reachable-set time horizon* of $0.4[s]$, while the forward dynamics was $0.01\ [s]$.
Finally, he task tolerance is set to $||x_{start} - x_{goal}||_2 \leq 0.1$.

For this model there are no solution found by RRT, therefore only the results provided by RG-RRT and R3T are shown and these allows us to prove the effectiveness of R3T on hybrid systems.

RG-RRT (Hopper 1D)

The expanded nodes are visible in Figure 12 in their $(x, \dot{x})$ coordinates. The plot shows a parabolic shape for $x > 1.1[m]$ $(= (l + p))$, as the system is in flight phase (the plot in pink indicates that those have not been inserted as real nodes) and then almost delineates a vertical lines for $x = 1.1[m](= l + p)$, corresponding to the *soft contact* nodes. The rest of the nodes are in the area and $1[m]$ $(= l) < x < 1.1[m]$ $(= l + p)$, corresponding respectively to inelastic collision mode. Then considering the final solution (Fig.13), the system manages to find the solution with 3 jumps.

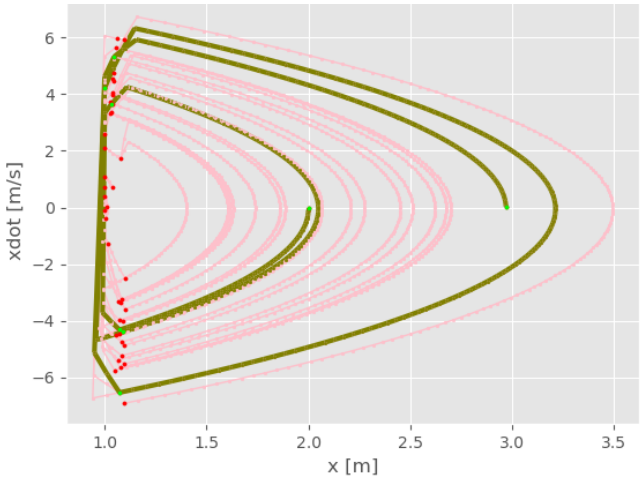

Figure 12: **58 expanded nodes** (in red) with RG-RRT. The final solution is highlighted in green.

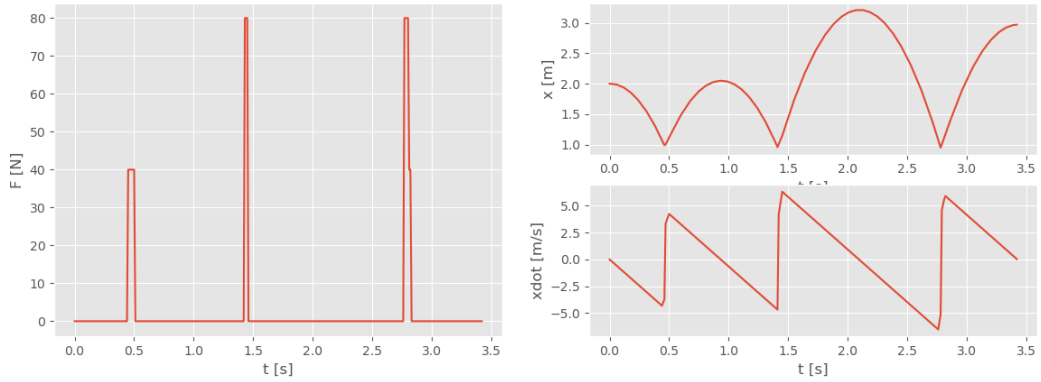

Figure 13: Inputs and trajectories in time.

R3T (Hopper 1D)

The expanded nodes are visible in Figure 14 in their $(x, \dot{x})$ coordinates. As in the RG-RRT case, the plot clearly resembles the behaviour of a vertical hopper 1D. The final solution (Fig. 15) obtained in this case is slightly better then the one obtained using RG-RRT in terms of expanded node.

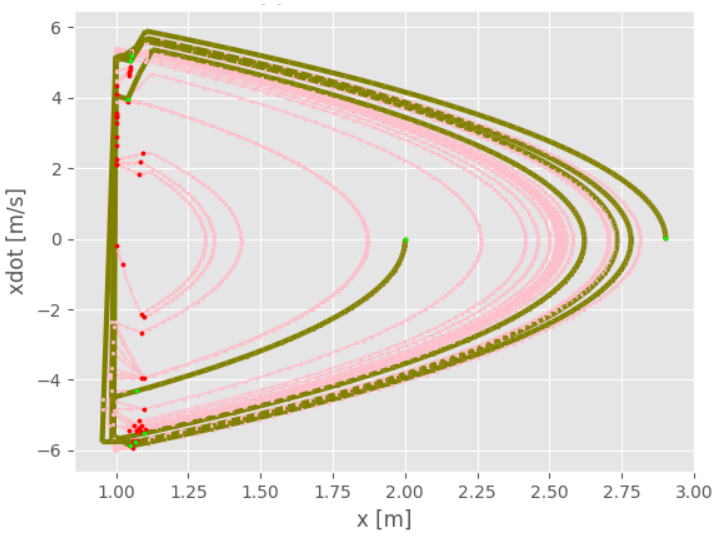

Figure 14: **53 expanded nodes** (in red) with R3T. The final solution is highlighted in green.

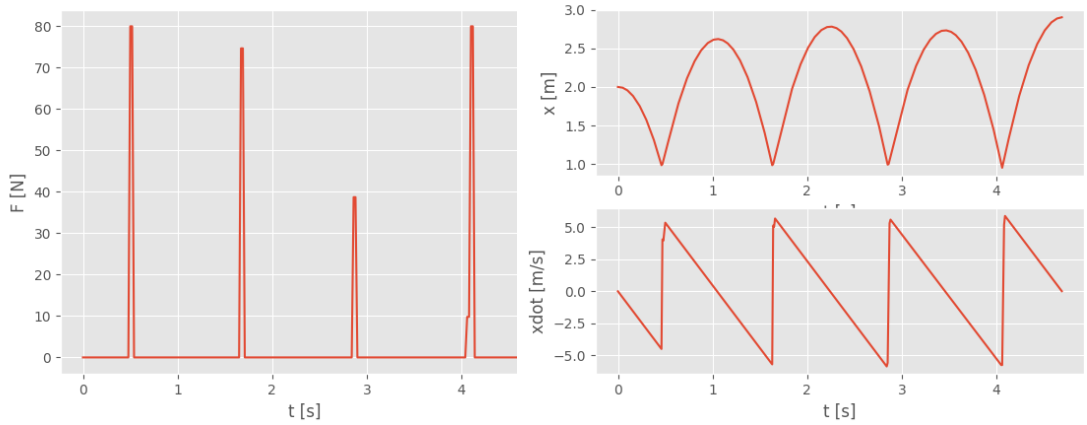

Figure 15: Inputs and trajectories in time.

Results comparison

10 consecutive planning tries have been performed for **RG-RRT** and **R3T**, whose results in terms of expanded nodes and average time are summarized in Table 2.

	RRT	RG-RRT	R3T
Mean	N/A	99	60
Maximum	N/A	256	145
Minimum	N/A	18	6
Mean time (s)	N/A	0.69	0.89

Table 2: motion Planning Statistics for Hopper 1d

R3T still outperforms RG-RRT, even if this time with a smaller margin.

6.3 Hopper 2D

Task: The task is to make the robot hop from a resting state with ($x_{start} = 0$ $[m]$, $y_{start} = 1.5$ $[m]$) to aims at reaching $x_{goal} = 10$ $[m]$. Both RG-RRT and R3T algorithm have been used to plan push-off and hip torque during contact, while a body-attitude controller was used in flight. In addition, the leg was modeled as a constant-k spring during compression.

Simulation parameters: we chose to model the hopper 2D with $m = 5$ $[Kg]$, $J = 500$ $[Kg \cdot m]$, $m_l = 1$ $[Kg]$, $J_l = 0.5$ $[Kg \cdot m]$, $l_1 = 0.0$ $[m]$, $l_2 = 0.0$ $[m]$, $k_g = 2e3[N/m]$, $b_g = 20$ $[Ns/m]$, $g = 9.81$ $[m/s^2]$ (symbols defined in Section 4.3). The input boundaries are $|\tau_{max}| = 500$ $[Nm]$ and $F_{max} = 2000$ $[N]$. In addition, we used a *reachable-set time horizon* of $0.4s$. and task tolerance is set to $||x_{start} - x_{goal}||_2 \leq 0.1$.

In the following paragraph we reported the evolution of x_{foot}, y_{foot}, $\dot{x}_{foot}$, $\dot{y}_{foot}$ (Fig. 18) and of the inputs (Fig. 19) in time, as well as final stroboscopic view of the simulation in Figure 17 got from a random application of R3T to the hopper 2D system.

R3T (Hopper 2D)

As shown in figure 16, the algorithm expands a limited number of nodes before managing to converge to a final solution. To be noticed that also some nodes with a negative x value, even if the starting configuration has $x_{foot} = 0$ $[m]$. This is due to the fact that, although the hip sampler picks hip coordinates with positive x, once converted into foot coordinates it may lead to a negative value (up to $x_{foot} = -5$). Therefore, the algorithm sometimes expands in the opposite direction to the goal. In addition, it can be also seen that some nodes with a negative y value. This is actually due to the elastic behaviour of the leg during the descending contact to the ground, as the virtual penetration indicates its compression.

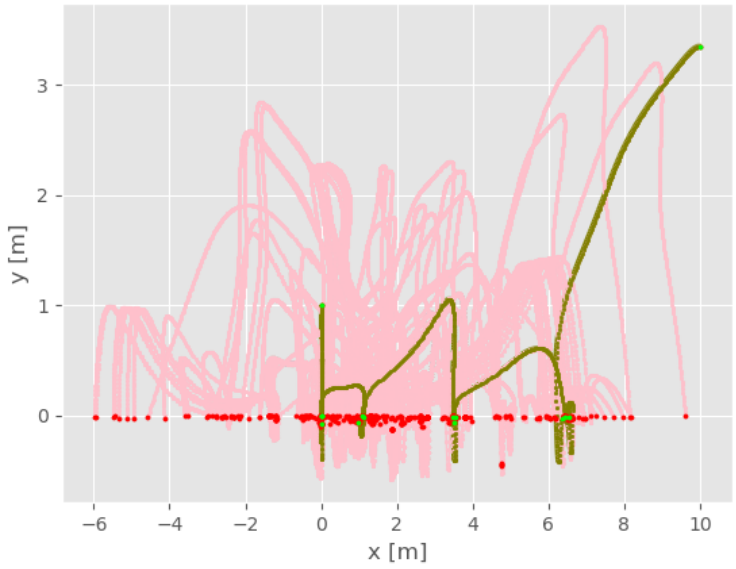

Figure 16: **509 expanded nodes** with R3T, visible in the (x,y) coordinates. The final solution is highlighted in green.

Regarding the resolution a stroboscopic view is shown in Figure 17. The robot starts in a fling descent mode, as the initial state has $y = 1.5m$ and clearly makes 5 jumps before reaching the goal, as also underlined in Figure 18. Meanwhile, the inputs commands (Fig. 19) are consistent with the behaviour of the hopper, as they never exceed the boundaries (they have been clamped every time a node is inserted) and the force F commanded to the prismatic joint only during contact (the timesteps when $y < 0m$ in Figure 18).

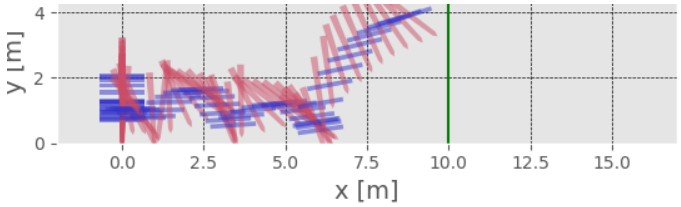

Figure 17: Stroboscopic view of the hopper 2d during the execution of the motion.

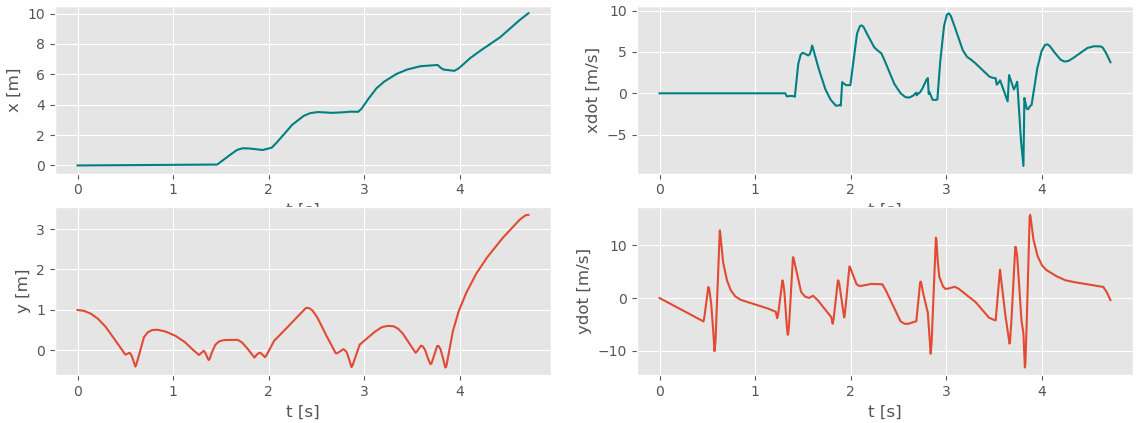

Figure 18: Evolution of the (x,y) coordinates and their derivative in time.

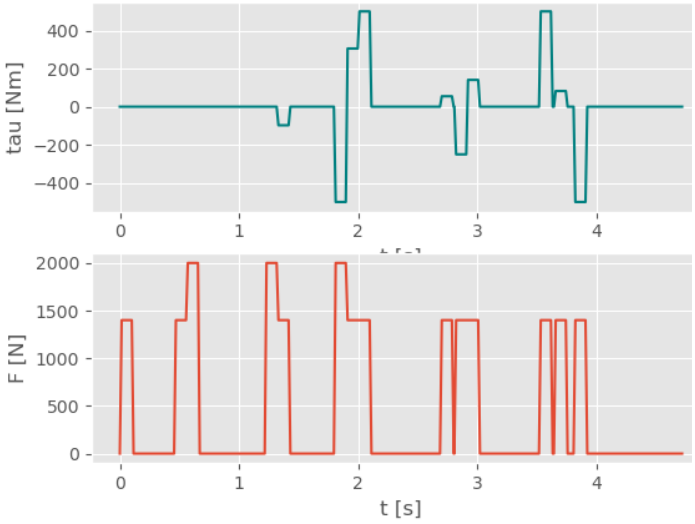

Figure 19: Inputs in time

After performing 10 consecutive planning tries, the algorithm shows a **maximum number of nodes** of **1247**, a **minimum number of nodes** of **318**, an **average number of nodes** of **651** and an **average run time** for a successful run is of **231.7s**.

7 Conclusions

In this project we implemented **R3T** and compared it with **RRT** and **RG-RRT**, showing the differences in the algorithms and performance on a set of benchmark planning problems for kinodynamically constrained non linear systems. The results we obtained underline the power of R3T in solving complex kinodynamic problems, but on the other hand also show that RG-RRT for the model of the hopper 1d has very interesting performances. Specifically, RG-RRT achieves good results in terms of number of nodes expanded mainly due to the approach explained in section 5.2, which has proven to be very efficient. Conversely, the power of R3T lies in its general approximation of the reachable sets, which can be applied to any system which admits a locally linear approximation. As the input space becomes larger in dimensions we expect even better results. As a matter of fact, for systems with low input spaces there is generally not much to lose approximating the reachable sets with motion primitives (our implementation of RG-RRT), and little to be gained using polytopes (R3T). Using motion primitives also has other favorable properties: the states are by construction always reachable and most importantly there's no need to solve for the input when expanding a node (just pick the associated motion primitive).

However, recent works have shown that RRT variants which choose the best motion primitive with a fixed time step, like our implementation of RG-RRT, may not be probabilistically complete in the general case [9]. Moreover, we argue that in general the number of control inputs needed to cover accurately the input space is exponential with the input dimension (*curse of dimensionality*). This is probably when more clever reachable sets approximation such as *AH-polytopes* may perform better. The reason is that only one polytope is needed regardless of the input space dimension. The complexity of the bottleneck of R3T, which is computing the distance from point to polytope, is polynomial with the input dimension using interior point algorithms (it is a convex problem) thus it is expected to scale better.

References

[1] Albert Wu, Sadra Sadraddini, and Russ Tedrake. R3t: Rapidly-exploring random reachable set tree for optimal kinodynamic planning of nonlinear hybrid systems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4245–4251, 2020.

[2] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. Reachability-guided sampling for planning under differential constraints. In *2009 IEEE International Conference on Robotics and Automation*, pages 2859–2865, 2009.

[3] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 473–479 vol.1, 1999.

[4] 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*, 12(4):566–580, 1996.

[5] Sadra Sadraddini and Russ Tedrake. Linear encodings for polytope containment problems, 2019.

[6] Albert Wu, Sadra Sadraddini, and Russ Tedrake. The nearest polytope problem: Algorithms and application to controlling hybrid systems. In *2020 American Control Conference (ACC)*, pages 1815–1822, 2020.

[7] Dustin J. Webb and Jur van den Berg. Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints, 2012.

[8] Marc H. Raibert. Hopping in legged systems — modeling and simulation for the two-dimensional one-legged case. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(3):451–463, 1984.

[9] Tobias Kunz and Mike Stilman. Kinodynamic rrts with fixed time step and best-input extension are not probabilistically complete. In *Workshop on the Algorithmic Foundations of Robotics*, 2014.