Using motion primitives to enforce vehicle motion constraints in sampling-based optimal planners

Basak Sakcak, Luca Bascetta, Gianni Ferretti and Maria Prandini
Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza L. Da Vinci 32, 20133, Milano, Italy – e-mail: name.surname@polimi.it

***Abstract*—This paper presents a newly conceived planning algorithm that is based on the introduction of motion primitives in RRT*. Online computational complexity of RRT* is greatly reduced by pre-computing the optimal constrained trajectories joining pairs of starting and destination configurations in a grid space, while taking into account vehicle motion constraints in the planning task. A numerical example shows the effectiveness of the algorithm.**

I. Introduction

In the last decade the number of applications of autonomous aerial, underwater, and ground vehicles has been continuously increasing, fostering the research of the scientific community in autonomy, efficiency, reliability, and safety. As far as autonomy of a vehicle is concerned, planning, control, and perception play an important role in terms of safety and performance. Among them, planning [1] is particularly important, as it is responsible for computing an obstacle-free trajectory that takes the vehicle to the desired location, and also for accommodating for actuation and kinodynamic vehicle constraints, while minimizing the trajectory cost [2].

This paper proposes a novel sampling-based motion planner, which rests on RRT* (Rapidly exploring Random Tree star) [3], but alleviates its computational load substituting the online usage of the steering function with a search in a database of pre-computed motion primitives. This strategy allows to explicitly take into account the vehicle motion model in planning but, at the same time, is computationally affordable and amenable for application to a dynamic or partially known environment, where frequent re-planning is typically required. The database of motion primitives is built by considering a set of initial and final configuration pairs in some grid space, and determining for each pair an optimal trajectory that is compatible with the system dynamics and actuation constraints, while minimizing a cost. This formulation could be also extended to include further constraints like, for example, passenger comfort for automated passenger vehicles [4].
Following the RRT* method, a tree of candidate solutions is built by progressively adding nodes, which are extracted at random in the gridded configuration space and then connected to an existing node by an edge which is the best obstacle-free motion primitive in the database. To guarantee optimality, the tree is rewired if some nodes can be reached from the newly sampled configuration through an obstacle-free motion primitive with lower cost.
Thanks to the adoption of motion primitives, the computationally more intensive part that characterizes sampling-based planners is thus moved to the preliminary phase of the database construction, that can be executed off-line and only once.
Note that gridding causes some performance degradation. However, the grid resolution can be tuned so as to compromise between (sub)optimality and size of the database.

A simulation example referring to a simple outdoor map shows the effectiveness of the proposed algorithm, also in terms of computation time for online planning.

II. Problem statement

In this work, we consider dynamical systems with state $\mathbf{q} \in \mathbb{R}^d$ and control input $\mathbf{u} \in \mathbb{R}^m$ governed by the following differential equation

$$\dot{\mathbf{q}}(t) = f\left(\mathbf{q}(t), \mathbf{u}(t)\right). \tag{1}$$

The control input $\mathbf{u}(t)$ is subject to actuation constraints, and we denote the admissible control space as $U \subset \mathbb{R}^m$. The state $\mathbf{q}(t)$ is constrained in the set $Q \subset \mathbb{R}^d$, and initialized with $\mathbf{q}(0) = \mathbf{q}_0 \in Q$. Both U and Q are assumed to be compact. An open subset Q_{goal} of Q represents the goal region that the state has to reach.

A *trajectory* of system (1) is defined as the tuple $\mathbf{z} = (\mathbf{q}(\cdot), \mathbf{u}(\cdot), \tau)$, where τ is the duration of the trajectory, and $\mathbf{q}(\cdot) : [0, \tau] \to Q$ and $\mathbf{u}(\cdot) : [0, \tau] \to U$ represent the state and control input evolution in the time interval $[0, \tau]$ and satisfy the differential equation (1) for $t \in [0, \tau]$, the initial condition $\mathbf{q}(0) = \mathbf{q}_0$, and the final condition $\mathbf{q}(\tau) \in Q_{goal}$.

Obstacles are represented via an open subset Q_{obs} of Q. The free space is then defined as $Q_{free} := Q \setminus Q_{obs}$, and we assume that $\mathbf{q}_0 \in Q_{free}$.

A trajectory $\mathbf{z} = (\mathbf{q}(\cdot), \mathbf{u}(\cdot), \tau)$ of system (1) is said to be *collision free* if it avoids collisions with obstacles, i.e., $\mathbf{q}(t) \in Q_{free}$, for $t \in [0, \tau]$.
We denote the set of collision free trajectories as Z_{free}.

An *optimal kinodynamic motion planning problem* can then be formalized as finding a collision free trajectory $\mathbf{z}^\star = (\mathbf{q}^\star(\cdot), \mathbf{u}^\star(\cdot), \tau^\star) \in Z_{free}$ that is *optimal* according to a cost criterion $J(\mathbf{z}) : Z_{free} \to \mathbb{R}_{\geq 0}$ that is expressed as

$$J(\mathbf{z}) = \int_0^\tau g\left(\mathbf{q}(t), \mathbf{u}(t)\right) \mathrm{d}t \tag{2}$$

where $g : Q \times U \to \mathbb{R}_{\geq 0}$ is an instantaneous cost function.
Note that, if $g(\mathbf{q}, \mathbf{u}) = 1$ we are addressing a minimum time

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

optimization problem. Indeed, the trajectory duration τ is one of the optimization variables of the problem.

In the context of motion planning, system (1) represents the equations of motion of the vehicle and, consequently, its state vector $\mathbf{q}$ includes the vehicle position with respect to a given absolute reference frame.

In the following developments, we shall assume that the problem is translation invariant. This means that, if we neglect obstacles and consider a pair of initial and final configurations and the associated optimal trajectory $\mathbf{z}^\star = (\mathbf{q}^\star(\cdot), \mathbf{u}^\star(\cdot), \tau^\star)$, by shifting the origin of the coordinate system and rewriting all relevant quantities – including system dynamics (1), and initial and final configurations – in the new coordinate system and applying input $\mathbf{u}^\star(\cdot)$, we get the optimal vehicle trajectory, which is $\mathbf{z}^\star$ rewritten in the new coordinates.

III. RRT* WITH MOTION PRIMITIVES

The approach here proposed is based on previous works on search-based ([5], [6]), and sampling-based ([3], [7]) methods, and combines them in a novel way. In particular, it relies on a uniform discretization of the state space, and on the construction of a finite set of motion primitives by solving a constrained optimization problem with boundary conditions on the grid points of a smaller (uniform) grid. The motion primitives are then embedded in the RRT* algorithm, where they are used to connect the randomly generated nodes to the tree, thus eliminating the need of solving online challenging and time consuming Two Point Boundary Value Problems (TPBVPs).

A. Database of Motion Primitives

The database of motion primitives is built by gridding the continuous state space in order to obtain a finite set of boundary conditions (initial and final states), and by solving a constrained boundary value optimization problem for each pair. The resulting set of motion primitives is then used repeatedly online implementing a procedure that, when an edge connecting two nodes is requested, simply picks from the database a stored optimal trajectory.

Motion primitives are computed for each pair of initial and final states $(\bar{\mathbf{q}}_0, \bar{\mathbf{q}}_f)$ in a uniformly grid space by solving the following TPBVP:

$$\begin{aligned} \underset{\mathbf{u}(\cdot),\tau}{\text{mininimize}} \quad & \int_0^\tau g\left(\mathbf{q}(t), \mathbf{u}(t)\right) \mathrm{d}t \\ \text{subject to} \quad & \dot{\mathbf{q}}(t) = f\left(\mathbf{q}(t), \mathbf{u}(t)\right) \\ & \mathbf{u}(t) \in U,\, t \in [0, \tau] \\ & \mathbf{q}(t) \in Q_{free},\, t \in [0, \tau] \\ & \mathbf{q}(0) = \bar{\mathbf{q}}_0,\ \mathbf{q}(\tau) = \bar{\mathbf{q}}_f \end{aligned} \tag{3}$$

Note that state $\mathbf{q} \in Q$ includes the vehicle position $\boldsymbol{\pi}$, i.e, $\mathbf{q} = [\boldsymbol{\pi}^T, \dots]^T$.

Thanks to the translation invariance property, the size of the database can be kept small while covering a wide range of the space where the vehicle is moving. Indeed, one can, without loss of generality, set the initial position $\boldsymbol{\pi}(0)$ to $\boldsymbol{\pi}(0) = \bar{\boldsymbol{\pi}}_0 = 0$ when building the database, and recover the optimal trajectory for an arbitrary initial position $\boldsymbol{\pi}_0$ by simply centering the coordinate system in $\boldsymbol{\pi}_0$. We shall denote the set of grid points adopted for the database construction by `BoundingBox`$(\bar{\boldsymbol{\pi}}_0)$ to recall that it is built considering as initial state $\bar{\mathbf{q}}_0 = [\bar{\boldsymbol{\pi}}_0^T, \dots]^T$, with position $\bar{\boldsymbol{\pi}}_0 = 0$ being the origin of the coordinate system.

In the following, we shall refer to the set of trajectories in the database computed using distinct pairs of initial and final states $(\bar{\mathbf{q}}_0^i, \bar{\mathbf{q}}_f^i)$, $i = 1, \dots, N_{\mathbb{Z}}$, as the collection of motion primitives $\mathbb{Z} = \{\mathbf{z}_i^\star = (\mathbf{q}_i^\star(t), \mathbf{u}_i^\star(t), \tau_i^\star),\, i = 1, \dots, N_{\mathbb{Z}}\}$. Correspondingly, $\mathbb{C} = \{C_i^\star,\, i = 1, \dots, N_{\mathbb{Z}}\}$ denotes the set of costs associated to the motion primitives.

B. Motion Planning

We now introduce `RRT*_MotionPrimitives`, which is the proposed variant of RRT* integrating the database of motion primitives for the computation of a collision free optimal trajectory.

As in RRT*, `RRT*_MotionPrimitives` is based on the construction of a random tree $T = (Q_T, E_T)$, where $Q_T \subset Q$ is the set of *nodes*, and E_T is the set of *edges*. Nodes are states and edges are optimal trajectories, each one connecting a pair of origin and destination nodes and solving the TPBVP in (3). The tree $T = (Q_T, E_T)$ is expanded for a maximum number of iterations N, defined by the user, starting from $Q_T = \{\mathbf{q}_0\}$, where $\mathbf{q}_0$ is the initial state, and $E_T = \emptyset$, as described in Algorithm 1. Every node $\mathbf{q} \in Q_T$ is connected to $\mathbf{q}_0$ via a single sequence of intermediate nodes $\mathbf{q}_i \in Q_T$, $i = 1, \dots, n-1$, $n \leq N$, and associated edges $e_i = e_{\mathbf{q}_i, \mathbf{q}_{i+1}} \in E_T$, $i = 0, 1, \dots, n-1$, with $\mathbf{q}_n = \mathbf{q}$. We can then associate to $\mathbf{q}_n$ a cost $C(\rightarrow \mathbf{q}_n)$

$$C(\rightarrow \mathbf{q}_n) = \sum_{i=0}^{n-1} C(e_i)$$

where $C(e_i)$ denotes the cost associated with edge $e_i \in E_T$ and computed as in (2).

Tree growing is based on four main steps: random sampling, finding near nodes, extending the tree, and rewiring.

The first step (`SAMPLE` in Algorithm 1) consists in randomly sampling a state $\mathbf{q}_{rand}$ from a uniform grid in the free state space Q_{free} according to a uniform distribution. The other ones (`NEAR_NODES`, `EXTEND`, and `REWIRE` in Algorithm 1) are illustrated in Algorithms 2, 3, and 4, respectively, and the main differences with respect to RRT* are explained in the following.

In kinodynamic RRT* [7] the edge computation is usually based on the minimization of a non-Euclidean metric, like the optimal cost (2). In this case, near nodes to a given node $\mathbf{q}_{rand}$ are selected from a set of reachable states, Q_{reach}, defined as follows

$$\begin{aligned} Q_{reach} = \{\mathbf{q} \in Q : C(e_{\mathbf{q}_{rand}, \mathbf{q}}) \leq l(n) \vee \\ C(e_{\mathbf{q}, \mathbf{q}_{rand}}) \leq l(n)\} \end{aligned}$$

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

Algorithm 1: RRT*_MotionPrimitives

$Q_T \leftarrow \{\mathbf{q}_0\}$, $E_T \leftarrow \emptyset$, $n \leftarrow 1$
while $n \leq N$ **do**
  $\mathbf{q}_{\text{rand}} \leftarrow \texttt{SAMPLE}(Q_{\text{free}})$
  $Q_{\text{near}} \leftarrow \texttt{NEAR\_NODES}(\mathbf{q}_{\text{rand}})$
  $\mathbf{q}_{\text{best}} \leftarrow \texttt{EXTEND}(Q_{\text{near}}, \mathbf{q}_{\text{rand}})$
  if $\mathbf{q}_{\text{best}} \neq \emptyset$ **then**
    $Q_T \leftarrow Q_T \cup \{\mathbf{q}_{\text{rand}}\}$
    $E_T \leftarrow E_T \cup \{(\mathbf{q}_{\text{best}}, \mathbf{q}_{\text{rand}})\}$
    $n \leftarrow n + 1$
    $E_T \leftarrow \texttt{REWIRE}(Q_T, E_T, \mathbf{q}_{\text{rand}}, Q_{\text{near}})$
return Q_T, E_T

Algorithm 2: NEAR_NODES

$Q_{\text{near}} \leftarrow \emptyset$
$\boldsymbol{\pi}_{rand} \leftarrow \texttt{GetPosition}(\mathbf{q}_{rand})$
for $\forall \mathbf{q} \in Q_T$ **do**
  if $\mathbf{q} \in \texttt{BoundingBox}(\boldsymbol{\pi}_{rand})$ **then**
    if $C(e_{\mathbf{q}_{rand},\mathbf{q}}) \leq l(n) \vee C(e_{\mathbf{q},\mathbf{q}_{rand}}) \leq l(n)$ **then**
      $Q_{\text{near}} \leftarrow Q_{\text{near}} \cup \{\mathbf{q}\}$
return Q_{near}

where $l(n)$ is a cost threshold that decreases over the iterations of the algorithm as $l(n) = \gamma_l (\log n/n)$, such that a ball of volume $\gamma^d (\log n/n)$ is contained within Q_{reach}, where γ_l and γ are suitable constants [7].
In RRT*_MotionPrimitives, however, the set Q_{reach} has to be further constrained to ensure that there is a pre-computed trajectory in the database for each connection in the set of near nodes. The set of near nodes is then redefined as follows

$$Q_{near} = Q_{reach} \cap \texttt{BoundingBox}(\boldsymbol{\pi}_{rand})$$

where $\texttt{BoundingBox}(\boldsymbol{\pi}_{rand})$ denotes the box of grid points in the state space Q adopted for the database construction, with the origin of the reference coordinate system shifted from $\bar{\pi}_0 = 0$ to π_{rand}, which is the vehicle position associated to state $\mathbf{q}_{rand}$ and obtained using GetPosition.
If Q_{near} occurs to be an empty set, the algorithm continues to the next iteration selecting a new $\mathbf{q}_{rand}$.

The EXTEND procedure connects $\mathbf{q}_{rand}$ to the tree using a minimum cost collision free trajectory extracted from the database (see Algorithm 3). Obstacle checking is performed, as in RRT*, by the $\texttt{CollisionFree} : E_T \rightarrow \{0, 1\}$ function, that returns 1 for an edge that is collision free and 0 otherwise. On the other side, the selection of an edge from the database, connecting $\mathbf{q}$ to $\mathbf{q}_{rand}$, is a peculiarity of RRT*_MotionPrimitives, and is performed by the FindTrajectory function as follows:

1) a translation is applied to the pair of initial and final states $(\mathbf{q}, \mathbf{q}_{rand})$ (Figure 1(a)), obtaining the normalized pair $(\tilde{\mathbf{q}}, \tilde{\mathbf{q}}_{rand})$, such that the resulting $\tilde{\mathbf{q}}$ has the position $\tilde{\boldsymbol{\pi}}$ corresponding to the null vector (Figure 1(b));
2) a query is executed on the database to look for the trajectory $\mathbf{z}_i \in \mathbb{Z}$ and the cost $C(e_{\tilde{\mathbf{q}},\tilde{\mathbf{q}}_{rand}}) \in \mathbb{R}$;
3) the inverse of the previous translation is applied in order to recover the trajectory connecting the actual pair of boundary values $(\mathbf{q}, \mathbf{q}_{rand})$, determining the edge $e_{\mathbf{q},\mathbf{q}_{rand}}$ (Figure 1(c)).

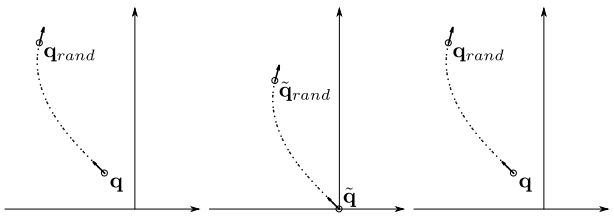
Algorithm 3: EXTEND

$\mathbf{q}_{\text{best}} \leftarrow \emptyset, e_{\text{best}} \leftarrow \emptyset, \mathtt{c}_{\text{best}} \leftarrow \infty$
for $\mathbf{q} \in Q_{\text{near}}$ **do**
  $z, C(e_{\mathbf{q},\mathbf{q}_{\text{rand}}}) \leftarrow \texttt{FindTrajectory}(\mathbf{q}, \mathbf{q}_{\text{rand}})$
  if $C(e_{\mathbf{q},\mathbf{q}_{\text{rand}}}) < c_{\text{best}}$ **then**
    if $\texttt{CollisionFree}(\mathtt{e}_{\mathbf{q},\mathbf{q}_{\text{rand}}})$ **then**
      $\mathbf{q}_{\text{best}} \leftarrow \mathbf{q}$, $e_{\text{best}} \leftarrow z$, $c_{\text{best}} \leftarrow C(e_{\mathbf{q},\mathbf{q}_{\text{rand}}})$
return $\mathbf{q}_{\text{best}}$

Finally, the REWIRE procedure (Algorithm 4) adopted in RRT*_MotionPrimitives is the same used by RRT*. If a new edge has to be generated, however, it is extracted from the database using the FindTrajectory function.

Algorithm 4: REWIRE

for $\mathbf{q} \in Q_{\text{near}}$ **do**
  $z, C(e_{\mathbf{q}_{\text{rand}},\mathbf{q}}) \leftarrow \texttt{FindTrajectory}(\mathbf{q}_{\text{rand}}, \mathbf{q})$
  if $C(\rightarrow \mathbf{q}_{rand}) + C(e_{\mathbf{q}_{\text{rand}},\mathbf{q}}) < C(\rightarrow \mathbf{q})$ **then**
    if $\texttt{CollisionFree}(\mathtt{e}_{\mathbf{q},\mathbf{q}_{\text{rand}}})$ **then**
      $\mathbf{q}_{parent} \leftarrow \texttt{Parent}(\mathbf{q})$
      $e_{prev} \leftarrow e_{\mathbf{q}_{parent},\mathbf{q}}$
      $e \leftarrow e_{\mathbf{q}_{rand},\mathbf{q}}$
      $E_T = \{E_T \setminus e_{prev}\} \cup \{e\}$
return E_T



(a) original query (b) translation to origin (c) inverse transformation and edge design

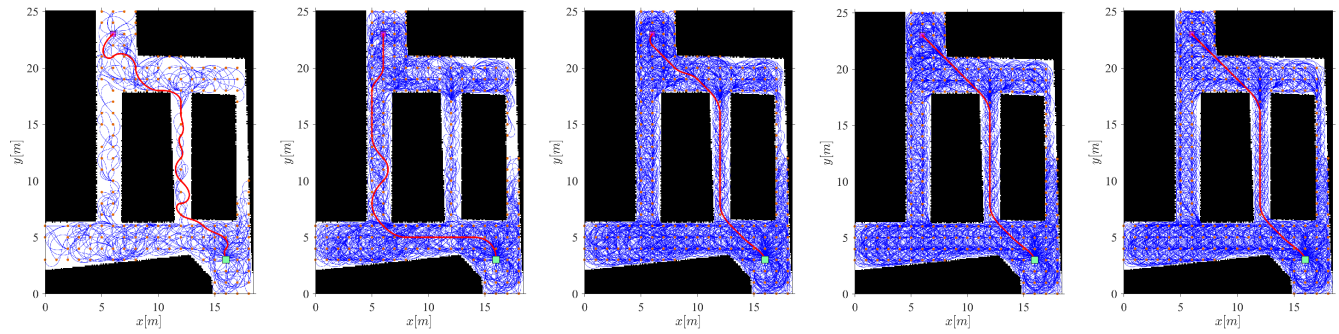
Fig. 1. Steps involved in the coordinate transformation during the FindTrajectory procedure.

IV. RESULTS

We now present a numerical example, which is based on the dynamic model of a unicycle vehicle. The vehicle motion equations are given by

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = w, \quad \dot{v} = a \tag{4}$$

978-1-5386-4881-0/18/$31.00 ©2018 IEEE



(a) 1000 iterations (b) 5000 iterations (c) 10000 iterations (d) 15000 iterations (e) 30000 iterations

Fig. 2. Trees (blue solid lines) and optimal trajectories (red solid lines) computed for different number of iterations. The initial state is marked with a light green square, the goal region corresponds to the magenta area.

where (x, y, θ) is the pose of the vehicle, v and w are the linear and angular velocity, respectively, and a is the linear acceleration. They are bounded as $v \in [0, 4]$ m/s, $w \in [-5, 5]$ rad/s and $a \in [-3, 3]$ m/s^2.

The motion primitives in the database are computed for each pair of initial and final state, $\mathbf{q}_0 = [x_0, y_0, \theta_0, v_0]$ and $\mathbf{q}_f = [x_f, y_f, \theta_f, v_f]$, solving the TPBVP in (3) for the differential equations given in (4) and the cost function

$$J(\mathbf{z}) = \int_0^{\tau} \left[1 + \mathbf{u}(t)^T R \mathbf{u}(t)\right] \mathrm{d}t$$

where $\mathbf{u} = [\omega, a]^T$ is the control input. This cost function penalizes the total time of the trajectory, τ, and the total actuation effort with a weight $R = 0.5 I_2$. The database is determined by considering $(x_0, y_0) = (0, 0)$ and making the final position $(x_f, y_f) \neq (0, 0)$ range within $[-2, 2]^2$ with an uniform gridding in square cells of size one meter. The initial orientation θ_0 is selected among three values $\{0, \pi/4, \pi/2\}$ rad, the final orientation θ_f can take nine values in the range $[0, 2\pi)$ rad. For the initial and final velocities, v_0 and v_f, a minimum and a maximum velocity of 1 m/s and 4 m/s is considered.

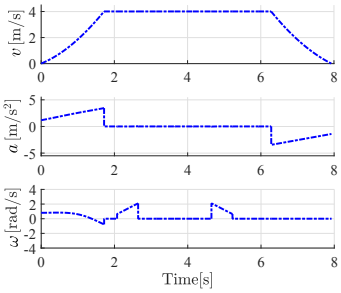
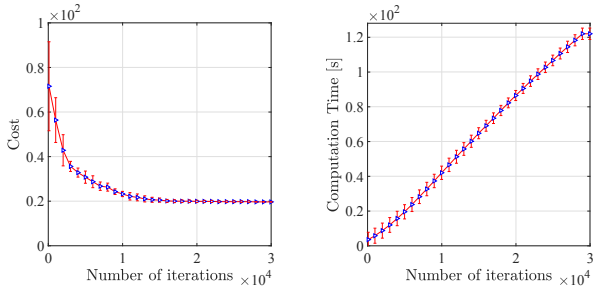


Fig. 3. Linear velocity and actuation profiles for the minimum cost trajectory.

Simulations are performed on an Intel Core i7@2.40 GHz personal computer with 8Gb RAM and the algorithm has been implemented in MATLAB. An outdoor map is considered, setting the vehicle initial pose at $(16, 3, \pi/2)$, with zero velocity. The goal area is defined as a square with side $0.5m$ and centred at $(6, 23)$. The vehicle should stop at the end of the trajectory.



(a) cost (b) computation time

Fig. 4. Cost and computation time with respect to the number of iterations. Blue triangles represent the average among the simulation runs.

Figure 2 shows the trees and the optimal trajectories for different number of iterations. Figure 3 shows the velocity and the actuation profiles for the minimum cost trajectory computed with 30000 iterations.

The same planning problem has been solved for 20 independent simulation runs. Figure 4(a) shows the average cost evolution and its standard deviation as the number of iterations increases. As expected, the cost reduces increasing the number of iterations, and converges to the same value for all the simulation runs: once this minimum is achieved the solution will not further improve. Finally, Figure 4(b) shows the computation time evolution as the number of iterations increases. Note that the minimum cost trajectory for the assigned resolution grid is determined after 15000 iterations, and computation time is approximately 1 minute.

V. Conclusion

This paper introduces `RRT*_MotionPrimitives`, an extension of RRT* that allows to account for vehicle motion constraints when planning an optimal trajectory, alleviating the online computational effort by using pre-computed motion primitives. A simulation example shows the effectiveness of the proposed algorithm. Code optimization and implementation in C or C++ are considered for further speed up.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

REFERENCES

[1] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[2] D. J. Webb and J. Van Den Berg, "Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints," *arXiv preprint arXiv:1205.5088*, 2012.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[4] L. Labakhua, U. Nunes, R. Rodrigues, and F. S. Leite, "Smooth trajectory planning for fully automated passengers vehicles: Spline and clothoid based methods and its simulation," in *Informatics in Control Automation and Robotics*. Springer, 2008, pp. 169–182.

[5] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[6] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[7] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *IEEE Conference on Decision and Control*. IEEE, 2010, pp. 7681–7687.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE