2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

RPGD: A Small-Batch Parallel Gradient Descent Optimizer with Explorative Resampling for Nonlinear Model Predictive Control

Frederik Heetmeyer*, *Member, IEEE,* Marcin Paluch*, Diego Bolliger*,
Florian Bolli*, Xiang Deng*, Ennio Filicicchia*, and Tobi Delbruck*, *Fellow, IEEE*

***Abstract*— Nonlinear model predictive control often involves nonconvex optimization for which real-time control systems require fast and numerically stable solutions. This work proposes RPGD, a Resampling Parallel Gradient Descent optimizer designed to exploit small-batch parallelism of modern hardware like neural accelerators or multithreaded microcontrollers. After initialization, it continuously maintains a small population of good control trajectory solution candidates and improves them using gradient information, followed by selection of elite candidates and resampling of the others. In simulation on a cartpole, the OpenAI Gym mountain car, a Dubins car with obstacles, and a high input dimensional 2D arm, it produces similar or lower MPC costs than benchmark cross-entropy and path integral methods. On a physical cartpole, it performs swing-up and cart target following of the pole, using either a differential equation or multilayer perceptron as dynamics model. RPGD drives an F1TENTH simulated race car at near-optimal lap times and a real F1TENTH car in laps around a cluttered room. We study alterations of RPGD's building blocks to justify its composition. RPGD compute time in Python with TensorFlow optimization running on CPU is 2 to 4 times slower than the FORCESPRO commercial embedded solver.**

I. INTRODUCTION

A nonlinear discrete-time control system is described by the difference equation

$$x_{k+1} = \mathrm{f}(x_k, u_k, w_k), \quad w_k \sim p(\mathbf{w})$$

with state $x_k \in \mathcal{X} \subseteq \mathbb{R}^n$, bounded control input $u_k \in \mathcal{U} \subset \mathbb{R}^m$ and disturbance $w_k \in \mathcal{W} \subseteq \mathbb{R}^l$. $\mathcal{X}$, $\mathcal{U}$ and $\mathcal{W}$ is each a connected constrained space. The current measured state $\hat{x}_k$ is estimated from observations.

Nonlinear Model Predictive Control (**NMPC**) computes a feedback control input $u_k = \mu(\hat{x}_k)$ by minimizing cost J over a finite horizon of N steps. Let $\bar{u}_{k:k+N-1}$ denote an input plan along the MPC horizon of N timesteps starting from control iteration k, and $\hat{x}_{k:k+N}$ the resulting state trajectory according to the approximated model $\hat{\mathrm{f}}$ of system f. Then, the solution control input plan $\nu_{k:k+N-1}$ is

$$\nu_{k:k+N-1} = \underset{\bar{u}_{k:k+N-1} \in \mathcal{U}^N}{\operatorname{argmin}} \left[\mathrm{J}(\hat{x}_{k:k+N}, \bar{u}_{k:k+N-1}) \right] \quad (1)$$

subject to

$$\hat{x}_{k+i+1} = \hat{\mathrm{f}}(\hat{x}_{k+i}, \bar{u}_{k+i})$$
$$\hat{x}_{k+i} \in \mathcal{X}, \quad \bar{u}_{k+i} \in \mathcal{U}, \quad i = 0, \ldots, N-1$$

The control taken for the next time step is then $\mu(\hat{x}_k) = \nu_k$.

* Sensors Group, Institute of Neuroinformatics, University of Zurich and ETH Zurich; https://sensors.ini.uzh.ch. This work started from the semester project of D. Bolliger supervised by M. Paluch.

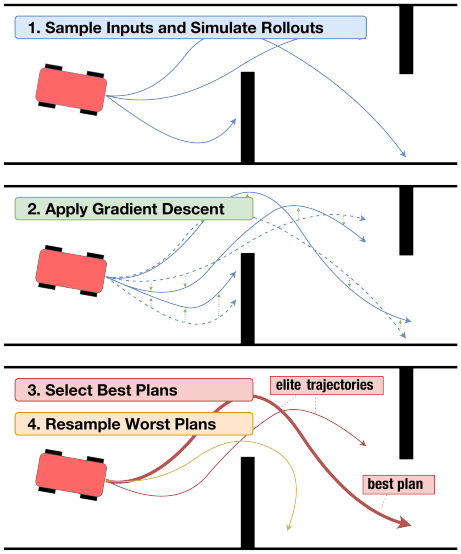


Fig. 1: A simplified visualization of our proposed RPGD optimizer for NMPC problems. A car navigates two obstacles. RPGD first randomly samples a set of input plans (here 3 plans). It simulates state trajectories from them using a model of the system. Next, it gradient-optimizes each plan with respect to cost. Finally, the best few (here 2) 'elite' control plans are retained for the next step, and the remaining ones are resampled. The elite plans thus receive repeated gradient optimization folded over control steps. At each control step, the first control of the best plan (in bold) is applied to the system.

The cost function J is crafted to obtain a desired objective. It is often quadratic and positive definite, but it may contain other nonlinear forms.

A major interest lies in efficiently solving the optimization problem in (1). In general there exists no closed-form solution. Current NMPC solvers are dominated by convexifying sequential linearization methods such as Sequential Quadratic Programming (**SQP**) [1]. These algorithms were developed for sequential computers and work well for many problems, but not all. The availability of neural network hardware [2] has increased interest in optimization algorithms that do not have the limitations of convex solvers and combine the advantages of parallel and iterative approaches. Fig. 1 illustrates how our proposed **Resampling Parallel Gradient Descent (RPGD)** method fulfills these criteria.

Gradient descent NMPC optimization [3] initializes a plan $\bar{u}_{k:k+N-1}$ of control inputs, for example by random sampling. Then the total gradient of the horizon's cost $\frac{\mathrm{dJ}}{\mathrm{d}\bar{u}}$ is computed. First-order gradient methods modify the input trajectory proportionally to the direction of steepest

cost decrease. This procedure is repeated multiple times. Typically, gradient optimizers terminate either after a fixed number of steps, if the cost has stopped decreasing, or if an optimality condition such as Karush-Kuhn-Tucker [4] is fulfilled. However, such procedures often converge to only locally optimal plans. Moreover, physical systems have limited compute time, which constrains the number of sequential optimizations.

Another family of solution approaches optimizes by sampling many solution candidates, for example Model Predictive Path Integral (**MPPI**) [5], which uses real time parallel Graphics Processing Unit (**GPU**) evaluation of thousands of rollouts per control step [6]. These powerful GPUs are expensive and burn tens to hundreds of watts.

Therefore, most of the optimizers currently available for NMPC either have limitations for highly nonlinear systems or require expensive and power-hungry computers. Our work is motivated by the desire to limit the resources consumed for running on embedded hardware, to allow a robot to run cooler and be smaller and cheaper.

NMPC optimization requires evaluations of the system model. When the system's governing equations cannot be derived analytically, a neural network can approximate its dynamics. This network can be evaluated on a hardware accelerator such as the Recurrent Neural Network (**RNN**) accelerators [7]–[9] developed by our lab, which can compute large RNNs much more quickly than desktop GPUs at a small fraction of the power. We consider such hardware accelerators to be integral to future embedded NMPC. However, limited on-chip memory means that only a small number (*e.g.*, 32) of rollouts can be computed in parallel, while powerful desktop GPUs can evaluate thousands of trajectories in parallel. The RPGD optimizer achieves good control with small sampling populations.

II. RPGD METHOD

The proposed RPGD method is sketched in Fig. 1 and detailed in Fig. 2. Table II lists RPGD hyperparameters for our experiments if not mentioned otherwise.

In step (1.), RPGD independently samples a population of P_{total} candidate input plans $\bar{u}^{(p)}_{k:k+N-1} \in \mathcal{U}^N, p \in \{1, \ldots, P_{\text{total}}\}$, from a fixed distribution by default chosen to be uniform over $\mathcal{U}$. This sampling ensures exploration of all potential inputs. Each plan describes an open-loop sequence of controls over the MPC horizon of N steps, and produces a total cost $\mathrm{J}^{(\mathrm{p})}$ according to model $\hat{\mathrm{f}}$.

(2.) is a gradient optimization loop: The Adam optimizer [10] is applied to improve each input plan using the gradient $\frac{\mathrm{dJ}^{(\mathrm{p})}}{\mathrm{d}\bar{\mathrm{u}}^{(\mathrm{p})}}$, which is calculated by automatic differentiation. Each plan has its own Adam dynamics. This gradient step is repeated G (*e.g.*, 5) times. Sec. IV-H shows that computing and applying the gradient takes about 5X more time than computing the forward rollouts.

Then, in (3.), the first input $\bar{u}^{(\tilde{p})}_k$ of the lowest-cost plan $\tilde{p}$ is selected as the solution. All input plans are then time-shifted for the next iteration.

Finally, in (4.), every $K^{\text{th}}_{\text{re}}$ control iteration k, the elite set of P_{elite} lowest-cost plans is retained, while the remaining $P_{\text{total}} - P_{\text{elite}}$ plans are discarded and replaced by newly sampled ones from the initial distribution. This selection allows exploration of $\mathcal{U}^N$, adjustable by P_{elite}.

A. Contributions and Novelty

Our RPGD optimizer is a novel combination of existing features to make gradient-based NMPC feasible for real-time applications on an embedded platform:

- RPGD performs a small number of sequential operations in small batches. The number of parallel instructions is similar to gradient- and cross-entropy method-based NMPC and much smaller than that of the sampling-based path integral methods. Section IV demonstrates that RPGD achieves better control at a computational cost comparable to that of other methods. RPGD escapes local minima and reacts to changes in the environment.
- RPGD uses the pragmatic assumption that the input plans of the previous control iteration are a good starting point for the next control iteration. Section IV-D.3 shows that controller performance is worse when resampling replaces all plans ($P_{\text{elite}} = 0$) compared to some ($P_{\text{elite}} > 0$).
- Sec. IV-D.1 shows that RPGD improves candidate input plans through folded gradient descent steps, resulting in more gradient descent steps in total than what can be executed within a single control period. This folding is achieved by retaining plans for subsequent control iterations.
- Section IV-H shows that RPGD computed on CPU runs 2-4X slower than the fastest commercial solver *FORCESPRO* but within real-time requirements. Secs. IV-F & IV-E demonstrate real-time control with RPGD on a physical cartpole and F1TENTH car.

III. RELATED WORK

The RPGD optimization procedure combines ideas from previous work in sampling- and gradient-based optimization for NMPC.

A. Pure Gradient MPC

A range of previous works have used gradient information to optimize in the MPC setting. Henaff et al. [11] also differentiated the MPC loss through the system model, and then applied gradient descent, and also suggested optimizing multiple trajectory candidates in parallel. Our inner gradient loop adopts this procedure with the extension of using the Adam optimizer. In contrast to our work, they consider learning a reward model and parameterizing discrete actions, which we omit. We achieve practical control frequencies by distributing the gradient steps of an input plan across multiple control iterations. Richter et al. [12] demonstrate another gradient MPC algorithm which is applied to a linear system at high control frequency on a real-time system.

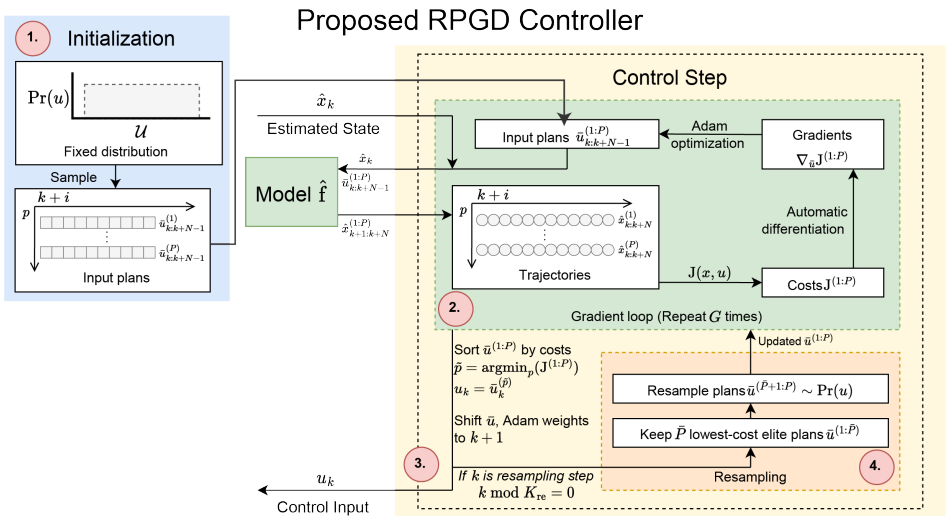


Fig. 2: The proposed RPGD optimizer for NMPC. The steps are described in Sec. II.

B. Cross-Entropy Methods

The Cross-Entropy Method (**CEM**) [13] is a sampling-based optimization scheme. Applications of it to MPC are found in [14] [15] [16]. The CEM, as in this work, also maintains a population of candidate trajectories. In contrast to our work, the result of previous timestep MPC optimization is reused through Monte Carlo importance sampling instead of gradient descent. Our resampling step, where only the best few trajectories are retained, is inspired by the elite set selection of CEM. Bharadhwaj's Gradient-CEM and Huang's CEM-GD [15], [17] also inspire our work by interleaving sampling and gradient descent in policy learning, but for computational efficiency RPGD folds the gradient descent onto the retained plans, while these methods start each timestep with plans drawn from a normal distribution.

C. Model Predictive Path Integral

Williams et al. introduced MPPI, a parallelizable approach to solve an optimal control problem using Monte Carlo importance sampling [5]. Their AutoRally cars convincingly demonstrated MPPI in aggressive autonomous solo racing on gravel tracks [18], [19].

IV. EXPERIMENTAL RESULTS

We compared RPGD control performance with other optimizers (Sec. IV-A) on four simulated environments (Sec. IV-B). We also used RPGD to control a simulated F1TENTH car (Sec. IV-E), a physical cartpole (Sec. IV-F), and a physical F1TENTH car (Sec. IV-G). Source code for reproducing the comparative results and the supplementary video are available[1].

A. Optimizers compared with RPGD

Our experiments compared RPGD in Table I with

- *CEM*: a standard CEM MPC controller that fits a diagonal Gaussian.
- *Gradient-CEM*: Bharadhwaj et al. [15] improves input plans using gradient descent and fits a diagonal Gaussian using the CEM.
- *Gradient MPC*: a population-based gradient descent MPC optimizer. This method is RPGD (our method) without resampling.
- *MPPI*: samples P_{total} input plans and averages them together using negative exponential cost weighting [5].
- *Winner-Take-All*: It samples P_{total} input plans and selects the one with the lowest cost.

To make the computational costs of all optimizers similar, we adjusted the optimizer hyperparameters to give all non-gradient methods a factor of five times more rollouts for every gradient computation of RPGD (see Sec. IV-H). This factor is assigned as larger P_{total} sampling sizes. MPPI and Winner-Take-All get another factor G, in this case 4, of more samples because they do not have an inner gradient loop.

B. Simulation Environments Tested

We compared RPGD control performance with the other optimizers using the following environments, as shown in our Table I results:

1) Continuous mountain car: The continuous mountain car is an OpenAI Gym [20] environment. The goal is to accelerate a cart to the top of a hill. The action space is the closed 1D interval $[-1, 1]$. The optimal strategy follows a bang-bang policy: First, accelerate maximally to the left slope, then flip the input and use the gained momentum to reach the top, where the episode ends. We added to the original singular reward for reaching the target state a continuous term that greedily rewards the altitude gained. This environment demonstrates that RPGD can return a noncontinuous, bang-bang control trajectory.

2) Modified Dubins car: The modified Dubins car is a simulation environment similar to the one in [13]. It is a forward-moving vehicle with a maximum turning rate on a bounded two-dimensional surface. We spawn it on the left side of the surface and set a target position on the right. The target steps vertically to new random positions every 30 time

[1] Link to Simulations Source Code on GitHub

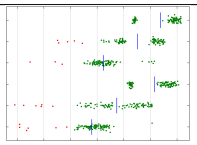
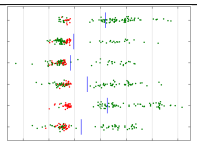
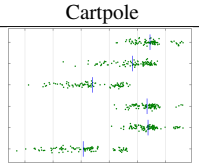
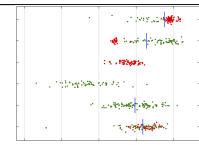
MPC Optimizer	parallel rollouts P_{total}	Mountain Car	Dubins Car	Cartpole	Arm
Proposed RPGD	32				
CEM [13]	160				
Gradient-CEM [15]	32				
Gradient MPC	32				
MPPI [16]	640				
Winner-Take-All	640				
Success Criteria		Arrive at target	Arrive at target	Swingup, reach target	Reach target
Failure Criteria		Timeout	Timeout, exit bound	None	Timeout

TABLE I: Comparative Simulation Results. Shown are the realized rewards across 100 random episodes. More positive reward (to the right) is better. Average is shown with a blue vertical tick, successful episodes as green dots, and failed episodes as red dots. Our proposed MPC optimizer RPGD belongs to the best-performing candidates across all four benchmark environments.

steps. Once the target position is reached, the episode ends. Random circular soft obstacles are spawned for each episode. Unlike the original Dubins planning problem [21], we are interested in navigating the obstacles. There is a continuous cost rewarding L^2-proximity to the target, a quadratic penalty for contact with obstacles, and a discrete value reward for reaching the target position. The challenge for the controller lies in approaching the target on the shortest path, while not getting stuck between obstacles. This experiment shows that RPGD can do motion planning, escape local minima, and explore a state space with few samples.

3) Simulated Cartpole: The simulated cartpole uses the differential equations of Green [22]. They are integrated using the standard Euler method with intermediate time steps. The pole is randomly initialized within $\pm\pi/4$ of the downward hanging position. The goal is to swing up and control the cart to a moving target cart position using only horizontal cart motion. The cost function contains the cosine of the angle of the pole plus a quadratic penalty for the cart position target distance. This experiment shows that RPGD can perform non-linear control to achieve a target far into the future.

4) Simulated Arm: To study whether RPGD can plan in a high-Degree Of Freedom (**DOF**) space, we simulated a 20-DOF 2D arm, with one obstacle (the large cyan disc) that the arm cannot touch. We formulate the inverse kinematics problem as an optimization problem. Its solution results in a control sequence of the arm joint angle changes so that the tip moves from its current position to a target position (the red dot). The control input at each step is the change in each joint angle subject to a predefined maximum change rate. The cost function sums four terms: *1:* Manhattan distance between the current and target tip positions, exponentially discounted over the horizon. *2:* Angle difference between neighboring segments. *3:* Self-crossing; tip crossing any segment of the arm incurs a large cost. *4:* Highly punitive obstacle cost for any joint touching the obstacle. In Table I some episode rewards are cropped off because they are highly negative and correspond to episode failures.

C. Comparative Control results

Table I summarizes the results for the 100 random trials of each system, using a consistent random seed for the trials for each method. The results show that for each system, RPGD consistently achieves a control reward higher than or comparable to the other methods. For the Mountain Car, RPGD slightly outperforms the pure gradient optimizer. On the Dubins car, RPGD performs similarly to MPPI. For the cartpole, RPGD, the pure gradient optimizer, and MPPI have similar control rewards. On the arm, RPGD outperforms other methods on average but with higher timeout rate, meaning it often does not quite reach the target within 1 segment length.

D. Variation of Hyperparameters

We now examine how the individual components of RPGD contribute to its performance by varying its hyperparameters, including ablation of individual components. We intend to justify our novel combination of population gradient descent, periodic resampling, and interpolation of sampled actions. We selected the Dubins car as the test bench. All unvaried hyperparameters are set to the values in Table II. If not specified otherwise, we measure performance as the average reward realized during an experiment, where a more positive reward means better control performance. We ran 100 random trials with consistent seeding. Each point in the scatter plots in Figs. 3&5 shows the average over all timesteps of the realized reward for one episode. Results across episodes vary from the initialization of the random environment and the random sampling of inputs.

1) Number of gradient steps G: Fig. 3 shows that controller performance improves with increased gradient steps G and starts to saturate with about 25 steps. We choose $G = 4$ for further experiments because this value is feasible for our real time robotic implementations.

2) Resampling interval K_{re}: The optimizer replaces the highest-cost-producing input plans with new samples every $K_{\text{re}}^{\text{th}}$ control step. Resampling is particularly beneficial when the environment changes (such as a target position) or exerts disturbances onto the state [19]. We studied the Dubins car simulation and randomly varied the target position vertically every 30 time steps. Our results show that smaller K_{re} results

TABLE II: Hyperparameters of Proposed RPGD Method

Parameter	Value
Population Parameters	
Population Size P_{total}	32
Retained Elite Plans P_{elite}	8
Resampling Interval K_{re}	10
Gradient Parameters	
Gradient Steps per Control Step G	4
Adam Optimizer	
Learning Rate	0.05
β_1, β_2, ε	0.9, 0.999, 10^{-8}
Sampling Parameters	
Interpolation Interval K_{interp}	5 Steps
MPC Horizon Length N	40 Steps
Episode Length E	200 Steps

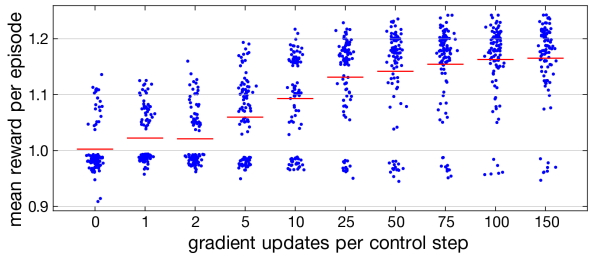
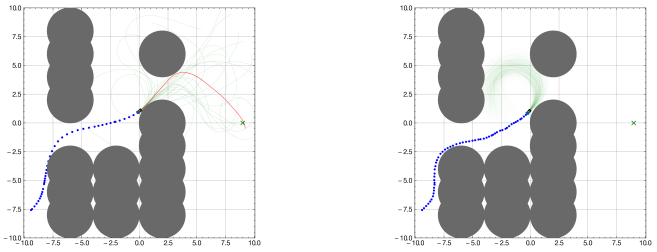


Fig. 3: Realized rewards of RPGD applied to 100 randomized episodes of the Dubins car, with varying number of gradient iterations G per control step. Increasing G tends to increase reward.

in slightly better realized rewards than larger resampling intervals. A case for resampling is made in Fig. 4.



(a) RPGD (b) MPPI

Fig. 4: Dubins car samples: X marks the target. **a)** Because RPGD resamples input plans (in this case $K_{\text{re}} = 10$), it can escape local minima quickly; the plan with highest reward is highlighted. **b)** MPPI adjusts its nominal input plan only incrementally. By the time a new, better optimum could be reached within the planning horizon, the nominal path cannot adjust abruptly enough.

3) Population sizes P_{total} and retained plans P_{elite}: Fig. 5 shows that reward increases with P_{total} and that retaining an elite number P_{elite} of plans helps escape local minima. We vary between $P_{\text{total}} = 1$ (a single rollout) and $P_{\text{total}} = 256$. P_{elite} is set to $P_{\text{total}}/4$. Two additional cases ($P_{\text{total}} = 32, P_{\text{elite}} = 0$ and $P_{\text{total}} = 32, P_{\text{elite}} = 32$) are also shown. Here, we can see that keeping a quarter of samples ($P_{\text{elite}} = 8$) tends to result in more positive reward than replacing all plans ($P_{\text{elite}} = 0$) or not resampling at all ($P_{\text{elite}} = 32$). This result supports the claim in Section II-A that a mixture between a previous iteration's set of input plans and some new random plans is a better initialization for the optimization than those two alternatives.

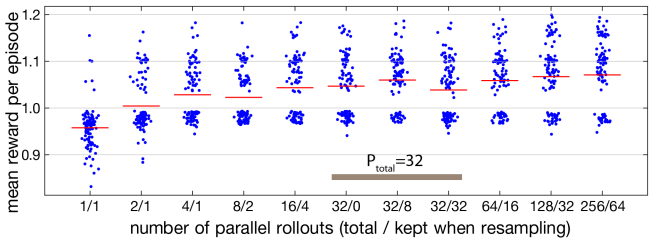


Fig. 5: Increasing the number of RPGD parallel rollouts ("population size") P_{total} steers the Dubins car around obstacles faster and gets stuck less between obstacles. Thus realized reward on 100 randomized episodes tends to increase with P_{total}. For this comparison, we generally chose that a quarter of input plans are kept when resampling. For $P_{\text{total}} = 32$, we isolate two additional cases.

E. Simulated F1TENTH Car

We used RPGD[2] to control a simulated F1TENTH car[3]. We achieved a 2-lap time of 38.73 ± 0.08 s averaged over 50 trials. The winner of the 2022 ICRA F1TENTH virtual competition achieved 50.6 s on this track. Our supplementary video shows a typical pair of laps along with the RPGD rollouts. The car drives very aggressively and drifts through many turns. This experiment demonstrates that RPGD can effectively control a car dynamics model with a 2D input (steering and throttle) and a 7D state space.

F. Physical Cartpole Experiments

To demonstrate RPGD practicality, we used it to control a $200 physical cartpole[4]. We tested RPGD using two different models: 1) an Euler stepped Ordinary Differential Equation (**ODE**) cartpole model where the model parameters are estimated from offline measurements, and 2) a tiny 2-layer Multilayer Perceptron (**MLP**), which we trained on repeated simulated swing-ups and swing-downs at randomly changing cart target positions[5]. The MLP input is the current state and cart motor command, and the output is the next state 20 ms ahead. Our experiments used an RPGD with control frequency of about 50 Hz, population size of $P_{\text{total}} = 32$, elite set size $P_{\text{elite}} = 8$, and MPC horizon $N = 35$ (700 ms).

Our supplementary video[6] shows that the cartpole stabilizes upright, swings up, and follows a moving cart position target. Fig. 6 compares pole angle and cart position over time. With both models, the cartpole executes a swing up and tries to track a changing target cart position (dashed position trace). RPGD works with both models, but here using the MLP model, it tracks the target cart position better.

[2]Parameters: control frequency = 33Hz, MPC horizon $N = 15$ (450 ms)

[3]Link to F1TENTH Home Page

[4]Link to Linear Inverted Pendulum - AliExpress

[5]The MLP has architecture 6IN-32H1-32H2-5OUT with hyperbolic tangent activation functions. It is trained on 10 recordings, 360s each, split into training/validation/testing: 8/1/1. Data generation used NMPC with MPPI optimizer, and added noise to control input to augment the data. The network during training was seeing the noisy control input effectively acting on the system, not the ideal one from MPPI.

[6]Link to ICRA 2023 supplementary video submission

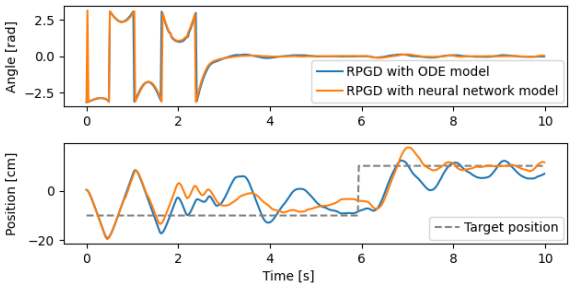


Fig. 6: Physical cartpole RPGD state data comparing control based on ODE and MLP dynamics models.

G. Physical F1TENTH car

We compared RPGD[7] and MPPI for controlling our physical F1TENTH race car. We ran the control on an external laptop connected to the car as a ROS node.

Fig. 7 compares trajectories of the car with RPGD and MPPI optimizer. Both optimizers complete 5 laps of the room following precalculated waypoints and using a target speed objective. The results are very close; MPPI might be slightly faster by staying a bit closer to the optimal trajectory, but its lap time variance is larger.

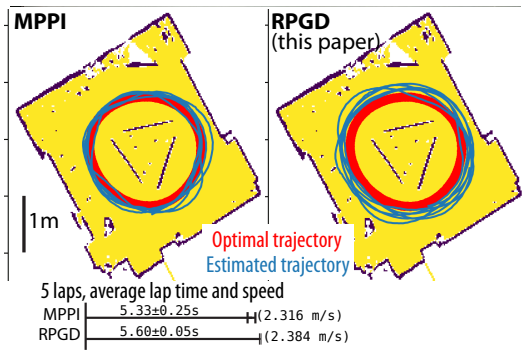


Fig. 7: Physical F1TENTH car trajectories with MPPI and RPGD.

H. Compute Time Comparisons

We want to use RPGD for optimal control in real time. To measure its compute time, we performed our experiments in Python 3.9 using TensorFlow 2.10. We did not use GPU computing because it was not necessary for the modest parallelism of RPGD. We applied TensorFlow graph compilation and Accelerated Linear Algebra (**XLA**) optimization for all experiments. With these optimizations, execution times are about 20X faster. We compared the wall clock compute times of RPGD with the popular commercial NMPC solver *FORCESPRO*[8]. For *FORCESPRO*, we used the Nonlinear Primal-Dual Interior-Point solver method because it provided the most reliable solutions. The comparison was made in the environments MountainCar and Pendulum, the latter being a simple pendulum actuated by a saturating motor at its base, which tries to swing up and stay upright. We did these experiments on a laptop computer[9].

Fig. 8 shows that RPGD is 2X-4X slower than *FORCESPRO*, the fastest solver for control problems available in the market. The breakdown of RPGD timing shows that the gradient steps in RPGD cost 5X the time of the rollouts.

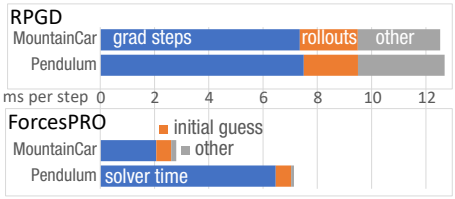


Fig. 8: RPGD vs FORCESPRO computation time.

V. CONCLUSION

Our results indicate that the construction of RPGD suits the application to NMPC. Qualitatively, the set of input plans adapts in an intuitively favorable way: During selection, low-cost input plans are preserved. They can be further optimized with respect to cost in the next iteration. High-cost input plans are replaced when they do not show promise within K_{re} control steps. The population sampling size and the number of gradient steps are the most important for its performance. Furthermore, the resulting controller produces near-optimal trajectories in simulation and physical robots.

Notwithstanding the results, RPGD has limitations. To use RPGD, the cost function $\mathrm{J}^{(\mathrm{p})}$ must contain terms differentiable with respect to x and u. It is still possible to incorporate sharp constraints into the cost function because the selection and resampling step considers them. The requirement of a differentiable cost implies a requirement of differentiable model, such as systems modeled as neural network or ODE. However, discrete control inputs would need to be mapped onto a continuous interval, and the discrete value could then be selected by quantizing the continuous value. RPGD is not suitable when the cost function is mainly composed of indicator step function terms because they have zero gradient.

We proposed RPGD as a practical optimization procedure for NMPC and show that it outperforms related control methods. Its partition of gradient steps across control iterations allows it to run in real time on conventional CPUs. In addition, it fulfills the requirements for embedded computation, where the system model is a neural network and is evaluated on a hardware accelerator.

ACKNOWLEDGMENTS

This work was supported by the Swiss National Center for Robotics (NCCR Robotics). We thank Prof. Manfred Morari (ETH Zurich & UPenn), Ante Marić (U Zagreb), Prof. Aaron Ames, Noel Csomay-Shanklin and Maegan Tucker (Caltech), Jérôme Jeannin, Nikita Voinov and Constantin Koch (ETH Zurich) for their help. We thank the ICRA reviewers for their helpful comments that led to this final version of the paper.

[7]We used the following RPGD parameters for the physical F1TENTH car. Control Frequency=12.5Hz, population size $P_{\text{total}} = 32$, elite set size $P_{\text{elite}} = 24$, learning rate = 0.1 and MPC horizon $N = 10$ (800 ms)

[8]Link to Embotech FORCESPRO Overview

[9]AMD Ryzen 4800h CPU @ 2.90GHz, Ubuntu 20. Solver from FORCESPRO used is PDIP_NLP, with N=40, maxit=250, TolStat=1E-2, TolEq=1E-5. Github link to reproduce benchmarking.

REFERENCES

[1] S. V. Rakovic and W. S. Levine, *Handbook of Model Predictive Control*. Springer, 2018.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017, ISSN: 1558-2256. DOI: 10.1109/JPROC.2017.2761740.

[3] K. Graichen and B. Kapernick, "A Real-Time Gradient Method for Nonlinear Model Predictive Control," in *Frontiers of Model Predictive Control*, T. Zheng, Ed., InTech, Feb. 2012, ISBN: 978-953-51-0119-2. DOI: 10.5772/37638.

[4] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of 2nd Berkeley Symposium*, Berkeley CA, USA: University of California Press, 1951, pp. 481–492. [Online]. Available: https://projecteuclid.org/ebooks/berkeley-symposium-on-mathematical-statistics-and-probability/Proceedings%20of%20the%20Second%20Berkeley%20Symposium%20on%20Mathematical%20Statistics%20and%20Probability/chapter/Nonlinear%20Programming/bsmsp/1200500249.

[5] G. Williams, A. Aldrich, and E. A. Theodorou, "Model Predictive Path Integral Control: From Theory to Parallel Computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017. DOI: 10.2514/1.G001921.

[6] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277.

[7] C. Gao, T. Delbruck, and S.-C. Liu, "Spartus: A 9.4 TOp/s FPGA-Based LSTM Accelerator Exploiting Spatio-Temporal Sparsity," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022, ISSN: 2162-2388. DOI: 10.1109/TNNLS.2022.3180209.

[8] C. Gao, A. Rios-Navarro, X. Chen, T. Delbruck, and S. Liu, "EdgeDRNN: Enabling Low-latency Recurrent Neural Network Edge Inference," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Aug. 2020, pp. 41–45. DOI: 10.1109/AICAS48895.2020.9074001.

[9] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, "DeltaRNN: A Power-efficient Recurrent Neural Network Accelerator," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey CALIFORNIA USA: ACM, Feb. 2018, pp. 21–30, ISBN: 978-1-4503-5614-5. DOI: 10.1145/3174243.3174261.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. arXiv: 1412.6980.

[11] M. Henaff, W. F. Whitney, and Y. LeCun, *Model-Based Planning with Discrete and Continuous Actions*, Apr. 2018. arXiv: 1705.07177 [cs].

[12] S. Richter, S. Mariethoz, and M. Morari, "High-speed online MPC based on a fast gradient method applied to power converter control," in *Proceedings of the 2010 American Control Conference*, Baltimore, MD: IEEE, Jun. 2010, pp. 4737–4743, ISBN: 978-1-4244-7427-1. DOI: 10.1109/ACC.2010.5531095.

[13] M. Kobilarov, "Cross-entropy randomized motion planning," in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, Jun. 2011. DOI: 10.15607/RSS.2011.VII.022.

[14] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models," *arXiv:1805.12114 [cs, stat]*, Nov. 2018. arXiv: 1805.12114 [cs, stat].

[15] H. Bharadhwaj, K. Xie, and F. Shkurti, "Model-predictive control via cross-entropy and gradient-based optimization," *Learning for Dynamics and*, 2020. [Online]. Available: http://proceedings.mlr.press/v120/bharadhwaj20a.html.

[16] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, Dec. 2018, ISSN: 1941-0468. DOI: 10.1109/TRO.2018.2865891.

[17] K. Huang, S. Lale, U. Rosolia, Y. Shi, and A. Anandkumar, "CEM-GD: Cross-Entropy method with gradient descent planner for Model-Based reinforcement learning," Dec. 2021. arXiv: 2112.07746 [cs.LG]. [Online]. Available: http://arxiv.org/abs/2112.07746.

[18] G. Williams, N. Wagener, B. Goldfain, *et al.*, "Information theoretic MPC for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1714–1721. DOI: 10.1109/ICRA.2017.7989202.

[19] G. Williams, B. Goldfain, P. Drews, K. Saigol, J. Rehg, and E. Theodorou, "Robust Sampling Based Model Predictive Control with Sparse Objective Information," in *Robotics: Science and Systems XIV*, Robotics: Science and Systems Foundation, Jun. 2018, ISBN: 978-0-9923747-4-7. DOI: 10.15607/RSS.2018.XIV.042.

[20] G. Brockman, V. Cheung, L. Pettersson, *et al.*, *OpenAI Gym*, Jun. 2016. arXiv: 1606.01540 [cs].

[21] L. E. Dubins, "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957, ISSN: 0002-9327. DOI: 10.2307/2372560.

[22] C. Green, *Equations of Motion for the Cart and Pole Control Task*, https://sharpneat.sourceforge.io/research/cart-pole/cart-pole-equations.html, Jan. 2020.