Flat-RRT*: A sampling-based optimal trajectory planner for differentially flat vehicles with constrained dynamics ⋆

Luca Bascetta * **Iñigo Mendizabal Arrieta** ** **Maria Prandini** *

* *Politecnico di Milano, Piazza Leonardo da Vinci, 32 - 20133 Milano (Italy) (e-mail: {luca.bascetta, maria.prandini}@polimi.it)*
** *Management Center Innsbruck - 6020 Innsbruck (Austria) (e-mail: i.mendizabal@mci4me.at)*

Abstract: This paper introduces the flat-RRT* algorithm, which is a variant of the optimal Rapidly exploring Random Tree (RRT*) planner, accounting for actuation constraints on the vehicle dynamics in the optimal trajectory design. The proposed algorithm is applicable to vehicles that can be modelled with differentially flat dynamics, like unicycle and bicycle kinematics. The main idea is to exploit the flatness property so as to finitely parametrize trajectories, and design a set of motion primitives that represent optimal constrained trajectories joining two configurations in a grid space. A procedure to determine constrained (though sub-optimal) trajectories joining arbitrary configurations based on the motion primitives is then proposed. This eases and accelerates the construction of the tree to the purpose of online trajectory (re)planning in an uncertain environment, where the obstacle map may be continuously updated as the vehicle moves around, or unexpected events may occur and alter the free configuration space.

Keywords: Trajectory and Path Planning; Autonomous Vehicles; Application of nonlinear analysis and design; Differentially flat systems; Optimal constrained trajectory design.

1. INTRODUCTION

In the last decade the number of applications of autonomous aerial, underwater, and ground vehicles has been continuously increasing, with the side effect of stimulating the scientific research on autonomy, efficiency, reliability, and safety. As far as autonomy of a vehicle is concerned, planning, control, and perception play all a crucial role and affect both safety and performance. Among them, planning is particularly important, as it is responsible not only of computing an obstacle-free trajectory that takes the vehicle to the desired location, but also of accommodating for actuation and kino-dynamic vehicle constraints, while minimizing the trajectory cost.

Two main approaches have been developed in the robotics literature to address the planning problem, i.e., search-based (Pivtoraiko et al. (2009); Likhachev and Ferguson (2009)) and sampling-based methods. This latter one represents the most widespread "practical approach", as it provides an effective technique to plan in high-dimensional spaces with a guarantee of probabilistic completeness, i.e., an obstacle-free trajectory is found, when there exists one, with probability that tends to one as the number of samples grows to infinity. The most popular sampling-based algorithms are Probabilistic RoadMaps (PRM) (Kavraki et al. (1996, 1998)), Rapidly-exploring Random Tree (RRT) (LaValle and Kuffner (2001)), and optimal Rapidly-exploring Random Tree (RRT*) (Karaman and Frazzoli (2010)). The interested reader is referred to (Ragaglia et al. (2015)) for a review on sampling-based algorithms and further references. Here we shall focus on the RRT* algorithm, which is an incremental sampling-based planner developed for searching an obstacle-free optimal trajectory in high-dimensional continuous state spaces. Like RRT (LaValle and Kuffner (2001)), it is based on the generation of a tree that randomly explores the free space. In addition, thanks to the introduction of a rewiring process, it can guarantee almost sure convergence to a globally optimal solution. The core of the algorithm is the tree extension procedure, that starts randomly sampling a new configuration (node) within the free portion of the configuration space. A minimum cost trajectory satisfying a set of differential constraints and connecting the newly sampled configuration to some node in the tree is then computed and added to the tree. Finally, the tree rewiring procedure is applied, to check if a minimum cost trajectory reaching any other node in the tree and passing through the newly added one exists.

As discussed in (Ragaglia et al. (2015)), in the presence of vehicle kino-dynamic constraints (e.g., differential and actuation constraints), computing the minimum cost trajectories to extend the tree becomes computationally challenging because it involves solving many boundary value problems. To overcome this limitation two different approaches can be adopted. The first one looks for an analytical or semi-analytical solution of the boundary value problem, as in (Ragaglia et al. (2015)) where a semi-analytical solution based on optimal and model predictive

⋆ Research was supported by the European Commission, H2020, under the project UnCoVerCPS, grant number 643921.

control is adopted. A completely different approach, inspired to search-based planners (Likhachev and Ferguson (2009)), can be devised moving the computational complexity out of the planning procedure, i.e., pre-computing a set of motion primitives that can be then used to design efficiently nearly-optimal trajectories that further extend the tree. The present paper follows this latter approach and introduces flat-RRT*, an RRT* based planner that exploits the flatness property of a nonlinear system to pre-compute a set of motion primitives that satisfy kino-dynamic constraints and are optimal with respect to a given cost function. A database of these motion primitives is built and used for fast computation of nearly-optimal constrained trajectories between new configurations.

The reformulation of the vehicle model as a differentially flat system is available for a variety of different kinematics (see e.g., Buccieri et al. (2009) and Fuchshumer et al. (2005)), and it allows to express the motion primitives as finitely parametrized functions of time in the flat output space. The optimal constrained trajectory design can then be recast as a boundary optimization problem where some parameters are determined by solving (offline) a constrained nonlinear optimization program and the remaining parameters are computed (online) by solving a linear set of equations corresponding to the boundary conditions. RRT* is finally used to plan (possibly on-the-fly) a (nearly)-optimal feasible and obstacle-free trajectory, where the sequence of vehicle configurations, starting from the initial one and ending with one that belongs to the goal region, is connected through trajectories that are close to the motion primitives. The time required to expand the tree is greatly reduced because of the pre-computed database of motion primitives, and this enables iterative re-planning of the trajectory based, e.g, on the updated measurements of the vehicle within its sensor field of view.

The paper is structured as follows. We explain the motion primitives and nearly-optimal trajectories construction in Section 2, and describe the flat-RRT* algorithm in Section 3. A numerical example shows the performance of the proposed approach (Section 4). Some concluding remarks are drawn in Section 5.

2. MOTION PRIMITIVES AND TRAJECTORY GENERATION

We now show how to compute an optimal trajectory for a vehicle with differentially flat dynamics, so as to drive it from an initial to a final configuration with minimum control effort while satisfying kino-dynamic constraints.

The definition of flat system was originally introduced in Fliess et al. (1995) and makes reference to dynamical systems whose states and inputs can be expressed in terms of a certain set of variables, called *flat outputs*, and their derivatives. More formally, a nonlinear system $\dot{s} = f(s,u)$ with state $s \in \mathbb{R}^n$ and input $u \in \mathbb{R}^m$ is called *differentially flat* if there exists a flat output vector $z \in \mathbb{R}^m$ that is a function of s, u and the time derivatives of u up to order ρ, i.e., $z = \eta(s, u, \dot{u}, \ddot{u}, ..., u^{(\rho)})$, such that s and u can be expressed as a function of z and its derivatives up to order σ and $\sigma+1$, respectively, as follows

$$s = \phi_s(z, \dot{z}, ..., z^{(\sigma)}), \quad u = \phi_u(z, \dot{z}, ..., z^{(\sigma+1)}). \tag{1}$$

In the solution of the boundary value problem that allows to compute an optimal trajectory, this property can be exploited to express the trajectory itself in terms of the flat outputs, which are in turn parametrized as a linear combination of basis functions of time. By imposing that the trajectory starts at the given initial configuration and ends at the final configuration (boundary conditions), the number of parameters that can be freely selected is reduced. Still, by introducing a sufficiently large number of coefficients in the expansion, one is left with some degrees of freedom, say μ, that can then be spent to solve the constrained optimal trajectory planning problem, thus obtaining an optimal value μ°. The idea is then to generate a set of motion primitives by gridding the configuration space and defining a finite set of boundary conditions. For each boundary condition one can compute the corresponding optimal μ° by solving a constrained optimization problem, where the control effort to reach the final configuration is minimized subject to the vehicle kino-dynamics constraints. These optimal values μ° are stored in a database that is then integrated in the flat-RRT* algorithm. When a new pair of initial and final configurations are given, one can look for the motion primitive that is closest in terms of boundary conditions, take its μ° and then determine the remaining parameters based on the actual boundary conditions. A feasibility test regarding constraint fulfilment is then run, to check that the computed trajectory is admissible. Though the proposed approach is general (and can be applied to any nonlinear system for which a flatness formulation exists), here we describe it in details for a unicycle vehicle.

2.1 Unicycle model of the vehicle

A unicycle is a simplified model of an anholonomous vehicle moving in a two dimensional space. It is characterized by three state variables, position (x, y) and orientation θ, and two inputs, linear velocity v and angular velocity ω:

$$\begin{cases} \dot{x} = v \cdot \cos(\theta) \\ \dot{y} = v \cdot \sin(\theta) \\ \dot{\theta} = \omega \end{cases} \tag{2}$$

Unicycle-like models are flat systems with flat outputs that are functions of the state, and given by $z_1 = x$ and $z_2 = y$ (see Buccieri et al. (2009) and De Luca et al. (2001)). As for (1), the state vector $s = [x\ y\ \theta]'$ can be recovered from $z = [z_1\ z_2]'$ and its first derivative ($\sigma = 1$) as follows:

$$x = z_1, \quad y = z_2, \quad \theta = \begin{cases} \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 > 0 \\ \pi + \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 < 0 \\ \frac{\pi}{2}\text{sign}(\dot{z}_2) & \dot{z}_1 = 0 \end{cases}$$

where sign denotes the sign function.

By simple derivations, one can verify that relation

$$\omega = \dot{\theta} = \frac{1}{1 + \left(\frac{\dot{y}}{\dot{x}}\right)^2} \cdot \frac{\dot{x} \cdot \ddot{y} - \dot{y} \cdot \ddot{x}}{\dot{x}^2} = \frac{\dot{x} \cdot \ddot{y} - \dot{y} \cdot \ddot{x}}{\dot{x}^2 + \dot{y}^2}$$

follows from the state equation in (2). The input $u = [v\ \omega]'$ can then be recovered from z, $\dot{z}$, and $\ddot{z}$ via map ϕ_u in (1), which takes the form:

$$v = \sqrt{\dot{z}_1^2 + \dot{z}_2^2}, \quad \omega = \frac{\dot{z}_1 \cdot \ddot{z}_2 - \dot{z}_2 \cdot \ddot{z}_1}{\dot{z}_1^2 + \dot{z}_2^2}. \tag{3}$$

Note that map $u = \phi_u(z, \dot{z}, \ddot{z})$ defined in (3) has a singularity when $\dot{z}_1^2 + \dot{z}_2^2 = 0$, since in that case the linear velocity v is zero and, hence, the angular velocity ω (and also the orientation θ) are not well-defined. In

the literature, different approaches have been proposed to cope with this singularity. In Buccieri et al. (2009), the singularity is avoided by resetting the velocity whenever it becomes small. In De Luca et al. (2001), if the linear velocity is zero at the initial time $t = 0$, then "higher-order differential information at $t = 0$ are used in order to determine the consistent initial orientation and the initial angular velocity command"; and if the velocity becomes zero at some time instant $t > 0$ along the geometric path, then "a continuous motion is guaranteed by keeping the same orientation attained at t^-". Here, we address it in a different way as explained in Remark 1 of Section 2.3.

2.2 Optimal trajectory design and the motion primitives

Our goal is to determine a final time instant $t_f > 0$ and an input $u : [0, t_f] \to \Re^2$ so as to drive the system (2) from an initial configuration q_0 to a final configuration q_f, while minimizing the cost

$$\int_0^{t_f} u'(\tau)\, R\, u(\tau) d\tau \tag{4}$$

where $R = R'$ is a positive definite matrix weighting the control effort, subject to the constraints

$$|v(t)| \le v_{\max}, \quad |\omega(t)| \le \omega_{\max},\ t \in [0, t_f]. \tag{5}$$

Initial and final configurations are specified in terms of position (x, y), orientation θ, and velocity v, i.e., $q_0 = (x_0, y_0, \theta_0, v_0)$ and $q_f = (x_f, y_f, \theta_f, v_f)$. Without loss of generality, we assume that the initial configuration q_0 is characterized by $x_0 = y_0 = \theta_0 = 0$, i.e., the vehicle is in the origin of the Cartesian coordinate system with orientation θ equal to zero. If this were not the case, a change of coordinates (roto-translation) could be performed to set the initial state configuration to zero. The initial coordinates can then be recovered by the inverse transformation.

To the purpose of solving the constrained optimization problem above, we adopt the reformulation of the unicycle model (2) in terms of its flat outputs $z_1 = x$ and $z_2 = y$, which are linearly parametrized as follows:

$$z_1 = \sum_{i=0}^{4} a_i \cdot t^i, \quad z_2 = \sum_{j=0}^{3} b_j \cdot t^j. \tag{6}$$

By imposing the boundary conditions, we get the following set of 8 equations in the 9 parameters a_i, $i = 0 \dots 4$ and b_j, $j = 0 \dots 3$, plus the final time t_f:

$$\begin{aligned}
&z_1(0) = a_0 = 0, \quad z_2(0) = b_0 = y_0,\\
&\dot z_1(0) = a_1 = v_0, \quad \dot z_2(0) = b_1 = 0,\\
&z_1(t_f) = \sum_{i=0}^{4} a_i \cdot t_f^i = x_f, \quad z_2(t_f) = \sum_{j=0}^{3} b_j \cdot t_f^j = y_f,\\
&\dot z_1(t_f) = \sum_{i=1}^{4} i \cdot a_i \cdot t_f^{i-1} = v_f \cos(\theta_f),\\
&\dot z_2(t_f) = \sum_{j=1}^{3} j \cdot b_j \cdot t_f^{j-1} = v_f \sin(\theta_f).
\end{aligned} \tag{7}$$

Based on (7), we can then express all parameters as a function of $\mu = (a_4, t_f)$, and then determine an optimal value μ° for μ by solving the following nonlinear constrained optimization problem:

$$\begin{aligned}
&\min_{\mu} J(\mu) \qquad (8)\\
&\text{subject to:}\\
&\dot z_1^2(t) + \dot z_2^2(t) \le v_{\max}^2,\ t \in [0, t_f]\\
&\left(\frac{\dot z_1(t) \cdot \ddot z_2(t) - \dot z_2(t) \cdot \ddot z_1(t)}{\dot z_1^2(t) + \dot z_2^2(t)}\right)^2 \le \omega_{\max}^2,\ t \in [0, t_f],
\end{aligned}$$

where we substitute (3) in (5). These constraints translate into constraints on μ, since the flat outputs z_1 and z_2 and their derivatives are functions of a_4 and t_f given the parametrization (6) and the boundary conditions (7). As for the cost function (8), it is given by (4).

A uniform gridding of the configuration space can be adopted to define the set of initial and final configurations for the motion primitives, with the initial configuration q_0 with all entries equal to zero except for the linear velocity, i.e., $q_0 = (0, 0, 0, v_0)$. Each motion primitive is then characterized by its optimal $\mu^\circ = (a_4^\circ, t_f^\circ)$ parametrization computed by solving the constrained optimization problem (8) for its specific initial and final configurations. All other parameters defining the motion primitive via the $x = z_1$ and $y = z_2$ linear expansions in (6) can be determined by solving the linear equations (7) with $a_4 = a_4^\circ$ and $t_f = t_f^\circ$. Finally, the inputs v and u can be determined via (3).

From an implementation perspective, the set of motion primitives can be represented by two matrices, A_4 and T_f, containing their a_4° and t_f° values and indexed by the initial and final configurations identifying the primitives. The collection of these two matrices is referred to in the sequel as the database of the motion primitives, and each pair (a_4, t_f) corresponding to the same indices in A_4 and T_f is called an entry in the database.

2.3 Trajectory design based on the motion primitives

Suppose that an initial configuration q_0 and a final configuration q_f are given. Then, after an appropriate change of coordinates (rotation and translation) we get $\tilde q_0 = (0, 0, 0, v_0)$ and $\tilde q_f = (\tilde x_f, \tilde y_f, \tilde\theta_f, \tilde v_f)$, and derive the boundary values $z(0) = [0\,0]'$, $\dot z(0) = [v_0\,0]'$, $z(t_f) = [\tilde x_f\ \tilde y_f]'$, and $\dot z(t_f) = [\tilde v_f \cos(\tilde\theta_f)\ \tilde v_f \sin(\tilde\theta_f)]'$. We can then select the motion primitive whose boundary values are closest to the computed ones according to the Euclidean distance, and take the corresponding entry (a_4°, t_f°) from the database. Based on a_4° and t_f°, all the other parameters defining the trajectory can be determined by solving the linear equations (7) with $a_4 = a_4^\circ$ and $t_f = t_f^\circ$. The trajectory coordinates x and y are then given by (6), and the inputs v and ω can be determined via (3). The admissibility of the obtained (sub-optimal) trajectory can be checked by verifying that inputs satisfy constraints in (5). The trajectory can be described in the original coordinate system applying the inverse of the roto-translation adopted to get $\tilde q_0 = (0, 0, 0, v_0)$ from q_0.

Remark 1. (singularity issue). When the velocity of either the initial or the final configuration is small, close to zero, the singularity issue due to the denominator in (3) getting close to zero arises. This issue is solved here by adding a linear piece of trajectory that brings the velocity to a non-zero value $v_{\min}$ through a predefine (linear) acceleration/deceleration. Suppose, for instance, that the velocity is small (almost zero) at the initial configuration q_0: $v_0 \simeq 0$. Then, we shall introduce a configuration q_0' with velocity $v_{\min}$ that is reached from q_0 via a straight

line trajectory with the same orientation as q_0, travelled at some constant acceleration $\bar{a}$. This trajectory is then pieced together with the one designed by considering q'_0 as initial configuration and q_f as final one. This latter one is determined based on the database of the motion primitives according to the procedure described above. The time to reach q_f from q_0 will then be given by the sum of the time to reach q'_0 from q_0 (which is equal to $(v_{\min} - v_0)/\bar{a}$), and the time t_f to reach q_f from q'_0 as given by the database search. A similar procedure is followed if the velocity v_f at the final configuration q_f is close to 0. In this case, q_f is reached by slowing down the velocity at constant deceleration $-\bar{a}$ along a linear trajectory starting from a configuration q'_f with $v'_f = v_{\min}$. The trajectory from q_0 to q'_f is determined based on the motion primitives. □

3. THE FLAT-RRT* PLANNING ALGORITHM

Flat-RRT* is an extension of RRT* aiming at making tractable the problem of computing an edge of the tree even considering kinematics, dynamics, and actuation constraints. As a consequence, the peculiarities of flat-RRT* with respect to RRT* are in the edge computation procedure. For this reason, we first describe the main steps of RRT*, and then detail the edge computation procedure. We start introducing the following definitions:

- a node or configuration $q \in Q$ is a vector $q = (x, y, \theta, v)$ representing the pose and velocity of the vehicle;
- Q is the bounded set of the vehicle configurations, i.e., $Q = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [\theta_{min}, \theta_{max}] \times [v_{min}, v_{max}]$;
- $Q_{free} \subset Q$ is the subset of the configuration space that is free of obstacles;
- $Q_{goal} \subset Q_{free}$ is the goal region the planned trajectory has to reach;
- q_{start} and $q_{goal} \subset Q_{goal}$ are the initial and final vehicle configurations;
- $Q_T \subset Q_{free}$ and E_T are the sets of nodes and edges representing the tree T, i.e., $T = (Q_T, E_T)$;
- N is the maximum cardinality of Q_T;
- $e_{q_1,q_2} \in E_T$ is the edge that connects q_1 to q_2, where $q_1, q_2 \in Q_T$;
- $C(e_{q_1,q_2})$ is the cost associated to e_{q_1,q_2}, computed here as the time t_f needed to reach q_2 from q_1;
- $C(\rightarrow q)$ is the total cost of the best trajectory starting from q_{start} and ending at q;
- $Q_{reach}(q) \subset Q_T$ is the set of all nodes that are inside a d-dimensional ball of radius γ_{ball} centred in q, and is defined as $Q_{reach}(q) = \{\bar{q} \in Q_T | \, \|q - \bar{q}\|_2 \leq \gamma_{ball}\}$ where $d = 4$ is the dimension of the configuration space, and $\gamma_{ball} = \gamma_{RRT^*}(\log |Q_T| \, / |Q_T|)^{1/d}$ with $\gamma_{RRT^*} > 2(1+1/d)^{1/d}(\mu(Q_{free})/\eta_d)^{1/d}$, $\mu(Q_{free})$ and η_d being the volume of Q_{free} and the volume of the unit ball in the d-dimensional Euclidean space, respectively (see Karaman and Frazzoli (2010) for further details).

3.1 The RRT algorithm*

The RRT* algorithm is composed by the following steps (that hold for flat-RRT* as well):

1) Tree initialisation: the algorithm starts with a tree $T = (Q_T, E_T)$ whose set of edges E_T is empty, while the set of nodes Q_T is initialized with the initial configuration, i.e., $Q_T = \{q_{start}\}$.

2) Random sampling: a new configuration q_{rand} is randomly sampled from Q_{free} according to a uniform distribution.

3) Neighbour radius computation: in order to reduce the number of nodes considered by the algorithm at each iteration, in particular in steps (4) and (5), we introduce the notion of neighbour of a node. The radius r of this neighbour can be computed using the following equation:

$$r = \underset{q \in Q_{reach}(q_{rand})}{\operatorname{argmax}} \left(\max\{C(e_{q,q_{rand}}), C(e_{q_{rand},q})\}\right)$$

4) Minimum-cost trajectory selection: a minimum-cost trajectory connecting q_{rand} to the tree is determined performing the following steps:

- the minimum cost collision-free edge $e_{q_{min},q_{rand}}$, among the edges $e_{q,q_{rand}}$, $q \in Q_T$, is found by computing
$$q_{min} = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} \left(C(\rightarrow q) + C(e_{q,q_{rand}})\right)$$
where set $\mathcal{Q}$ is defined as $\mathcal{Q} = \{q \in Q_T | C(e_{q,q_{rand}}) \leq r \wedge \text{CFree}(e_{q,q_{rand}})\}$. $\text{CFree}(e_{q_1,q_2})$ is a function that returns true when e_{q_1,q_2} is a collision-free trajectory, false otherwise, and the procedure adopted in flat-RRT* to compute the edges $e_{q,q_{rand}}$ and their costs $C(e_{q,q_{rand}})$ is detailed in the following of this section.
- q_{rand} and $e_{q_{min},q_{rand}}$ are added to the tree:
$$Q_T = Q_T \cup \{q_{rand}\} \qquad E_T = E_T \cup \{e_{q_{min},q_{rand}}\}$$

5) Tree rewiring: in order to ensure the optimality of the computed trajectory, every time a new node q_{rand} is added to the tree one has to check if a minimum-cost trajectory reaching any other node inside the tree and including q_{rand} exists. As a consequence, $\forall q \in Q_T$ if

- $e_{q_{rand},q}$ is a collision-free trajectory,
- $C(e_{q_{rand},q}) \leq r$,
- $C(\rightarrow q_{rand}) + C(e_{q_{rand},q}) < C(\rightarrow q)$,

the tree is rewired: $E_T = \{E_T \setminus \{e_{prev}\}\} \cup \{e_{q_{rand},q}\}$ where e_{prev} is the previous edge connecting q to the tree.

6) Termination: the algorithm keeps iterating steps (2) to (5), until $|Q_T| = N$.

7) Optimal trajectory: if the goal configuration set has been reached, i.e., $Q_{goal} \cap Q_T \neq \emptyset$, the minimum cost-to-go node inside Q_{goal} is selected: $q_{goal} = \operatorname{argmin}_{q \in (Q_{goal} \cap Q_T)} C(\rightarrow q)$, and the sequence of edges connecting q_{start} with q_{goal}, each one represented by its a_i and b_i vectors and by t_f, is returned along with the entire tree T.

3.2 The edge computation procedure

We next detail the procedure used in flat-RRT* to compute an edge e_{q_1,q_2} connecting two configurations q_1 and q_2, and to associate to this edge a cost $C(e_{q_1,q_2})$.

Assume that both the configurations q_1 and q_2 have a linear velocity which is different from zero. Then, the computation of the edge e_{q_1,q_2}, and the associated cost $C(e_{q_1,q_2})$ involves the following steps:

1) Normalizing the initial and final configurations: as explained in Section 2.3, in order to determine the (sub-optimal) trajectory between the initial and final configurations q_1 and q_2, a roto-translation is applied to the initial

and final pose, so that the initial pose (x_0, y_0, θ_0) is $(0, 0, 0)$.
2) Searching the database for (a_4, t_f): the entry (a_4, t_f) in the database corresponding to the motion primitive with boundary condition values closest, in Euclidean norm, to those associated with $\tilde{q}_1$ and $\tilde{q}_2$ is extracted as detailed in Section 2.3.
3) Enforcing the boundary conditions and determining the edge: all the other polynomial coefficients required to compute the trajectory joining $\tilde{q}_1$ and $\tilde{q}_2$ are computed enforcing the boundary conditions as in (7). The inverse roto-translation is applied to recover the trajectory joining q_1 and q_2 in the original coordinates x and y, and, hence, determine the edge e_{q_1,q_2}.
4) Checking for feasibility and assigning a cost: the edge e_{q_1,q_2} obtained with the values (a_4, t_f) extracted from the database may violate the actuation constraints in equation (5). If constraints are violated, then the cost $C(e_{q_1,q_2})$ is set equal to $+\infty$, otherwise, $C(e_{q_1,q_2})$ is set equal to t_f.

In the case when either q_1 or q_2 have a linear velocity which is zero (or close to zero), Remark 1 applies and, hence, a linear piece of trajectory is added in a preliminary step (0) to have both the initial and final configurations with a non zero linear velocity, then steps (1) to (4) are applied to the new initial and final configuration pairs with the only difference that the cost in step (4) is the sum of the computed t_f and the time needed to travel the added piece of linear trajectory.

Remark 2. (infeasible trajectory). At step (4) of the edge computation procedure, edge e_{q_1,q_2} is assigned a cost that is infinity when the trajectory is infeasible. Edge e_{q_1,q_2} will be eventually discarded at step (4) of the RRT* algorithm, when the minimum-cost trajectory is selected. □

4. A NUMERICAL EXAMPLE

A vehicle with unicycle kinematics, as described in (2), is moving in a square region $[0, 8] \times [0, 8]\, m$ characterized by 5 non-overlapping circular obstacles with radius $1\, m$ centred at $(2.25, 2.25)$, $(2.25, 4.75)$, $(4.75, 2.25)$, $(4.75, 4.75)$, $(3.5, 6.75)$. The vehicle has to perform an optimal minimum time trajectory so as to reach a circular goal region with radius $0.5\, m$ centred at $(6.5, 6.5)$, starting from the origin $(0, 0)$ with initial orientation $\pi/4$ and linear velocity $2\, m/s$. The vehicle control inputs v ad ω are subject to the following constraints: $v \in [0, 2]\, m/s$, and $\omega \in [0, 3]\, rad/s$. As for the solution to the singularity issue (see Remark 1), we set $v_{\min} = 0.1\, m/s$ and $\bar{a} = 2.5\, m/s^2$.
In order to minimize the control effort the quadratic cost (4) is adopted with R given by the identity matrix, so as to equally weight v and ω in the input vector $u = [v\, \omega]'$.

We first describe the database representing the motion primitives, and then present the results obtained when planning an optimal constrained trajectory from the origin to the target region by the proposed flat-RRT* algorithm. The impact on trajectory cost and computing time of different tree cardinalities will be analysed, as well.

4.1 Generation of the motion primitives

In order to build the database of motion primitives associated with the normalized initial and final configurations, $q_0 = (0, 0, 0, v_0)$ and $q_f = (x_f, y_f, \theta_f, v_f)$, we grid the boundary conditions as follows:

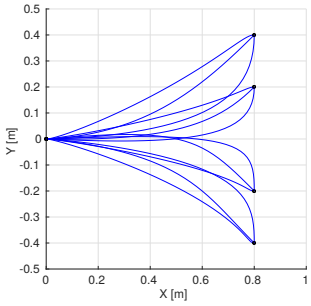

Fig. 1. A subset of the motion primitives in the database.

- $\dot{z}_1(0) = v_0$, linearly spaced vector with 15 values in the range $[0, 2]\, m/s$;
- $z_1(t_f) = x_f$, linearly spaced vector with 21 values in the range $[0, 1]\, m$;
- $z_2(t_f) = y_f$, linearly spaced vector with 21 values in the range $[-0.5, 0.5]\, m$;
- $\dot{z}_1(t_f) = v_f \cos(\theta_f)$, linearly spaced vector with 31 values in the range $[0, 2]\, m/s$;
- $\dot{z}_2(t_f) = v_f \sin(\theta_f)$, linearly spaced vector with 31 values in the range $[-2, 2]\, m/s$.

For each point in the grid a motion primitive is then generated by imposing the corresponding boundary conditions (7), and solving the optimization problem (8). In our implementation, the constrained optimization problem was solved in MATLAB® via the *fmincon* function, which finds a minimum of a constrained nonlinear multivariable function. Figure 1 shows some of the motion primitives generated to populate the database. All the reported primitives start from configuration $(x_0, y_0, \theta_0, v_0) = (0, 0, 0, 0)$ and have final velocity equal to 0, $\sqrt{2}$, 0.5. Those with final position characterized by a coordinate $y_f > 0$ have final orientation equal to 0, $\pi/4$ and $\pi/2$, whereas those with final position with $y_f < 0$ have, instead, final orientation equal to 0, $-\pi/4$ and $-\pi/2$.

*4.2 Planning trajectories with flat-RRT**

The flat-RRT* algorithm is tested in the virtual environment previously described, considering the following tree cardinalities: 1000, 2500, 5000, 7500, and 10000 nodes. Flat-RRT* was implemented in MATLAB® R2015b and run on a laptop with an Intel® i7-4702MQ 2.2GHz CPU. Figure 2 shows the whole tree that is built in a run of the flat-RRT* algorithm with the tree cardinality set to 1000. Figure 3 shows the resulting trajectories obtained for the different cardinality values, including also the case of "first-arrival", corresponding to the trajectory planned using a tree whose expansion is stopped the first time a node within the goal region is added to the tree. The time needed to reach the goal set decreases as the cardinality increases, as can be seen in Figure 4 where the average time to reach the goal over 50 runs of the flat-RRT* algorithm is plotted together with the minimum and maximum times. Finally, Figure 5 shows the average, minimum and maximum computation times required to generate a trajectory as a function of the tree cardinalities. Less than a minute is needed to determine a trajectory when a tree cardinality $N = 10000$ is used. This is quite an encouraging result for a possible online usage of the planner.

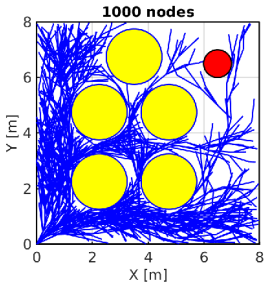

Fig. 2. Tree generated in a run with the tree cardinality set to 1000.

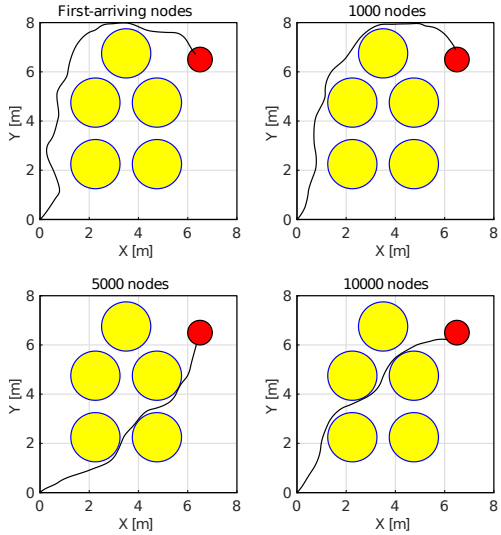

Fig. 3. Trajectories generated with different tree cardinalities and adopting the "first-arrival" stop criterion.

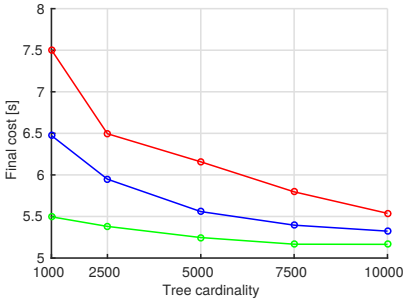

Fig. 4. Average (blue), minimum (green), maximum (red) time to reach the goal region as a function of the tree cardinality.

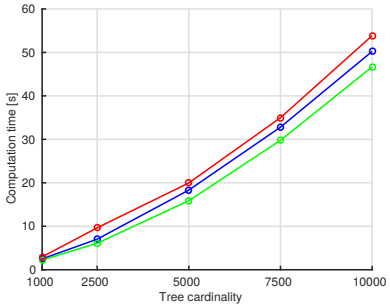

Fig. 5. Average (blue), minimum (green), maximum (red) computation time as a function of the tree cardinality.

5. CONCLUSIONS

This paper addresses optimal trajectory planning in presence of kino-dynamic constraints for a vehicle with differentially flat dynamics. We propose a novel RRT* algorithm that exploits pre-computed motion primitives in the flat output representation to accelerate the construction of the edges, while building the tree from randomly extracted nodes in the configuration space. Our goal is to allow for the online usage of RRT* when the vehicle is moving in a partially known environment. This task is simplified by the fact that in RRT* edges are checked against the presence of obstacles only afterwards, when building the tree. Our solution rests on a database of a few pre-computed edges (the motion primitives) built based on a gridding of the configuration space. These motion primitives are then used as tentative solutions to derive the (sub-optimal) edge joining the initial and final nodes while satisfying the actuation constraints. The finer the grid the better the performance, but the larger is the database.

REFERENCES

Buccieri, D., Perritaz, D., Mullhaupt, P., Jiang, Z.P., and Bonvin, D. (2009). Velocity-scheduling control for a unicycle mobile robot: Theory and experiments. *IEEE Transactions on Robotics*, 25(2), 451–458.

De Luca, A., Oriolo, G., and Vendittelli, M. (2001). Control of wheeled mobile robots: An experimental overview. *Lecture Notes in Control and Information Sciences*, 270, 181–226.

Fliess, M., Lévine, J., Martin, P., and Rouchon, P. (1995). Flatness and defect of non-linear systems: introductory theory and examples. *International Journal of Control*, 61(6), 1327–1361.

Fuchshumer, S., Schlacher, K., and Rittenschober, T. (2005). Nonlinear vehicle dynamics control - a flatness based approach. In *IEEE Conference on Decision and Control*, 6492–6497.

Karaman, S. and Frazzoli, E. (2010). Optimal kinodynamic motion planning using incremental sampling-based methods. In *IEEE Conference on Decision and Control*, 7681–7687.

Kavraki, E., Kolountzakis, M., and Latombe, J.C. (1998). Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1), 166–171.

Kavraki, L.E., Švestka, P., Latombe, J.C., and Overmars, M.H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4), 566–580.

LaValle, S. and Kuffner, J. (2001). Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5), 378–400.

Likhachev, M. and Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *International Journal of Robotics Research*, 28(8), 933–945.

Pivtoraiko, M., Knepper, R., and Kelly, A. (2009). Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26, 308–333.

Ragaglia, M., Prandini, M., and Bascetta, L. (2015). Poli-RRT*: Optimal RRT-based planning for constrained and feedback linearisable vehicle dynamics. In *European Control Conference*, 2521–2526.