




# Real-Time Implementation of Randomized Model Predictive Control for Autonomous Driving

Arun Muraleedharan , Hiroyuki Okuda , *Member, IEEE*, and Tatsuya Suzuki , *Member, IEEE*

**Abstract**—Model predictive control (MPC) using randomized optimization is expected to solve different control problems. However, it still faces various challenges for real-world applications. This paper attempts to solve those challenges and demonstrates a successful implementation of randomized MPC on the autonomous driving using a radio-controlled (RC) car. First of all, a sample generation technique in the frequency domain is discussed. This prevents undesirable randomness which affect the smoothness of the steering operation. Second, the proposed randomized MPC is implemented on a Graphics Processing Unit (GPU). The expected GPU acceleration in calculation speed at various problem sizes is also presented. The results show the improved control performance and computational speed that was not achievable using CPU based implementation. Besides, the selection of parameters for randomized MPC is discussed. The usefulness of the proposed scheme is demonstrated by both simulation and experiments. In the experiments, a 1/10 model RC car is used for collision avoidance task by autonomous driving.

**Index Terms**—Autonomous vehicles, graphics processing unit (GPU), model predictive control, sampling based optimization.

## I. INTRODUCTION

MODEL Predictive Control (MPC) [1]–[3] is a popular control method for multi-variable constrained control problems. Although MPC had first appeared in [4] published in the 1960s, the major growth in its application emerged later as the result of the improvements in computational resources. The model in MPC stands for the underlying model representing the dynamics of the system being controlled. This model enables future prediction of the controlled system. In addition, MPC observes the state at each control cycle. This makes the system tolerant of environmental changes in real-time. Motion control is a problem that has been difficult to handle in real-time. It is, however, well addressed by MPC due to its above-mentioned advantages. Autonomous driving has positioned itself in the difficult end of motion control problems. Being highly dynamic

and safety-critical, it requires the calculation of accurate control inputs in real-time. The computation involved within the MPC to solve the model-based optimization problem has been challenging in real-time implementations. One of the ideas to overcome the computational complexity is a sampling based MPC called Randomized MPC (RMPC) [5]–[10] that enables us to balance computational demands at the expense of accuracy. Regardless of its simplicity in implementation, RMPC explores the global solution space and perform strict constraint satisfaction checks in every iteration. The infinite horizon stability and optimality of RMPC are proven in [5] and [7], respectively. Lack of an index of optimality and being semi-optimal at the same time remains to be a challenge in RMPC application.

On the other hand, behavior control of an autonomous vehicle can be seen as two distinct tasks- path planning and path following. They are executed alternately. Planning has to be repeated since the planner has to consider constraints on safety such as collision avoidance and speed limitation. This is in addition to the basic target of path planning for keeping a lane, overtaking, etc. Many path planning schemes have been well demonstrated in robotics and automotive applications as seen in [11]–[14]. Considering that the car has an updated path that is planned, the next step is to follow it accurately. For accurate path following, a controller needs to observe the present state of the car and provide correct steering and acceleration inputs. There have also been many works on path following [1]–[3], [15], [16]. Path following control could be performed using various control strategies such as proportional–integral–derivative (PID) control, state feedback controllers, MPC, and so on. Having the path planning and tracking as two different processes is a popular method in the literature. Considering vehicle dynamics constraints in both stages separately implies that such consideration becomes unnecessarily redundant. Combining these two into one problem formulation clearly has the benefit of a lower computational burden by removing redundancies in vehicle dynamics calculations. Such combined problems are often referred to as simultaneous motion planning and control (SMPLC) problem [17]–[19].

During the studies on SMPLC by the authors in [17] and [18], real-time implementation of SMPLC was found to be facing various challenges. The major challenge was due to the computational complexity in solving an SMPLC problem. Some notable examples that had demonstrated real time steering control with non-linear MPC are [3] and [20]. The work [3] has reported an average computation time, which is over 150 ms, limiting their stable experiment speed at 7 m/s. While, [20] demonstrates an average processing time of 60 ms (with pre-calculated invariant sets), enabling multiple obstacle avoidance at a speed

Manuscript received May 14, 2020; revised September 29, 2020 and December 23, 2020; accepted February 22, 2021. Date of publication March 1, 2021; date of current version April 21, 2022. (*Corresponding author: Arun Muraleedharan.*)

Arun Muraleedharan and Tatsuya Suzuki are with the Department of Mechanical Systems Engineering, Nagoya University, Nagoya 464-8603, Japan (e-mail: muraleedharan.arun@e.mbox.nagoya-u.ac.jp; t\_suzuki@nuem.nagoya-u.ac.jp).

Hiroyuki Okuda is with the Department of Mechanical Science and Engineering, Nagoya University, Nagoya 464-8603, Japan (e-mail: h\_okuda@nuem.nagoya-u.ac.jp).

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/TIV.2021.3062730>.

Digital Object Identifier 10.1109/TIV.2021.3062730

of 14 m/s. Both [3] and [20] show obstacle avoidance while driving on a straight road. The strict real-time limits associated with a safety-critical problem like autonomous driving made it difficult to find an optimal solution in real-time. Even sampling-based optimization schemes like RMPC could not be realized in necessary control frequency with traditional CPU based implementation. During the simulation study of RMPC for vehicle control, another issue was also noticed. The randomness in the sampling step was found to give a negative impact on the smoothness of the car which is being driven. This will be more emphasized when the RMPC is applied to passenger vehicles rather than autonomous robots. This necessitates a need for a smoothing technique in the sampling phase so that the control action remains smooth.

By taking a deeper look at the RMPC framework, the observer would notice its inherent nature of parallelism. The sequence of processes like series creation and calculation of cost could be performed on all sample points simultaneously. This interesting feature of RMPC can directly be linked to the parallel processing feature of a Graphics Processing Unit (GPU). At present, every application that contains abundant parallelism is showing interest in GPUs. GPUs are of interest due to their significantly larger computational throughput in comparison to the CPUs, which are built for higher latency. This gives GPU's a tremendous potential to improve the computational speed of applications with massive parallelism [21].

Previous works on sampling based MPC can be divided into two categories. The first category is the development of the sampling based solver for MPC [5], [22], [23]. The second one is the study focusing on the implementation of Sampling based MPC [24]–[28]. In the first category, [22] developed a method that searches the control space by seeding a tree from the previous best sample and expanding it in a manner similar to rapidly exploring random tree (RRT). The paper [5] suggested a sampling based MPC for one-leg robot navigation using a potential field representation of obstacles. Even though implementation is not the focus of this work, the authors claimed that they could improve performance only at the expense of computational load. The paper [23] applied a modified version of RMPC to the drive of high-speed RC cars, which enabled faster real-time computation.

In the second category, the control of a drinking water network is addressed in [24]. Since the network is large and contains multiple controllable elements, a GPU based implementation was suggested. Although the problem size is large, this application adopted longer control intervals than the case of controlling a car due to its slow dynamics. In [25], the authors successfully controlled a mobile robot for obstacle avoidance using an embedded GPU. Here the robot dynamics were considered as a point mass model and the robot speed was extremely low. Different approaches for nonlinear model predictive control that runs on GPU can be seen in papers [26] and [27]. The optimization algorithms used were not sampling based in nature and the application domain was much different from the control of an autonomous vehicle. Finally, in the survey paper [28], a summary of various parallel implementations of MPC was described. This paper presents a detailed discussion of the present challenges that MPC faces and how parallel computation helps to solve those challenges. This paper concludes that the majority of GPU based implementations are faster in doing calculations,

but suffer from a high overhead of memory transfer time and kernel creation time. Once combining the time of overheads and calculation, any improvement over the CPU case is not significant at regular use cases.

Since none of the previous works directly addressed autonomous driving with RMPC, we have studied the feasibility of the RMPC for autonomous driving. The paper [17] has implemented obstacle avoidance driving of a car using RMPC in simulation. This paper suggested some ideas to smoothen the randomness by sampling in the frequency domain. This idea showed smoother steering input, however, the controller was restricted to straight line driving at a constant speed. Also, the effectiveness was verified only in simulation with pre-calculated inputs. In addition, the ego car being represented with a bicycle model, this study concluded that more computational power is necessary to use the RMPC in real-time. This demand for computational improvement was also investigated in our previous works [29] and [30]. Computation speed of linear MPC problems was accelerated using GPU in [29], however, this was limited to linear MPC problems. On the other hand, [30] discussed some preliminary results showing the GPU's potential to accelerate RMPC. This study was also limited to simulations and to straight line driving at constant speed.

Based on these considerations, this paper addresses the RMPC as a solution for autonomous driving control. First, it extends the previous works [17] and [30] with an improved vehicle model, thereby removing the limitations of driving only in straight line and with a constant speed. Then, it solves the problem of randomness by frequency domain sampling and implements the RMPC along with the new sampling methodology. Second, the proposed RMPC with frequency domain sampling is implemented on GPU. In-order to transfer the algorithm to GPU, each sample is processed by a different GPU thread at the same time. With this higher computational power, RMPC can process a larger number of samples or even consider a longer prediction horizon in strict real-time limits. The presented idea of GPU implementation eliminates the need for transferring large data between GPU and CPU. This method also keeps the kernel creation overhead to a minimum by not splitting the MPC into subproblems. This solves the issues of time overheads that are typically present in GPU based implementations [28]. Third, a novel method of step-by-step parameter selection for RMPC is suggested. Simulations and experiments demonstrate how this new sampling method combined with the realization by GPU improves the control performance. The validity of the proposed ideas is demonstrated by using a 1:10 RC car that has the same kinematics as a normal road car.

Attending existing bottlenecks faced by RMPC and demonstrating its potential as a fast and easy-to-implement controller, this work becomes promising for the future of autonomous driving. Although our target is autonomous driving control, the presented ideas of RMPC, frequency-domain sampling and GPU implementation can be powerful tools for other applications, particularly, fast real-time mechanical systems control.

## II. PROBLEM SETTING

### A. Task Description

In-order to demonstrate RMPC and its improvements, a collision avoidance task as shown in Fig. 1 is chosen. There is an ego

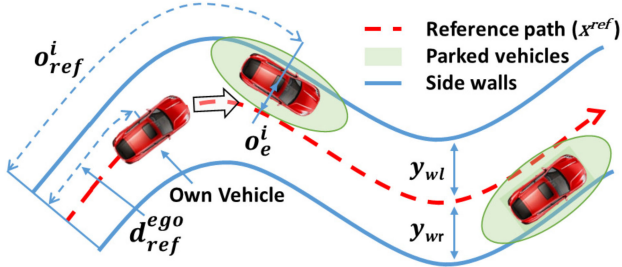


Fig. 1. Target environment.

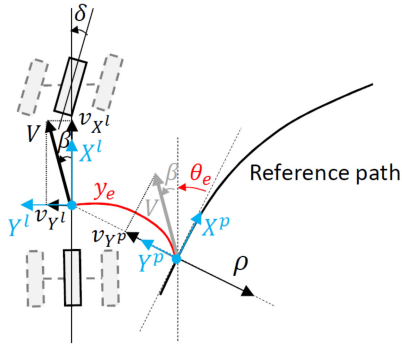


Fig. 2. Approximate bicycle model of the car.

car controlled by the controller following a pre-defined reference path  $x^{ref}$ . The reference path, expressed in a local coordinate frame attached to the car ( $\Sigma^l(t)$ ) as seen in Fig. 2 is defined as follows [29];

$${}^l\chi_{ref}(t) = \{{}^l p_{ref,i}(t) | i = 0, 1, \dots, N_p\} \quad (1)$$

$${}^l p_{ref,i}(t) = \{{}^l x_{ref,i}^p(t), {}^l y_{ref,i}^p(t)\}. \quad (2)$$

Here,  ${}^l\chi_{ref}(t)$  is the reference path at time  $t$ , it contains  $N_p$  path nodes  ${}^l p_{ref,i}(t)$ , where  ${}^l p_{ref,i}(t)$  denotes the relative position of the path node,  ${}^l x_{ref,i}^p(t)$  and  ${}^l y_{ref,i}^p(t)$ , from the car's origin. Here the deviation of the car from the reference path,  $y_e(t)$ , is computed as the distance from the car to  $OP^p(t)$ , the origin of the path coordinate frame  $\Sigma^p(t)$  at the time of  $t$ , which is computed by applying the suitable interpolation between path nodes. The car ideally should drive along the center-line of this 6 m wide path. It starts at the point  $(d_{ref}^{ego}(t=0), y_e(t=0), \theta_e(t=0)) = (0, 0, 0)$ , which is the origin. Variable  $t$  is the time index,  $d_{ref}^{ego}$  represents the distance covered by ego car along the reference path and  $y_e, \theta_e$  represent lateral error and yaw error of ego car from the reference path. The car should also make necessary steering maneuver not to collide with the  $L$  parked cars on both sides of the street whose position is represented by  $(o_{ref}^i, o_e^i), i \in \{1, 2, \dots, L\}$  for the  $i$ th parked car. The controller controls the ego car motion by providing speed and steering angle commands.

This problem setting is approached as a simultaneous motion planning and control (SMPLC) Problem. The controller is an MPC with the necessary constraints. Since SMPLC unifies the

TABLE I  
DEFINITION OF PARAMETERS AND VARIABLES

Parameter	Description	Units
$C_f$	Cornering stiffness - Front tire	N/rad
$C_r$	Cornering stiffness - Rear tire	N/rad
$m$	Mass of the vehicle	kg
$I_z$	Yaw moment of inertia	kgm <sup>2</sup>
$l_f$	Distance from gravity to front axis	m
$l_r$	Distance from gravity to rear axis	m
$\delta$	Steering angle	rad
$V$	Forward velocity of vehicle (const.)	m/s
$\rho$	Curvature of reference path	1/m

planning and tracking into one problem considering the constraints and car dynamic model, there is no need to plan a path first and then follow it for obstacle avoidance.

In order to represent the dynamics of the car, an equivalent bicycle model (Section II-B) is used. The previous works that implemented MPC for navigation using GPU [25] have adopted a point-mass model for simplifying the problem. Due to the higher speed of motion, an equivalent bicycle model is of choice. To keep a safe distance from the obstacles and the sidewalls, this framework utilizes some input and safety constraints as shown in Section II-C.

For reliable real-time performance, sample-based approach is chosen to solve the nonlinear optimization problem. The sample-based method used in this work is known as RMPC, and is attracting increased attention recently.

Random sampling based methods often lead to random variations in the control input. This is not recommended for smoothness critical tasks like driving. The author's previous work [17] indicates some preliminary results indicating a significant advantage of sampling from the frequency domain for smooth driving control. In [17], an Inverse Discrete Cosine Transform (IDCT) was used in the sample generation process from frequency domain. We demonstrate the IDCT method being executed in a GPU and driving our RC car smoothly.

### B. State Space Equation for Car Dynamics

The vehicle behavior in this paper is assumed to be represented by an approximate bicycle model with its coordinate system along the reference path (Fig. 2). The state equation of this model is expressed as follows [31].

$$\begin{bmatrix} \dot{l}v_Y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} -\frac{a_{11}}{l_{v_X}} & \frac{a_{12}}{l_{v_X}} - l_{v_X} \\ -\frac{a_{21}}{l_{v_{X^l}}} & \frac{a_{22}}{l_{v_X}} \end{bmatrix} \begin{bmatrix} l_{v_Y} \\ r \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \delta, \quad (3)$$

where

$$\begin{aligned} a_{11} &= (C_f + C_r)/m, & a_{12} &= -(l_f C_f - l_r C_r)/m, \\ a_{21} &= (l_f C_f - l_r C_r)/I_z, & a_{22} &= -(l_f^2 C_f + l_r^2 C_r)/I_z, \\ b_1 &= C_f m, & b_2 &= l_f C_f / I_z. \end{aligned}$$

where  $l_{v_X}$  and  $l_{v_Y}$  represents the longitudinal and the lateral velocity in local coordinate frame, respectively. Please refer to Table I for other variables. We further transform the nonlinear state equation that is given above into an approximate bicycle model with its coordinate system along the reference path. With the assumption of vehicle speed  $V$  being constant within

prediction horizon, the slip angle of the vehicle  $\beta$  and  $\theta_e$  to be small enough, the dynamics of the lateral tracking error  $y_e$  and the angle  $\theta_e$  are summarized as follows (discretized with Euler's Method):

$$x(k+1) = A_d(k)x(k) + B_d(k)u_t(k) + W_d(k)\rho(k), \quad (4)$$

$$y(k) = Cx(k), \quad (5)$$

$$x(k) = [y_e(k) \dot{y}_e(k) \theta_e(k) \dot{\theta}_e(k) \delta(k)]^T \quad (6)$$

$$u_t(k) = \delta^*(k) \quad (7)$$

$$A_d(k) = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 \\ 0 & 1 - \frac{a_{11}}{V(k)}\Delta t & a_{11}\Delta t & \frac{a_{12}}{V(k)}\Delta t & b_1\Delta t \\ 0 & 0 & 1 & \Delta t & 0 \\ 0 & -\frac{a_{21}}{V(k)}\Delta t & a_{21}\Delta t & 1 + \frac{a_{22}}{V(k)}\Delta t & b_2 \\ 0 & 0 & 0 & 0 & 1 - \alpha\Delta t \end{bmatrix} \quad (8)$$

$$B_d(k) = [0 \ 0 \ 0 \ 0 \ \Delta t]^T \quad (9)$$

$$W_d(k) = [0 \ (a_{12} - V^2(k))\Delta t \ 0 \ a_{22}\Delta t \ 0]^T \quad (10)$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \quad (11)$$

Where  $\rho$  represents the curvature of the reference path at the nearest point and  $\delta^*$  represents the tire angle reference. Actual tire angle  $\delta$  is considered to have a first order delay from  $\delta^*$  to represent physical delay in steering system. Readers are referred to our previous work [29] for further details.

### C. Input and Safety Constraints

There are two constraints in this formulation, one is the hard constraint to ensure feasible input commands and the other one is to prevent entry into the prohibited area.

The range of control input  $u_t$  and its rate of change  $\Delta u_t$  are constrained as follows to maintain physical limits of steering system and to prevent urgent steering action, respectively.

$$|u_t| < 0.1745 \ (\approx 10 \text{ [degrees]}) \quad (12)$$

$$|\Delta u_t| < 0.35 \text{ radians/sec} \ (\approx 20 \text{ [degrees/sec]}) \quad (13)$$

For each parked car, an ellipse around them defines the prohibited area with a safety margin. This area for  $i$ th parked car is expressed by an inequality

$$\left(\frac{d_{ref}^{ego}(t) - o_{ref}^i}{r_a^i}\right)^2 + \left(\frac{y_e(t) - o_e^i}{r_b^i}\right)^2 > 1 \quad \forall i \in \{1, 2, \dots, L\} \quad (14)$$

where  $L$  is the number of cars parked,  $(o_{ref}^i, o_e^i)$  is the position of  $i$ th parked car. The length of the major and minor axis of the ellipse around  $i$ th parked car are  $r_a^i$  and  $r_b^i$ , respectively.

The areas beyond sidewalls are also expressed as prohibited areas represented with a logarithmic function as explained in the next subsection (19).

### D. Cost Function and Reference State

The goal of the controller is to drive the car on the center of the street, without colliding with the obstacles. On taking a closer look at the given case, it can be seen that the need to follow the street center and to stay away from obstacles are obviously conflicting. While avoiding obstacles, it is also important to keep a safe distance from them, than following the closest path to the obstacle. This is satisfied by adding a potential field term in the cost function. There are weight parameters to balance between terms to stick to the reference path and terms to avoid a collision. Finally, another potential field is also added to keep a safe distance from the side walls.

The cost function  $J(u_t)$  for an input series  $u_t = [u(0|t), u(1|t), \dots, u(N-1|t)]$  to be optimized at time  $t$  is

$$\begin{aligned} J(u_t) = & \sum_{k=1}^{N-1} s_0(k|t) (\phi^T(k|t)Q\phi(k|t) + \Delta u(k|t)^T R \Delta u(k|t)) \\ & + \phi^T(N|t)Q_f\phi(N|t) + Q_{obs} \sum_{k=1}^{N-1} s_j P_{obs}^j(k|t) \\ & + Q_{wall} \sum_{k=1}^N P_{wall}(k|t) \end{aligned} \quad (15)$$

$$\text{where } \phi(k|t) = y(k|t), \quad (16)$$

$$\begin{aligned} \Delta u(0|t) = 0, \Delta u(k|t) = |u(k|t) - u(k-1|t)|_1 \\ \forall k \in \{1, 2, \dots, N-1\}, \end{aligned} \quad (17)$$

where  $N$ ,  $\phi$  and  $\Delta u$  represents prediction horizon length of MPC, state error and the time difference in control input, respectively. The representation  $u(k|t)$  is the value of  $u$  at prediction horizon step  $k$  at time  $t$ .  $Q = \text{diag}(Q_{y_e}, Q_{\theta_e})$  and  $R$  being weight parameters, balance the control performance to the effort.  $Q_f$  acts on the last step in the prediction horizon as a penalty to the residue.

This framework enables us to follow any reference path  $x^{ref}$  that we need, the reader can refer to our previous work [29] for a detailed study on the path tracking performance of this dynamic model. Further details on the weight parameter matrices and the terms  $s_i(k|t)$ ,  $s_0(k|t)$  can be found in author's previous work [17]. When the ego car approaches a parked car  $i$ , the switching parameter  $s_i$  approaches to 1. This makes the repulsive force more significant. Otherwise, following the reference state is of higher priority.

$Q_{obs}$  and  $Q_{wall}$  works as weight parameters for the potential field around parked cars and sidewalls. Potential function around  $j$ th parked car is expressed by  $P_{obj}^j$ :

$$P_{obj}^j(k|t) = C \exp\left(-\left(\frac{d_{ref}^{ego}(k|t) - o_{ref}^j}{r_a^j}\right)^2 - \left(\frac{y_e(k|t) - o_e^j}{r_b^j}\right)^2\right) \quad (18)$$

where  $[d_{ref}^{ego}(k|t), y_e(k|t)]$  are the ego car position along reference path predicted for prediction horizon step  $k$  at time  $t$ .  $C$ ,  $r_a^j$  and  $r_b^j$  can adjust potential field area and magnitude. Side walls

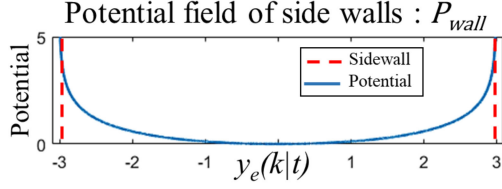


Fig. 3. Potential field of sidewalls.

have their own potential field  $P_{wall}$ :

$$P_{wall}(k|t) = (\log |y_{wl}| + \log |y_{wr}|) - (\log(y_{wl} - y_e(k|t)) + \log(y_e(k|t) - y_{wr})) \quad (19)$$

where  $y_{wl} = 3$ ,  $y_{wr} = -3$  are side wall locations.  $P_{wall}$  is designed to increase steeply close to the walls, as shown in Fig. 3.

### E. Formulation of Input Optimization Problem

The optimization problem to be solved in every control step in order to generate optimum control input series can be formulated as follows:

**given**

$$x(0|t) = x(t), x^{ref}, \quad (20)$$

**find**

$$u(k|t), \quad (k \in \{0, 1, \dots, N-1\}) \quad (21)$$

**which minimize**

$$J(u_t = \{u(k|t)\}), \quad (k \in \{0, 1, 2, \dots, N-1\}) \quad (22)$$

**subject to**

$$\begin{aligned} &\text{Input constraints (12), (13)} \\ &\text{Prohibited area constraint (14), (19)} \\ &\text{Car dynamics (4).} \end{aligned} \quad (23)$$

## III. SEMI OPTIMAL SOLUTION USING SAMPLING IN FREQUENCY DOMAIN

The problem in consideration is a non-linear optimization problem and it has to be solved in real-time. There are non-linear constraints associated with safe driving and a non-linear vehicle model. Typical methods to find the solution is approximating the non-linear constraints in some way for the solvers [23]. This paper tries to eliminate these approximations by the application of a random sample based approach. This method is proven to provide a semi optimal solution which is close enough to the optimal solution, once the necessary sample size is considered [7]. The process of this method of optimization starts with sample generation.  $N_s$  number of sample sequences that are of the length  $N$  is picked from a specific distribution of random numbers. The next step is to calculate the future trajectory for the car, assuming it follows each of these sample sequences. At this point, some constraint based filtering is performed to remove the samples that are infeasible. In the case if majority of the samples are found to be infeasible, there is threat on safety. In such a case, the controller should switch to a mode of emergency stop or request manual driving. This follows the calculation of the cost

associated with each sample according to a cost function. The minimum cost series is then identified and it's first element goes as the input to the ego car. The controller repeats these steps for each control cycle.

### A. Sampling in the Frequency Domain for Smoother Input Samples

Generating samples randomly is expected to carry the randomness into the control input as well. Since given control input is the steering command to the car, this randomness is not recommended for smooth driving performance. The suggested sampling method generates the samples from the frequency domain. These are then converted into the time domain using the process of Inverse Discrete Cosine Transform(IDCT). Following this process, it is possible to obtain smoother samples that drives the car smoother.

The following set of equations explains the process of sample generation in detail.

$$u_{IDCT}^i(0|t) = u(t-1) \quad (24)$$

$$u_{IDCT}^i(k|t) = u_{IDCT}^i(k-1|t) + \Delta u_{IDCT}^i(k|t), \quad \forall k \in \{1, \dots, N\} \quad (25)$$

$$\Delta u_{IDCT}^i(t)^T = \gamma D U_{IDCT}^i(t)^T \quad (26)$$

$$U_{IDCT}^i(l|t) \begin{cases} \sim \mathcal{U}(-1, 1) & \text{if } l \leq F_{c/o} \\ = 0 & \text{other.} \end{cases} \quad (27)$$

Here,  $D \in R^{N \times N}$  is the coefficient of IDCT and  $l$  is the index representing the frequency component. Each element of  $D$  is calculated as follows:

$$D_{ij} = \sqrt{\frac{2}{N}} k_i \cos\left(\frac{(i-1)(j-1/2)\pi}{N}\right), \quad i \in \{1, 2, \dots, N\}, j \in \{1, 2, \dots, N\} \quad (28)$$

where  $N$  is the length of input series.  $k_i$  ( $i = 1, 2, \dots, N$ ) is a normalizing factor that has two values

$$k_i = \begin{cases} \frac{1}{\sqrt{2}} & i = 1 \\ 1 & i \neq 1 \end{cases}. \quad (29)$$

$U_{IDCT}^i(t)$  is sampled from a uniform distribution ranging  $(-1, 1)$  in this case.  $\gamma$  is a parameter that can be used to adjust the resulting input rate of change according to (13).  $F_{c/o}$  is a cut-off threshold that prevents higher frequency components in the resulting input sequence. The smaller the  $F_{c/o}$  the smoother in the resulting input series. Unless specified, an  $F_{c/o}$  of 15 is used in this paper. If any of the element  $u_t$  generated in (25) is found to violate the constraint (12), a new random number is generated to replace the  $U_{IDCT}^i(l|t)$  until  $u_t$  satisfies (12). The control performance with and without IDCT can be seen in Fig. 4 ( $N_s = 500$ ). The graphs show visible improvement in control signal smoothness and steering performance. In summary, the contents of IDCT is essentially random number generation followed by some matrix multiplications and series making. The following sections will demonstrate how the above-mentioned operations were efficiently ported to GPU.

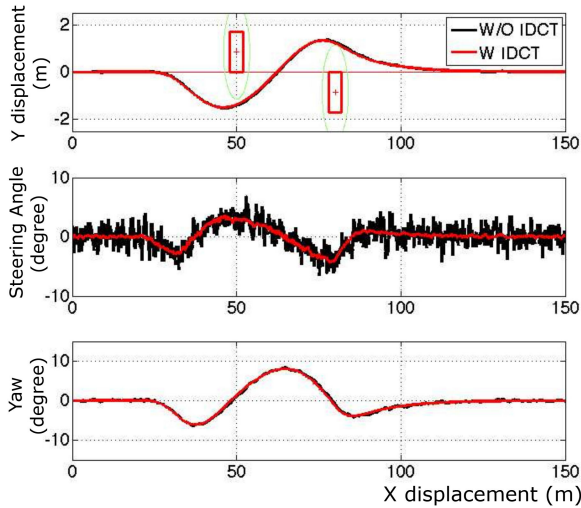


Fig. 4. Control performance with and without IDCT.

### B. Validation of Proposed Sampling Method

Before using RMPC for real experiments, its performance is evaluated in simulation-based experiments. Since the experiments are performed in a 1:10 scale car, a high fidelity car simulator is necessary for simulations. Carsim by Virtual Mechanics Inc. is used owing to its excellent vehicle and environment models. The controller issues an updated control input in every control cycle. In the simulation experiments, the control interval  $dT$  is set as 0.1 seconds. The prediction horizon length is set as  $N = 50$ . This means that the controller calculates the optimal input considering vehicle behavior predictions of 5 seconds into the future. The following combination of parameters resulted in the best possible tracking performance.  $Q_f = 1$ ,  $Q = \text{diag}(10, 10)$ ,  $R = 3000$ ,  $Q_{obs} = 3000$  and  $Q_{wall} = 5$ , respectively. The parameter  $\gamma$  is set to 1. The parked cars are at the positions  $o^1 = (50, 0.85)$  and  $o^2 = (80, -0.85)$ , respectively. Since the optimization is based on random samples, there is an obvious question of its closeness to the global optimum. There have been some previous works [7] that quantifies the minimum sample size for the solution to be global with a defined probability level. Referring to [7], it is calculated that a sample size of  $N_s > 458$  is necessary for the solution to have confidence level of 99% ( $\alpha_{believe} = 0.01$ ). Hence, the controller is tested at a sample size of  $N_s = 500$ . Fig. 5 demonstrates the performance. Tracking performance is very accurate without any intrusion to the obstacle area which is prohibited. While the author's previous works [17] were limited to the performance of the controller at sample sizes less than 500, this paper tries to go beyond this number. Even though  $N_s = 500$  is sufficient for satisfiable performance at the simulation conditions, sample sizes beyond 500 are found to provide better controller performance as shown in Fig. 5. The majority of control tasks have multiple control variables and stricter constraints. Such cases would clearly demand a higher number of samples to be processed in real-time. Another interesting trend in the case of driving is related to driving speed. The faster the car is, the shorter the control interval has to be. Both these conditions demand higher computational performance. But due to hardware

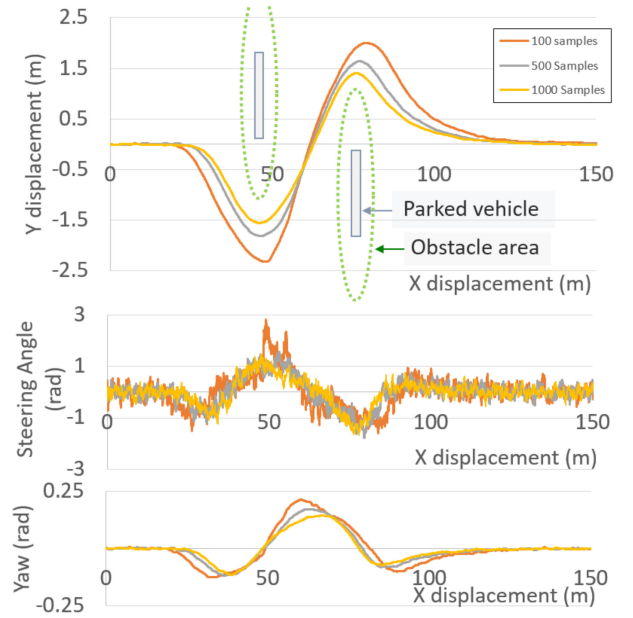


Fig. 5. Number of samples and controller performance.

TABLE II  
STEPS IN COMPUTATION

Step	Function	Equation reference
1	Random Number Generation	(27)
2	IDCT	(26)
3	Series Generation	(25)
4	State Matrix Calculation	(4)
5	Cost Calculation	(15)
6	Minimization of cost	–

limits, the sample size  $N_s < 500$  was the limit that a CPU based implementation could process in real-time. This limitation in computation power was also expressed in [17] and [23].

## IV. IMPLEMENTATION USING GPU

### A. Computational Steps Involved

The process of calculating the optimum control input can be expressed in a sequence of 6 steps as described by Table II. Once the present state variables are obtained from the car, the controller performs steps 1 to 6. The final step does a minimization of all the samples based on a cost function and comes up with the best sample sequence. The second element in this sequence will be the optimum input for the car in the next step. This will be then sent to the car. This process is repeated in every control cycle. The following subsections are dedicated to the detailed contents of these steps and the methods in moving them to support parallel computation using GPU.

### B. CPU Baseline

In order to have a better understanding about the improvements by parallel computing, The controller was first implemented using a CPU based algorithm. The steps are expressed in Algorithm 1. The CPU used in this study is a 2.2 GHz Intel Core i5. CPU program uses a random device function of the C++ Numerics library [32] to generate random numbers. Please

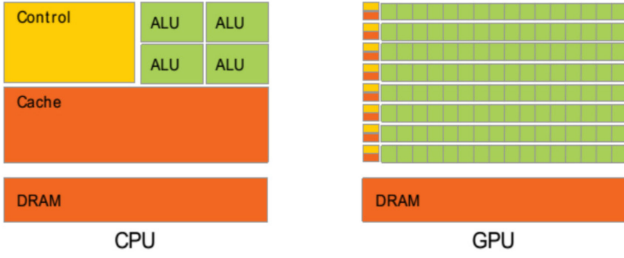


Fig. 6. CPU vs GPU architecture.

**Algorithm 1:** Randomized MPC.

---

```

1: for Every time step do
2:   Generate  $N_s$  input samples of length  $N$  acc. to 2.
3:   Initialize state matrix.
4:   for  $Sample = 1, 2, \dots, N_s$  do
5:     for  $Length = 1, 2, \dots, N$  do
6:       Calculate the extended state matrix acc. to 1.
7:     end for
8:   end for
9:   for  $Sample = 1, 2, \dots, N_s$  do
10:    for  $Length = 1, 2, \dots, N$  do
11:      Calculate the cost for each sample acc. to 3.
12:    end for
13:  end for
14:  Find minimum cost sample  $S_{min}$ .
15:  Return second element of  $S_{min}$  as next control input.
16: end for

```

---

note that the steps from 2 to 5 in Table II are performed on every element, one element at a time using two ‘for loops’. First one is a ‘for loop’ that index the sample number  $N_s$ , the second one for the length of the sample  $N$ . This makes up  $N_s \times N$  calls to the CPU one after the other.

*C. Code Design for GPU*

The various steps involved in the MPC algorithm were divided into 6 discrete steps as seen in Table II. These steps were studied in detail to find possible improvements by porting to a 2560 core GPU (Nvidia GeForce GTX1080 1.7 GHz). The basic difference in hardware of CPUs and GPUs are as visualized in Fig. 6 [34]. While a normal CPU has several fast ALUs (Arithmetic Logic Units) sharing common cache memory and controlled by a common controller, GPUs have thousands of ALUs with individual controller and cache memory. Since all of the ALUs in a GPU can run a thread of operation in parallel, the basic idea of implementation is to assign a sample point to each GPU thread, as shown in illustrative picture Fig. 7. Random samples are stored in an array of size  $N_s \times N$ , same number of GPU threads are also assigned as a block of size  $N_s \times N$ .

The first step of random number generation is performed with Pseudo-random number generation function in CURAND library [33], which is a part of Nvidia’s CUDA library [34]. Step 2 that performs the IDCT operation (As seen in (22)) and Step 5 that calculates cost (As seen in (10)), are independent

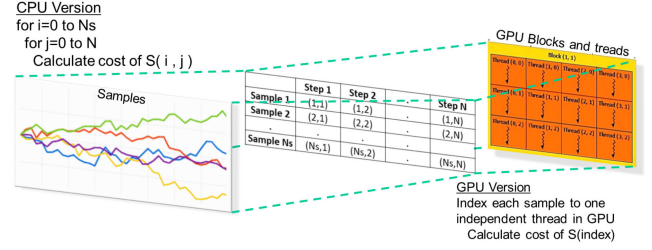


Fig. 7. GPU implementation image.

at each discrete sample points. Being independent, they can use as many GPU threads as the hardware allows simultaneous operation. Matrix multiplication operations associated with Step 2 can also use the maximum possible capacity of GPU. These steps while using one sample point per GPU thread gives  $N_s \times N$  times faster computation. The remaining steps in the computation like Steps 3 and 4 depend on previous prediction horizon/series value to compute next. These steps have to be performed in  $N$  steps, calculating all  $N_s$  samples at a given time using  $N_s$  number of threads. Even here we get  $N_s$  times improvement in computational time over CPU.

As described in the survey paper [28], the majority of GPU based implementations are faster in calculation but suffer from a high overhead of memory transfer time and kernel creation time. The GPU implementation presented here avoid these problems in following ways:

- By generating the random numbers within the GPU memory (step 1 in table II), all the steps of computation are kept within the GPU and its memory. Only the present state of the car is transferred to GPU and the optimum input is sent back.
- Certain problem-parallel approaches in literature splits the MPC to smaller problems by dividing the prediction horizon and assign each section to different GPU kernels. Thanks to the random sample based method, our data-parallel approach does not split the MPC into subproblems, limiting the kernel creation time to the minimum.

## V. EVALUATION USING RC CAR

Using GPU, we expect to produce the same quality control signals as that of CPU with significantly lower computation time. In the context of controlling a car, this can give multiple benefits. Previous works based on simulations did not consider the delays caused by localization, error estimation and communication delay. Model errors and various other noises were also ignored by relying solely on simulations. Hence, extensive experiments are conducted using an RC car to demonstrate the controller under an environment closer to real situation.

*A. Experiment Setup*

The RC car used is a Tamiya 1:10 scale car as shown in Fig. 8. The car has a ‘Raspberry Pi’ computer on-board receiving the control PWM signals from the control PC via Wi-Fi. A ROS framework is used for communication. The position of the car is captured in realtime using a camera-based motion capture system called ‘Motive’. The motion capture system runs at

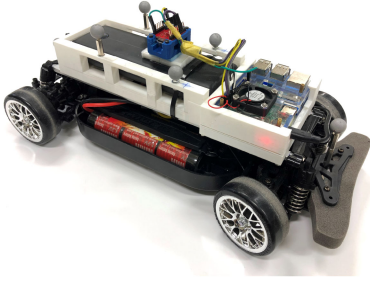


Fig. 8. 1:10 RC car with control board and markers for motion capture.

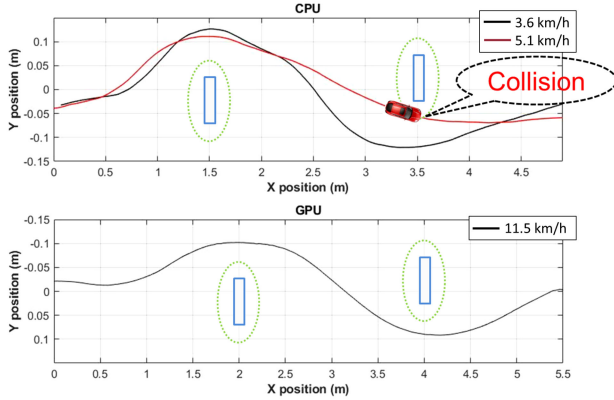


Fig. 9. Speed limit comparison.

200 Hz frequency which is more than the maximum control frequency. The prediction horizon length  $N$  is chosen to be 30 (to predict 3 metres ahead).

### B. Advantages of Higher Control Frequency

One of the advantages of faster computation using GPU is to run the controller at a higher frequency. This enables the obstacle avoidance possible at higher speeds than the CPU. Under the condition of  $N = 30$ , and minimum  $N_s$  of 500, the maximum control frequency using CPU is 30 Hz. This controller is found to work fine at a speed of 3.6 km/h.<sup>1</sup> But, the CPU based system is found to collide with the obstacles at a speed of 5.1 km/h. This is due to the fact that CPU based implementation cannot run at a higher control frequency. GPU was able to run at 200 Hz with 1000 samples, The GPU performs smooth avoidance until 11.5 km/h. The details of the experiment is shown in Fig. 9 and the video [35].

### C. Advantages of Higher Sample Sizes

Another advantage of using the higher computation speed is the capability of considering more samples at every control interval. As discussed earlier in Section III-B, a higher sample number is expected to bring the solutions closer to the global optimum. In the RC car based tests, the control frequency is set to be 100 Hz and the number of samples were increased. The results are indicated in Table III and Fig. 10. At 100 Hz, CPU can

<sup>1</sup>In a real car, this correspond to  $3.6 \times 10 = 36$  km/h [36].

TABLE III  
COST FUNCTION AT DIFFERENT SAMPLE SIZES

Sample Number	Cost ( $\times 10^4$ )	Improvement over CPU (%)
100	16.30	—
500	13.62	16.4
1000	12.43	23.7
5000	11.19	31.4
10000	10.37	36.4
20000	9.77	40.1
30000	9.70	40.5

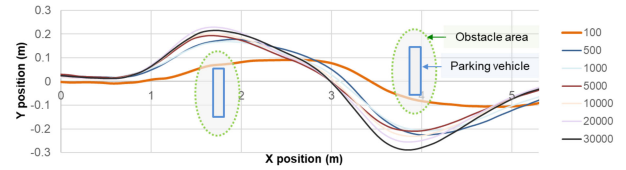


Fig. 10. Sample size comparison.

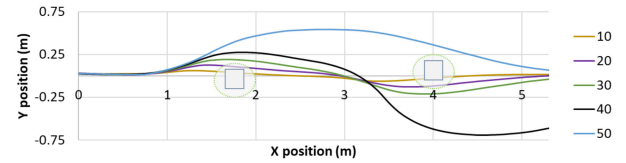


Fig. 11. Controller performance for different prediction horizon lengths.

process only 100 samples in real-time. Using a higher number of samples in real-time, the GPU based controller can have lower cost values. Even though the reduction in cost value is low at higher sample sizes, the car behavior is visibly smoother and maintains better safety distance while driving.

## VI. DISCUSSION ON PARAMETER SELECTION FOR RMPC

RMPC controllers are characterized by the samples used and the frequency at which they are updated. The samples are characterized by the number of samples  $N_s$  and length of each sample  $N$ . Let us denote the control interval as  $dT$ . The selection of these parameters is not as easy as it seems due to their relationship with each other. Increasing  $N_s$  or  $N$  forces  $dT$  to increase. For a fixed  $N$ , there could be multiple combinations of  $N_s$  and  $dT$  that satisfy computational limits.

The recommended procedure to identify these parameters starts with the identification of  $N$ .  $N$  is decided first because of its unique effect on control behavior. Value of  $N$  has to be bounded with both upper and lower limits as seen in Fig. 11. On the other hand, acceptable values of  $N_s$  and  $dT$  could only be bounded on one side, minimum  $N_s$  and maximum  $dT$ .

$N$  can be identified with a few experiments as demonstrated in Fig. 11. We choose the values for  $N_s$  and  $dT$  intuitively for this step. It can be seen that the RC car, while performing the obstacle avoidance task as explained in Section V, has an optimal range of prediction horizon length  $N$  for best tracking performance. Here, the best  $N$  value was found to be 30 steps.

Once we identify the value of  $N$ , it is recommended to choose the optimal  $dT$  corresponding to the speed of the system. Rule of thumb is that a car moving at double speed will cover double the



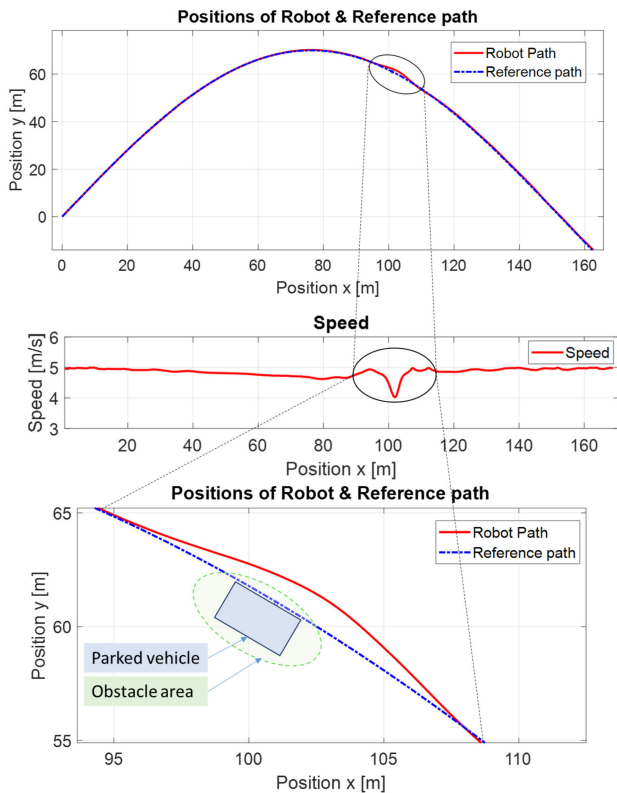


Fig. 12. Tracking a curved path with speed control.

distance in the given time. Hence control frequency should also be doubled. It is always recommended to have some tolerance in the  $dT$  value considering the computational time slightly varies depending on situation.

After choosing the  $N$  and  $dT$ , it is recommended that the maximum sample number is chosen considering given computational resources. This is because the cost function is found to decrease monotonously as the sample size increases as seen in Table III. This method of selection was employed during parameter selection for the RC car experiments and simulations presented in this work.

## VII. EXTENSION TO COOPERATION WITH SPEED CONTROL ON CURVY ROADS

The simulations and experiments discussed until Section VI deal with a constant speed obstacle avoidance task on a straight road. Such a task is found to be the best choice to highlight the effects of various control parameters and computational speed on the proposed RMPC controller.

This controller works equally well in any kind of road, even with those having tight turns. Fig. 12 demonstrates the ego car maneuvering a tight turn and facing a parked car at the exit of the curve. The control parameters are chosen to match with Section III-B. The ego car is observed to perform smooth collision avoidance.

The constant speed assumption is replaced with the addition of a speed controller that slows the car down when higher steering

angles are commanded. The input vector (7) becomes

$$u_t(k) = [\delta^*(k), v(k)]^T \quad (30)$$

Where,

$$v(k) = C_f(v_{avg}(k)), \quad v_{avg}(k) = \frac{1}{n} \sum_{i=1}^n v(k-i).$$

$C_f$  is a cubic function of  $v_{avg}$  and  $n = 10$ . The speed profile during the collision avoidance is seen to be smooth in Fig. 12. The proposed lateral controller being independent, speed control can be easily replaced with any other type of controller.

## VIII. CONCLUSION

This paper has presented a randomized nonlinear model predictive controller for autonomous driving while giving special emphasis to obstacle avoidance task. Although the vehicle model and the obstacle constraints have nonlinearity in their nature, the proposed scheme enabled direct consideration of nonlinear constraints by using a randomized optimization method. Random samples for optimization were generated from the frequency domain using IDCT method. This prevents undesirable oscillation in the control input, which is particularly important in the autonomous driving. This paper also investigated the impact of using GPU for implementation of randomized MPC. By using GPU, the real-time performance limit was extended. A parameter selection methodology for randomized MPC has also been discussed. The proposed scheme and improvements were confirmed in simulation and experiments using an RC-car. The RC-car experiment has realized higher driving speeds and shown better control performance compared with CPU based controller. Our proposed implementation scheme for GPU is available for other types of control systems which are in need of high control performance under nonlinear constraints.

## REFERENCES

- [1] M. Oshima and M. Ogawa, "Model predictive control-I: Basic principle: History & present status," *Trans. Inst. Sys., Cont. Inf. Eng.*, vol. 46, no. 5, pp. 286–293, 2002.
- [2] M. Kano and M. Oshima, "Model predictive control-II: Linear model predictive control," *Trans. Inst. Sys., Cont. Inf. Eng.*, vol. 46, no. 7, pp. 418–424, 2002, (in Japanese).
- [3] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Trans. Control Sys. Tech.*, vol. 15, no. 3, pp. 566–580, May 2007.
- [4] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.
- [5] J. L. Piovesan and H. G. Tanner, "Randomized model predictive control for robot navigation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 94–99.
- [6] G. Schildbach, G. C. Calafiore and L. Fagiano, "Randomized model predictive control for stochastic linear systems," in *Proc. Amer. Control Conf.*, 2012, pp. 417–422.
- [7] M. Vidyasagar, "Randomized algorithms for robust controller synthesis using statistical learning theory," *Automatica*, vol. 37, no. 10, pp. 1515–1528, 2001.
- [8] D. D. Dunlap, E. G. Collins, Jr., and C. V. Caldwell, "Sampling based model predictive control with application to autonomous vehicle guidance," in *Proc. Florida Conf. Recent Adv. Robot.*, 2008.

- [9] G. Ripaccioli, D. Bernardini, S. Di Cairano, A. Bemporad, and I. V. Kolmanovskiy, "A stochastic model predictive control approach for series hybrid electric vehicle power management," in *Proc. Amer. Control Conf.*, 2010, pp. 5844–5849.
- [10] X. Zhang, S. Grammatico, G. Schildbach, P. Goulart, and J. Lygeros, "On the sample size of random convex programs with structured dependence on the uncertainty," *Automatica*, vol. 60, pp. 182–188, 2015.
- [11] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Conf. Intell. Transp. Syst.*, vol. 1, no. 1, pp. 33–55, Mar. 2018.
- [12] J. P. Laumond, P. E. Jacobs, M. Taix, and R. M. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Trans. Robot. Automat.*, vol. 10, no. 5, pp. 577–593, Oct. 1994.
- [13] L. Han, H. Yashiro, H. T. N. Nejad, Q. H. Do, and S. Mita, "Bezier curve based path planning for autonomous vehicle in urban environment," in *Proc. IEEE Intell. Veh. Symp.*, 2010, pp. 1036–1042.
- [14] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.
- [15] S. A. Arogeti and N. Berman, "Path following of autonomous vehicles in the presence of sliding effects," *IEEE Trans. Veh. Technol.*, vol. 61, no. 4, pp. 1481–1492, May 2012.
- [16] A. Koga *et al.*, "Autonomous lane tracking reflecting skilled/un-skilled driving characteristics," in *Proc. 42th IEEE Annual. Conf. IEEE Ind. Electron. Soc.*, 2015, pp. 3175–3180.
- [17] H. Okuda, Y. Liang, and T. Suzuki, "Sampling based predictive control with frequency domain input sampling for smooth collision avoidance," in *Proc. 21st Int. Conf. Intell. Transp. Syst.*, 2018, pp. 2426–2431.
- [18] H. Okuda, N. Sugie, and T. Suzuki, "Real-time collision avoidance control based on continuation method for nonlinear model predictive control with safety constraint," in *Proc. Asian Control Conf.*, 2017, pp. 1086–1091.
- [19] C. Gotte, M. Keller, C. Hass, K. H. Glander, A. Seewald, and T. Bertram, "A model predictive combined planning and control approach for guidance of automated vehicles," in *Proc. IEEE Int. Conf. Veh. Electron. Saf.*, 2015, pp. 69–74.
- [20] Y. Gao, A. Gray, H. Eric Tseng, and F. Borrelli, "A tube-based robust nonlinear predictive control approach to semi autonomous ground vehicles," *Veh. Syst. Dyn.* 52, no. 6, pp. 802–823, 2014.
- [21] M. Garland *et al.*, "Parallel computing experiences with CUDA," *IEEE Micro.*, vol. 28, no. 4, pp. 13–27, Jul./Aug. 2008.
- [22] A. Brooks, T. Kaupp, and A. Makarenko, "Randomised MPC-based motion-planning for mobile robot obstacle avoidance," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 3962–3967.
- [23] J. V. Carrau, A. Liniger, X. Zhang and J. Lygeros, "Efficient implementation of randomized MPC for miniature race cars," in *Proc. Eur. Control Conf.*, 2016, pp. 957–962.
- [24] A. K. Sampathirao, P. Sopsakis, A. Bemporad, and P. P. Patrinos, "GPU-accelerated stochastic predictive control of drinking water networks," *IEEE Trans. Control Syst. Technol.*, vol. 26, no. 2, pp. 551–562, Mar. 2018.
- [25] Phung, D.K., B. Hérisse, J. Marzat, and S. Bertrand, "Model predictive control for autonomous navigation using embedded graphics processing unit," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11883–11888, 2017.
- [26] Y. Gang and L. Mingguang, "Acceleration of MPC using graphic processing unit," in *Proc. 2nd IEEE Int. Conf. Comput. Sci. Netw. Technol.*, 2012, pp. 1001–1004.
- [27] S. Ohyama and H. Date, "Parallelized nonlinear model predictive control on GPU," in *Proc. 11th IEEE Asian Control Conf.*, 2017, pp. 1620–1625.
- [28] K. M. Abughalieh and S. G. Alawneh, "A survey of parallel implementations for model predictive control," *IEEE Access*, vol. 7, pp. 34348–34360, 2019.
- [29] A. Muraleedharan, H. Okuda, and T. Suzuki, "Path tracking control using model predictive control with on GPU implementation for autonomous driving," *J. Arid Land Stud.*, vol. 28, no. 5, pp. 163–167, 2018.
- [30] A. Muraleedharan, H. Okuda, and T. Suzuki, "Improvement of control performance of sampling based model predictive control using GPU," in *Proc. IEEE Intell. Veh. Symp.*, 2019, pp. 1999–2004.
- [31] M. Abe, *Vehicle Handling Dynamics: Theory and Application*, 2nd ed. Butterworth-Heinemann, 2015, pp. 52–56.
- [32] "The C++ numerics library," [Online]. Available: <https://en.cppreference.com/w/cpp/numeric>
- [33] "NVIDIA Corporation The nvidia cuda random number generation library (cuRAND)," [Online]. Available: <https://developer.nvidia.com/curand>
- [34] "NVIDIA Corporation. CUDA C++ Programming Guide," [Online]. Available: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>
- [35] A. Muraleedharan, "Real-time implementation of randomized model predictive control for autonomous driving," YouTube, Apr. 2020. Accessed: Apr. 30, 2020. [Online]. Available: [https://youtu.be/Mb-3\\_PnUdJ8](https://youtu.be/Mb-3_PnUdJ8)
- [36] S. Brennan and A. Alleyne, "Using a scale testbed: Controller design and evaluation," *IEEE Control Syst. Mag.*, vol. 21, no. 3, pp. 15–26, Jun. 2001.



**Arun Muraleedharan** was born in Kerala, India, in 1993. He received the B. Tech. degree in production engineering from the National Institute of Technology Calicut, India, in 2014. He received the M.S. degree in mechanical systems engineering from Nagoya University, Japan, in 2019. He was a Research Engineer with Honda R&D India Pvt. Ltd. from 2014 to 2017. He is currently a doctoral student with the Mechanical Systems Engineering Department of Nagoya University, Japan. His work focuses on path planning and tracking algorithms for autonomous driving. He currently studies stochastic MPC for autonomous driving in shared roads. It is his goal to help the smooth transition to autonomous driving while maintaining the fun factor of driving. He enjoys riding his motorcycle and playing percussion in his free time. He is a member of JSAE.



**Hiroyuki Okuda** (Member, IEEE) was born in Gifu, Japan, in 1982. He received B.E. and M.E. degrees in advanced science and technology from Toyota Technological Institute, Japan, in 2005 and 2007, respectively. He received the Ph.D. degree in mechanical science and engineering from Nagoya University, Japan, in 2010. He was the PD Researcher with the CREST, JST from 2010 to 2012, and an Assistant Professor with Nagoya University's Green Mobility Collaborative Research Center from 2012 to 2016. He was a Visiting Researcher with the Mechanical Engineering Department of U. C. Berkeley in 2018. He is currently an Assistant Professor with the Department of Mechanical Science and Engineering, Nagoya University. His research interests include the areas of system identification of hybrid dynamical system and its application to the modeling and analysis of human behavior and human-centered system design of autonomous/human-machine cooperative system. Dr. Okuda is a member of IEEE, SICE, and JSME.



**Tatsuya Suzuki** (Member, IEEE) was born in Aichi, Japan, in 1964. He received the B.S., M.S., and Ph.D. degrees in Electronic Mechanical Engineering from Nagoya University, Japan, in 1986, 1988, and 1991, respectively. From 1998 to 1999, he was a Visiting Researcher of the Mechanical Engineering Department of U.C. Berkeley. He is currently a Professor of the Department of Mechanical Systems Engineering, Executive Director of Global Research Institute for Mobility in Society (GREMO), Nagoya University. He also has been a Principal Investigator in JST, CREST in 2013–2019. He won the Best Paper Award in International Conference on Autonomic and Autonomous Systems 2017 and the outstanding paper award in International Conference on Control Automation and Systems 2008. He also won the Journal Paper Award from IEEE, SICE and JSAE in 1995, 2009, and 2010, respectively. His current research interests include the areas of analysis and design of human-centric intelligent mobility systems, and integrated design of transportation and smart grid systems. Dr. Suzuki is a member of the SICE, ISICE, IEICE, JSAE, RSJ, JSME, IEEE.