Interactive Car Parking Simulation Based on On-line Trajectory Optimization

Jungsub Lim, Hyejin Kim, and Daseong Han(✉)

School of Global Entrepreneurship and ICT, Handong Global University, Pohang, Republic of Korea
jsrimr@naver.com, gobetty20@gmail.com, dshan@handong.edu

Abstract. This paper presents an on-line trajectory optimization method to simulate the autonomous parking of a car-like vehicle in an environment with static or dynamic obstacles. We employ a stochastic and derivative-free optimization technique called Covariance Matrix Adaptation (CMA) to seamlessly integrate collision events between the car and the environment into the formulation of our autonomous parking problem without resorting to any preprocessing steps to make the problem differentiable. Given the current and target car states, our system repeatedly predicts a sequence of control inputs for a short time window to move the car to the target while shifting the window along the time axis, which facilitates on-line performance. We also present a simple and effective scheme to make our optimization robust to environmental changes by adjusting its parameters in an on-line manner. We show the effectiveness of our method through simulation results for garage parking, parallel parking, and interactive parking based on on-line user input.

Keywords: Autonomous car parking · On-line trajectory optimization
Simulation · Motion planning · Stochastic optimization

1 Introduction

Over the last few years, autonomous driving has been one of the most important topics in many research areas due to the ever-growing demands and market scale for autonomous vehicles. Many technical innovations have been presented to effectively address the topic and commercial autonomous cars have already been being in market in several countries. However, it is still very challenging to robustly control an autonomous vehicle in urban environments, where it must recognize and obey traffic lights and also needs to avoid colliding with other moving vehicles and pedestrians whose motions are not fully predictable in general while moving to the destination. Instead of dealing with the topic as a whole, we focus on how to autonomously park a

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-319-91806-8_21) contains supplementary material, which is available to authorized users.

A. Marcus and W. Wang (Eds.): DUXU 2018, LNCS 10920, pp. 270–284, 2018.
https://doi.org/10.1007/978-3-319-91806-8_21

vehicle while avoiding obstacles in a computer simulation through numerical optimization with simple and intuitive objectives given on-line user input such as a desired parking location and orientation. Although the car parking is much simplified version of the autonomous driving problem, it can be very useful not only for attaining the goal of the latter but also for equipping conventional vehicles with an intelligent parking system.

In this paper, we present an on-line trajectory optimization method for autonomously moving a car to a desired parking slot based on a stochastic and derivative-free optimization method called covariance matrix adaptation (CMA) [1]. CMA repeatedly alternates evaluating a given objective function at possible solutions populated by a multivariate normal distribution and updating the distribution with the evaluation results until a certain termination condition is satisfied. Since the method does not rely on any derivative information of the objective function, an objective for collision avoidance from environmental obstacles, which introduces discontinuity into optimization in general, can be seamlessly integrated into a problem formulation together with the other objectives. It effectively removes precomputation steps to enforce the differentiability of objectives unlike existing derivative-based frameworks.

In order to facilitate on-line performance, we also employ a model predictive control (MPC) scheme, which repeatedly solves a finite-horizon trajectory optimization problem for a relatively short time window while shifting the window along the time axis. Since our MPC framework continuously re-computes the optimal trajectory in run time, environmental changes can be more effectively dealt with on the fly. The result of MPC is a sequence of optimal control vectors for the time window, each of which is composed of the front wheel angle and acceleration of the car. In turn, these control vectors are provided to the simulator to update the current car state.

The contributions of this paper are two-fold. In the systematic point of view, we present an on-line control framework which can produce realistic parking movements in an environment with static or dynamic obstacles according to a desired location and orientation interactively given by the user. In the technical point of view, we formulate an autonomous car-parking problem with simple and intuitive high-level objectives based on a stochastic and derivative-free optimization technique. Our method does not rely on any complicated system analysis or control law design. We also introduce an MPC scheme to effectively support on-line performance and to enhance time efficiency. This is due to the fact that MPC does not compute the whole trajectory to reach a final destination at once but repeatedly optimizes a trajectory for a short time window while moving it along the time axis. Our method is effective to generate convincing parking movements for challenging scenarios such as moving a car into a narrow parking slot while avoiding a moving obstacle without any reference trajectory or preprocessing.

2 Related Work

For the autonomous parking problem, various approaches have been proposed over the last decades. Control design and analysis for wheeled mobile robots have been actively done by transforming their kinematic equations to so-called chained form [2]. Reeds and Shepp simplified car movements to a combination of circular and linear motions,

and then calculated the minimum-length path [3], which has also been adopted by many other researchers [4, 5]. However, this approach works only in an obstacle-free environment.

Laumond et al. proposed a motion planner for mobile vehicles based on a two-step approach [6]. In the first step, the planner computes a collision-free holonomic path, which is relatively easy to derive. The path is then transformed into the corresponding non-holonomic one by integrating non-holonomic constraints. Muller et al. also employed this approach using the clothoid curve [7]. However, it is difficult to derive the non-holonomic path corresponding to a holonomic in general, which often results in a lot of maneuvering efforts. In order to avoid this complexity, our problem formulation relies on simple high-level objectives supported by derivative-free optimization.

Kim et al. derived collision-free reachable areas for every movement of a vehicle, and then connected points in these areas to generate multiple path candidates [8]. This method is effective to deal with environmental changes, such as the change of an obstacle position or parking space. But it cannot deal with moving obstacles. Unlike this, our method can avoid moving obstacles as well as static.

Barraquand and Latombe discretized a configuration space into a grid map and then planned a path on the map by exhaustive search equipped with heuristic rules [9]. The scheme provides practical solutions in many cases. However, the method's accuracy for generating a collision-free path is not guaranteed and it requires high computing power.

Sampling-based approaches have successfully applied to path planning. LaValle and Kuffner proposed Rapidly-exploring Random Trees (RRT) [10], which generates a tree structure by connecting the randomly sampled nodes satisfying certain constraints and conditions until the tree includes a target position. Its effectiveness was also demonstrated in the 2007 DARPA Urban Challenge [11, 12]. Probabilistic roadmap (PRM) was proposed by Kavraki et al. to plan a path in high-dimensional configuration spaces [13]. PRM planners allocate samples in a configuration space and construct a path graph connecting these sample points.

A two-phase path planning approach was proposed for autonomous vehicles by Dolgov et al. [14], which was also experimentally validated in the same competition mentioned above [12]. In the first phase, A* algorithm is used to search a kinematically feasible trajectory. Then, a derivative-based optimization is employed to improve the trajectory in the second phase. In order to keep the objective function differentiable in this phase, nearest obstacles at each time step must be precomputed and fixed in the previous phase. Unlike this approach, we propose a single-phase path planning framework that seamlessly integrates an obstacle avoidance objective into optimization without any precomputation steps.

There have been also efforts to apply artificial intelligence techniques such as fuzzy controllers [15] or neural networks [16] to motion planning problems with wheeled mobile robots. These are relatively easy to implement but the performance is highly dependent on their training data set.

Zips et al. suggested a fast motion planning algorithm for a wheeled vehicle based on heuristic rules, where the next move is determined by taking into account only a resulting state of the vehicle followed by the move but not the total path [17]. However, since the algorithm is rule-based, it is hard to be generalized. For example, an algorithm

for parallel parking on the left side cannot be applied to that for the right side. Also, it does not work in a dynamic environment.

Tassa et al. proposed a variant of differential dynamic programming (DDP) to support bound constraints on control variables [18] unlike the original version which can deal with only unconstrained variables. The method was effectively applied to the car parking problem in an obstacle-free environment so as to constrain the wheel angle and acceleration to their respective valid ranges. In more complicated environments with obstacles, however, it would be difficult to integrate the constraints for collision avoidance into an objective function because these are generally not differentiable while DDP requires a differentiable objective function. In order to address this issue, we employ a sampling-based optimization method called CMA [1]. It does not require any derivative information, which allows us to enforce collision avoidance constraints to an objective function in an intuitive manner.

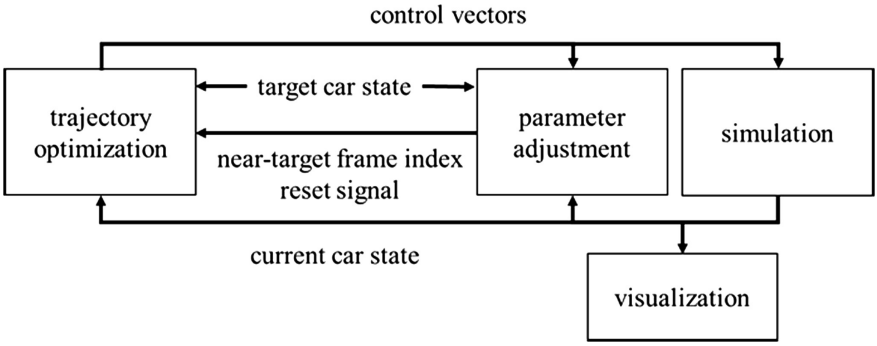

Fig. 1. System overview.

3 System Overview

Our system consists of three main components: trajectory optimization, parameter adjustment, and simulation (Fig. 1). Provided with the current and target car states and optimization parameters such as the near-target frame index and reset signal, the first component repeatedly solves a finite-horizon trajectory optimization problem for a short time window with CMA while shifting it along the time axis. It then produces a sequence of control vectors to make the car closer to its target state composed of the car position, speed, and orientation. In order to save computational time, the optimization starts with all the context data of the last optimization including the mean and covariance of a normal distribution maintained by CMA.

The second component computes the near-target frame index and reset signal to make trajectory optimization produce a better result and performance given the current and target car states together with the control vectors. The former represents an index to the first of the frames at which the car state is close enough to the target state while it is also true at all the following frames. It is used to make an objective function encourage

optimization to quickly converge to the target car state once the car comes near it. If the latter is on, the trajectory optimization is completely restarted without relying on any context data of the last optimization. It is often needed because CMA may have got stuck at a local optimum depending on the sampling data populated from its last normal distribution.

When the resulting control vectors are given to the last component, they are used to update the current car state. In the remaining sections, we do not deal with the simulation component in details but focuses on the first two components because the third component can be implemented by the system dynamics of the car introduced in trajectory optimization (Sect. 4.1).

4 Trajectory Optimization

In this section, we first discuss the simplified system dynamics of a car (Sect. 4.1). Then, we formulate a finite-horizon trajectory optimization problem for parking the car in a multi-obstacle environment based on its system dynamics (Sect. 4.2). Last, we explain how to solve the problem with CMA (Sect. 4.3).

4.1 System Dynamics

Though the actual system dynamics of a car is highly complicated, we employ its simplified version introduced in [18] because it is sufficient for producing realistic car movements. In order to formulate the system dynamics of a car, let us first define state vector $\mathbf{x} \in \mathbb{R}^4$ and constrained control vector $\mathbf{u}^c \in \mathbb{R}^2$ as follows:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \\ v \end{bmatrix} \text{ and } \mathbf{u}^c = \begin{bmatrix} \omega \\ \alpha \end{bmatrix}. \tag{1}$$

$$\omega^{lo} \leq \omega \leq \omega^{hi}, \tag{2}$$

$$\alpha^{lo} \leq \alpha \leq \alpha^{hi}. \tag{3}$$

Here, (x, y) is the midpoint between the back wheels. θ is the angle of the car with respect to the x-axis and v is the speed of the front wheels. ω and α are the angle and acceleration of the front wheels whose valid ranges are constrained by Eqs. (2) and (3), respectively. In all our experiments, we set $\omega^{lo} = -50^{\circ}$, $\omega^{lo} = 50^{\circ}$, $\alpha^{lo} = -15\,\text{m/s}^2$, and $\alpha^{hi} = 15\,\text{m/s}^2$. Then, system dynamics $\mathbf{x}' = \mathbf{f}^c(\mathbf{x}, \mathbf{u}^c)$, which returns next-step state $\mathbf{x}'$ given current state $\mathbf{x}$ and control $\mathbf{u}^c$, is formulated as follows:

$$\mathbf{f}^{\mathrm{c}}(\mathbf{x}, \mathbf{u}^{\mathrm{c}}) = \begin{bmatrix} x + b \cdot \cos(\theta) \\ y + b \cdot \sin(\theta) \\ \theta + \sin^{-1}\left(\sin(\omega)\frac{f}{d}\right) \\ v + h \cdot \alpha \end{bmatrix}, \tag{4}$$

where d is the distance between the front and back axles and h is the integration step size. f and b are the rolling distance of the front and back wheels, respectively, as shown below:

$$f = h \cdot v, \tag{5}$$

$$b = f \cdot \cos(\omega) + d - \sqrt{d^2 - f^2 \sin^2(\omega)}. \tag{6}$$

We refer readers to [18] for more details on the above car dynamics.

Since CMA is an unconstrained optimization method, we introduce new control vector $\mathbf{u} = [u_1 \quad u_2]^{\mathrm{T}}$ which is used to express ω and θ as functions of unconstrained variables u_1 and u_2, respectively:

$$\omega(u_1) = \omega^{lo} + \frac{(\omega^{hi} - \omega^{lo})}{2}\left(1 - \cos\left(\pi \frac{u_1 - \omega^{lo}}{\omega^{hi} - \omega^{lo}}\right)\right), \tag{7}$$

$$\alpha(u_2) = \alpha^{lo} + \frac{\alpha^{hi} - \alpha^{lo}}{2}\left(1 - \cos\left(\pi \frac{u_2 - \alpha^{lo}}{\alpha^{hi} - \alpha^{lo}}\right)\right). \tag{8}$$

Here, bound constraints $\omega(u_1) \in [\omega^{lo}, \omega^{hi}]$ and $\alpha(u_2) \in [\alpha^{lo}, \alpha^{hi}]$ are always satisfied no matter what values control variables u_1 and u_2 have. With the new control vector $\mathbf{u}$, we define the unconstrained version of system dynamics $\mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{u})$ as follows:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} x + b \cdot \cos(\theta) \\ y + b \cdot \sin(\theta) \\ \theta + \sin^{-1}\left(\sin(\omega(u_1))\frac{f}{d}\right) \\ v + h \cdot \alpha(u_2) \end{bmatrix}, \quad \text{where} \tag{9}$$

$$f = h \cdot v, \tag{10}$$

$$b = f \cdot \cos(\omega(u_1)) + d - \sqrt{d^2 - f^2 \sin^2(\omega(u_1))}. \tag{11}$$

4.2 Problem Formulation

Based on the system dynamics in Eq. (9), our system optimizes state trajectory $\mathrm{X} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$ for a short time window of N frames to drive the autonomous car to

reach target state $\bar{\mathbf{x}} = \begin{bmatrix} \bar{x} & \bar{y} & \bar{\theta} & \bar{v} \end{bmatrix}^{\mathrm{T}}$ composed of target position $(\bar{x}, \bar{y})$, target orientation $\bar{\theta}$, and target speed $\bar{v}$. In order to do so, we formulate a finite-horizon trajectory optimization problem to find control vector sequence $\mathrm{U} = \{\mathbf{u}^1, \mathbf{u}^2, \ldots, \mathbf{u}^{N-1}\}$ given current car state $\mathbf{x}$ and target car state $\bar{\mathbf{x}}$ together with near-target frame index k and reset signal η:

$$\min_U \mathrm{J}(X, U) \tag{12}$$

$$\text{subject to} \quad \mathbf{x}^1 = \mathbf{x}, \tag{13}$$

$$\mathbf{x}^{i+1} = \mathbf{f}(\mathbf{x}^i, \mathbf{u}^i), \tag{14}$$

$$i = 1, 2, \ldots, N-1.$$

Here, $\mathbf{x}^i$ and $\mathbf{u}^i$ are the state and control vectors of the car at frame i, respectively. $\mathrm{J}(X, U)$ is the objective function that specifies high-level objectives to perform car-parking tasks using state trajectory X and control vector sequence U. The first two constraints in Eqs. (13) and (14) represent that state trajectory X starts from the current state and the state transition in the trajectory can be made only by the system dynamics.

Objective function $\mathrm{J}(X, U)$ is formulated as the sum of the following cost terms:

$$\mathrm{J}(X, U) = c^{dir} + c^{tgt} + c^{col} + c^{eng} + c^{len}. \tag{15}$$

The first cost term c^{dir} is used to drive the car to have the same orientation with the target orientation before it gets close to the target state by taking into account differences between target orientation $\bar{\theta}$ and orientation θ^i at frame $i \in [1, k-1]$:

$$c^{dir} = h \cdot w^{dir} \cdot \sum_{i=1}^{k-1} \mathrm{sabs}(\bar{\theta} - \theta^i). \tag{16}$$

Here, w^{dir} is the weight constant and $\mathrm{sabs}(x)$ is the smoothed version of absolute function $|x|$ as follows [20]:

$$\mathrm{sabs}(x) = \sqrt{x^2 + \gamma^2} - \gamma. \tag{17}$$

In Eq. (17), γ is the coefficient used to adjust the smoothness around at $x = 0$. As γ becomes larger, the function is more smoothed. We set $\gamma = 1.0$ in all our experiments. As pointed out in [20], we adopt $\mathrm{sabs}(x)$ instead of a commonly-used quadratic cost term because the former increases almost linearly as x gets farther from zero. It prevents the optimization from chasing after the target too hard when the car state is currently quite far from the target, which results in more natural car movements. Cost term c^{dir} plays an important role especially when the car's initial orientation is opposite to its corresponding target so as to align them before moving to the target position.

The second cost term c^{tgt} is defined as a weighted sum of differences between target car state $\bar{\mathbf{x}}$ and car state $\mathbf{x}^i$ at each frame $i \in [k, N]$ to reach the target car state after aligning the car orientation with its target as shown below:

$$c^{tgt} = h \cdot (\boldsymbol{w}^{tgt})^{\mathrm{T}} \sum_{i=k}^{N} \begin{bmatrix} \mathrm{sabs}(\bar{x} - x^i) \\ \mathrm{sabs}(\bar{y} - y^i) \\ \mathrm{sabs}(\bar{\theta} - \theta^i) \\ \mathrm{sabs}(\bar{v} - v^i) \end{bmatrix}, \tag{18}$$

where $\boldsymbol{w}^{tgt} \in \mathbb{R}^4$ is the weight vector. Cost term c^{tgt} encourages the optimization to converge to the target car state as quickly as possible starting from frame k once the car has come near the target car state at frame k.

The third cost term c^{col} is used to avoid collisions between the car and any of environmental objects:

$$c^{col} = h \cdot w^{col} \cdot \sum_{i=1}^{N} \sum_{j=1}^{n_c^i} \phi\left(i, j, \mathbf{x}^i\right). \tag{19}$$

In the above equation, w^{col} is the weight constant. n_c^i is the number of contact points between the car and the environment at frame i and $\phi(i, j, \mathbf{x}^i) \geq 0$ is the function that returns the penetration depth at contact point j at frame i. Our system computes the penetration depth by simply executing a conventional collision detection algorithm during the evaluation of the cost term in an on-line manner without any precomputation. This is possible because our system employs a sampling-based optimization method, which allows us to deal with a non-differentiable objective like collision avoidance in the same way with differentiable ones.

The fourth cost term is to make the synthesized path smoothed by minimizing the front wheel's acceleration and the change of its angle at each frame $i \in [1, N-1]$ with weight vector $\boldsymbol{w}^{eng} \in \mathbb{R}^2$ as follows:

$$c^{eng} = h \cdot (\boldsymbol{w}^{eng})^{\mathrm{T}} \sum_{i=1}^{N-1} \begin{bmatrix} \left(\omega^i(u_1) - \omega^{i-1}(u_1)\right)^2 \\ \alpha^i(u_2)^2 \end{bmatrix}. \tag{20}$$

Here, $\omega^0(u_1)$ is the front wheel angle that has been applied in the last simulation time step. The last cost term c^{len} is used to minimize the length of the synthesized path to the target position, that is, the sum of the Euclidean distances between a pair of consecutive points on the state trajectory as follows:

$$c^{len} = w^{len} \cdot \sum_{i=1}^{N-1} \left\| \begin{bmatrix} x^{i+1} \\ y^{i+1} \end{bmatrix} - \begin{bmatrix} x^i \\ y^i \end{bmatrix} \right\|. \tag{21}$$

Here, w^{len} is the weight constant.

We solve the above trajectory optimization problem with CMA. The details of how to use CMA and the role of reset signal η are explained in Sect. 4.3.

4.3 Optimization

Given current car state $\mathbf{x}$ and target car state $\bar{\mathbf{x}}$ together with near-target frame index k and reset signal η, our system repeatedly solves a trajectory optimization problem

explained in Sect. 4.2 at each simulation time step using CMA, which facilitates on-line performance. CMA maintains multivariate normal distribution $\mathbf{y} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, iteratively updated based on the evaluation results of a given objective function at the samples populated by the distribution until a certain condition is met. In our problem, multivariate random variable $\mathbf{y}$ is regarded as the concatenation of control vectors $\mathrm{U} = \left\{\mathbf{u}^1, \mathbf{u}^2, \ldots, \mathbf{u}^{N-1}\right\}$ as

$$\mathbf{y} = \begin{bmatrix} \mathbf{u}^1 \\ \mathbf{u}^2 \\ \vdots \\ \mathbf{u}^{N-1} \end{bmatrix} \in \mathbb{R}^{2(N-1)}. \tag{22}$$

If reset signal η is on, our system purely restarts CMA with $\boldsymbol{\mu} = 0$ and $\boldsymbol{\Sigma} = \sigma^2 \mathrm{I}_{2(N-1)}$, where σ is a coefficient (standard deviation for each individual random variable of $\mathbf{y}$) and I_m is an $m \times m$ identity matrix (we set $\sigma = 0.6$ in all our experiments). Otherwise, all the context data of the previous optimization including $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ is reused to enhance time efficiency under the assumption that a solution to the current optimization will be similar to that of the previous one. In the cases when the assumption does not meet, e.g., in the case when the user has changed the target position, reset signal η is turned on by the parameter adjustment component (Sect. 5) before executing CMA.

At each optimization, the update of the normal distribution of CMA is done in a different way depending on the value of η. If η is off, it is iterated a specific number of times (300 times in all our experiments). Otherwise, CMA is completely reinitialized as mentioned above and the update is iterated more times (500 times in all our experiments).

After finishing the update, mean $\boldsymbol{\mu}$ is used as the current solution to our autonomous parking problem. We provide the first control vector in $\boldsymbol{\mu}$ to the simulation component to update the current state of the car.

5 Parameter Adjustment

Given current car state $\mathbf{x}$, target car state $\bar{\mathbf{x}}$, and current control sequence $\mathrm{U} = \left\{\mathbf{u}^1, \mathbf{u}^2, \ldots, \mathbf{u}^{N-1}\right\}$, the parameter adjustment component (Fig. 1) computes near-target frame index k and reset signal η. The former represents an index to the frame at which the car is close enough to the target state, and the latter is used to completely reinitialize CMA. We first explain how to compute near-target frame index k in Sect. 5.1. We then discuss how to determine reset signal η in Sect. 5.2.

5.1 Near-Target Frame Index Computation

In order to compute near-target frame index k, our system first produces state trajectory $\mathrm{X} = \left\{\mathbf{x}^1, \ldots, \mathbf{x}^N\right\}$, where $\mathbf{x}^1 = \mathbf{x}$ and $\mathbf{x}^{i+1} = \mathbf{f}(\mathbf{x}^i, \mathbf{u}^i)$ for $\mathbf{u}^i \in \mathrm{U}$, $i = 1, 2, \ldots, N-1$, and then measures how close the car state is to target car state $\bar{\mathbf{x}}$ at each frame using the following function derived from cost term c^{tgt} in Eq. (18):

$$\varphi(\mathbf{x}^i) = h \cdot (\boldsymbol{w}^{tgt})^{\mathrm{T}} \begin{bmatrix} \mathrm{sabs}(\bar{x} - x^i) \\ \mathrm{sabs}(\bar{y} - y^i) \\ \mathrm{sabs}(\bar{\theta} - \theta^i) \\ \mathrm{sabs}(\bar{v} - v^i) \end{bmatrix}. \tag{23}$$

Based on the measurement results, the system searches the frames satisfying $\varphi(\mathbf{x}^i) \leq \epsilon_1$ and $\left\| \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix} - \begin{bmatrix} x^i \\ y^i \end{bmatrix} \right\| \leq \epsilon_2$ in reverse order staring from the last frame, where ϵ is a small constant (we set $\epsilon_1 = 0.1$ and $\epsilon_2 = 0.5$ in all our experiments) and sets k to the index to the first of these. The former condition represents the car state must be close enough to the target car state and the latter one the car position must be near the target car position. The computed k is passed to the trajectory optimization component to determine how many frames are involved to align the car orientation with the target one before moving the car to the target position (See the formulation of c^{dir} and c^{tgt} in Sect. 4.2).

5.2 Reset Signal Computation

Whenever the optimization finishes, our system updates the near-target frame index using the new control sequence from the optimization. Let us denote the updated index by k' and the previous one by k. If $k' < k$, it means that the car becomes closer to the target state and otherwise it gets farther from the target or there is no improvement. Our system keeps track of the number of times when $k' \geq k$ denoted by T, while resetting $T = 0$ otherwise. Then, if $T > T_{max}$, where T_{max} is the maximum number of trials that have failed to find more improvement (we set $T_{max} = 5$ in all our experiments), our system turns reset signal η on to completely restart the optimization as explained in Sect. 4.3 in order to encourage CMA to find a better solution with a new normal distribution. However, we do not use this scheme when k' is less than small threshold k_{min} (we set $k_{min} = 10$ in all our experiments), because it means that the car is already quite near the target state and any more improvement is hard to be made. Our system also activates reset signal η when the user has changed the target state to restart the optimization with the new target.

6 Experimental Results

We show results for garage parking (Sect. 6.2), parallel parking (Sect. 6.1), and interactive parking (Sect. 6.3) based on on-line user inputs. The proposed framework was implemented in C/C++ programming languages. We adopted the C implementation of CMA [19] to solve our trajectory optimization problems. All experiments were conducted on a desktop computer with Intel Core TM i7 processor (3.5 GHz, 6 cores) and 32 GB memory. We set $h = 0.05$ s and used the time window of 3.5 s ($N = 70$). All the weight values used for all experiments are provided in Table 1.

Table 1. Weight values.

w^{dir}	$\boldsymbol{w}^{tgt}$	w^{col}	$\boldsymbol{w}^{eng}$	w^{len}
10^{-1}	$(5, 5, 3, 1)$	10^{5}	$(10^{-1}, 10^{-5})$	10^{-3}

6.1 Garage Parking

In this experiment, we applied our method to park the car driving into a garage without and with a dynamic obstacle moving near it when the car's initial orientation was different from the target one by 90 degrees (See the supplementary video). In the former (Fig. 2), the car first changed its orientation in front of the entrance to be aligned with the target orientation and then immediately moved into the garage. In the latter (Fig. 3), the car first moved to a side wall of the garage to avoid collision with the moving obstacle (red car) unlike the former case, while the obstacle was passing by the entrance of the garage, and then moved backwards until fully entering the garage.

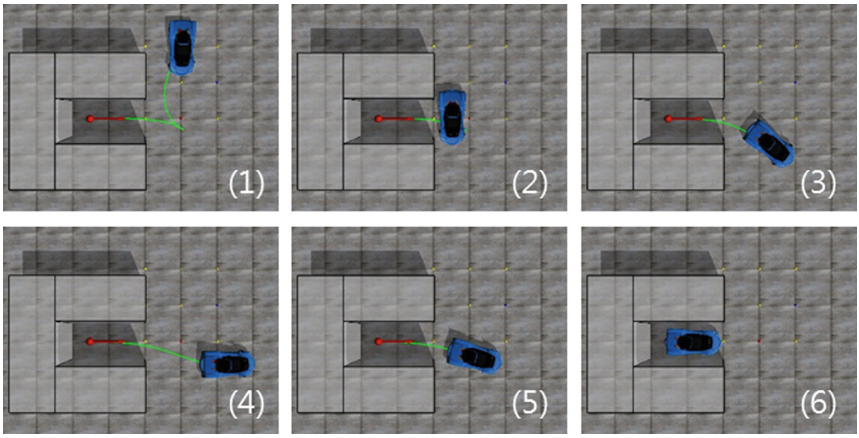

Fig. 2. Garage parking without a moving obstacle.

6.2 Parallel Parking

We conducted experiments on parallel parking without and with a moving obstacle (See the supplementary video). In the former (Fig. 4), the car first headed left-forward in preparation for getting into the parking space and then moved backward handling to the left near the wall behind. It steered to the right toward the side wall to adjust its orientation more accurately. In the latter (Fig. 5), we observed that the car moved much more forward than the former case to avoid collision with the moving obstacle. After this, the car showed a similar sequence of moves to the former.

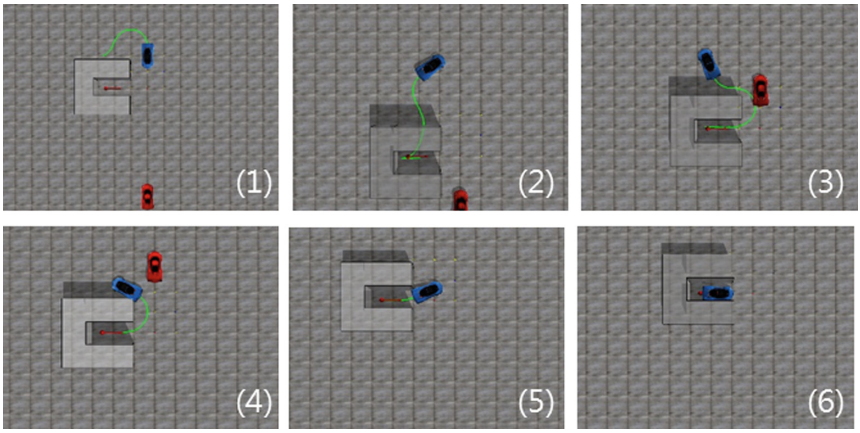

Fig. 3. Garage parking with a moving obstacle. (Color figure online)

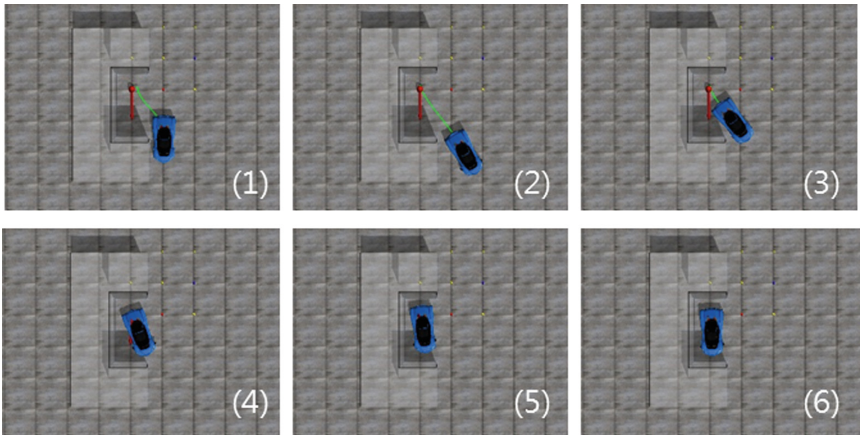

Fig. 4. Parallel parking without a moving obstacle.

6.3 Interactive Parking

In this experiment, our method allowed the user to interactively change the environment by moving the position of the walls and introducing a moving obstacle (red car) at any time point as well as the target position and orientation (See the supplementary video). Whenever the environment changed, our method restarted the trajectory optimization to find a new path that reflects the current environmental changes while rejecting the previous path (Fig. 6). As the moving obstacle appeared at a certain time point and approached, the car moved backward to avoid collision with it. After this, however, it moved back to the target position while satisfying the target orientation as well.

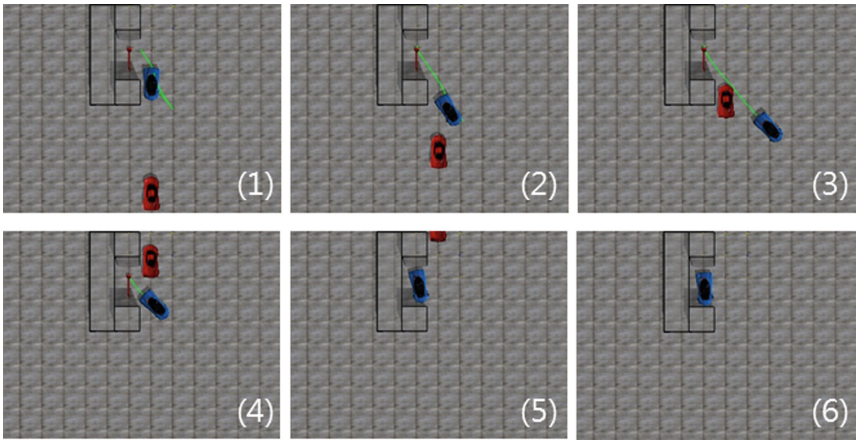

Fig. 5. Parallel parking with a moving obstacle.

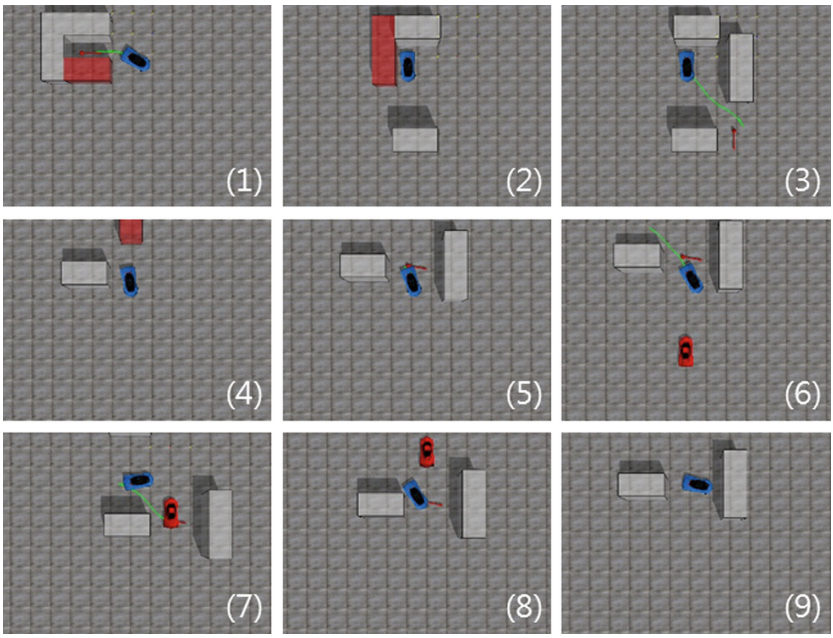

Fig. 6. Interactive parking with on-line user intervention. (Color figure online)

7 Conclusions

In this paper, we presented an on-line trajectory optimization framework for generating the parking movements of a car in a multi-obstacle environment. The proposed system repeatedly generates an optimal control sequence to drive the car toward the target state

by solving a finite-horizon trajectory optimization problem for a short time window while moving it along the time axis. As an optimization solver, we employed a sampling-based optimization technique called CMA, which requires no derivative information for an objective function. It facilitates formulating a car-parking problem with high-level objectives and dealing with collision events between the car and environment without any precomputation. We also proposed a simple and effective scheme to adjust optimization parameters such as near-target frame index and reset signal so that it may converge into the target state quickly and keep improving its solution. Our method can produce convincing parking maneuvers for garage parking, parallel parking, and interactive parking with on-line user inputs.

Though our framework can effectively avoid collision with a moving object as well as static obstacles, there is no theoretical guarantee on this constraint. We also assumed that the environmental state is fully known to perform collision detection, but it is not available to actual autonomous vehicles in general. This issue could be addressed by introducing a more realistic problem setup based on sensor inputs for environmental information. Our framework is somewhat slower than real time due to a large number of optimization variables (138 variables). It would be an interesting future research direction to implement the evaluation of an objective functions on GPU to improve the performance.

References

1. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evol. Comput. **11**(1), 1–18 (2003)
2. Sekhavat, S., Laumond, J.P.: Topological property for collision-free nonholonomic motion planning: the case of sinusoidal inputs for chained form systems. IEEE Trans. Robot. Autom. **14**(5), 671–680 (1998)
3. Reeds, J., Shepp, L.: Optimal paths for a car that goes both forwards and backwards. Pac. J. Math. **145**(2), 367–393 (1990)
4. Hsieh, M.F., Ozguner, U.: A parking algorithm for an autonomous vehicle. In: 2008 IEEE Intelligent Vehicles Symposium, pp. 1155–1160. IEEE, June 2008
5. Lee, K., Kim, D., Chung, W., Chang, H.W., Yoon, P.: Car parking control using a trajectory tracking controller. In: International Joint Conference on SICE-ICASE 2006, pp. 2058–2063. IEEE, October 2006
6. Laumond, J.P., Jacobs, P.E., Taix, M., Murray, R.M.: A motion planner for nonholonomic mobile robots. IEEE Trans. Robot. Autom. **10**(5), 577–593 (1994)
7. Muller, B., Deutscher, J., Grodde, S.: Continuous curvature trajectory design and feedforward control for parking a car. IEEE Trans. Control Syst. Technol. **15**(3), 541–553 (2007)
8. Kim, D., Chung, W., Park, S.: Practical motion planning for car-parking control in narrow environment. IET Control Theor. Appl. **4**(1), 129–139 (2010)
9. Barraquand, J., Latombe, J.C.: Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles. Algorithmica **10**(2–4), 121 (1993)
10. LaValle, S.M., Kuffner Jr., J.J.: Randomized kinodynamic planning. Int. J. Robot. Res. **20** (5), 378–400 (2001)

11. Kuwata, Y., Fiore, G.A., Teo, J., Frazzoli, E., How, J.P.: Motion planning for urban driving using RRT. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2008, pp. 1681–1686. IEEE, September 2008
12. Buehler, M., Iagnemma, K., Singh, S. (eds.): The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, vol. 56. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03991-1
13. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. Autom. **12**(4), 566–580 (1996)
14. Dolgov, D., Thrun, S., Montemerlo, M., Diebel, J.: Path planning for autonomous vehicles in unknown semi-structured environments. Int. J. Robot. Res. **29**(5), 485–501 (2010)
15. Zhao, Y., Collins Jr., E.G.: Robust automatic parallel parking in tight spaces via fuzzy logic. Robot. Auton. Syst. **51**(2–3), 111–127 (2005)
16. Gorinevsky, D., Kapitanovsky, A., Goldenberg, A.: Neural network architecture for trajectory generation and control of automated car parking. IEEE Trans. Control Syst. Technol. **4**(1), 50–56 (1996)
17. Zips, P., Bock, M., Kugi, A.: A fast motion planning algorithm for car parking based on static optimization. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2392–2397. IEEE, November 2013
18. Tassa, Y., Mansard, N., Todorov, E.: Control-limited differential dynamic programming. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 1168–1175. IEEE, May 2014
19. https://www.lri.fr/∼hansen/cmaes_inmatlab.html
20. Tassa, Y., Erez, T., Todorov, E.: Synthesis and stabilization of complex behaviors through online trajectory optimization. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4906–4913. IEEE, October 2012