TUNERCAR: A Superoptimization Toolchain for Autonomous Racing

Matthew O'Kelly*,1, Hongrui Zheng*,1, Achin Jain*,1, Joseph Auckley[1], Kim Luong[1], and Rahul Mangharam[1]

***Abstract*— TUNERCAR is a toolchain that jointly optimizes racing strategy, planning methods, control algorithms, and vehicle parameters for an autonomous racecar. In this paper, we detail the target hardware, software, simulators, and systems infrastructure for this toolchain. Our methodology employs a parallel implementation of CMA-ES which enables simulations to proceed 6 times faster than real-world rollouts. We show our approach can reduce the lap times in autonomous racing, given a fixed computational budget. For all tested tracks, our method provides the lowest lap time, and relative improvements in lap time between 7-21%. We demonstrate improvements over a naive random search method with equivalent computational budget of over 15 seconds/lap, and improvements over expert solutions of over 2 seconds/lap. We further compare the performance of our method against hand-tuned solutions submitted by over 30 international teams, comprised of graduate students working in the field of autonomous vehicles. Finally, we discuss the effectiveness of utilizing an online planning mechanism to reduce the reality gap between our simulation and actual tests.**

I. INTRODUCTION

Since its inception, racing has been a key driver of new technology in the automotive industry. Some domains, such as powertrain engineering, are obvious beneficiaries of racing development. However, the goals of racing – fast and aggressive driving – are seemingly orthogonal to the safety-oriented specifications of the autonomous vehicle industry. Nevertheless, scenarios faced by racing drivers (and autonomous racers) force the development of technology which must operate safely in both nominal conditions and, more importantly, at the limits of vehicle performance.

The objective of developing an optimal autonomous racer is motivated by the desire to create safe and reusable core autonomy components, namely vehicle and environment agnostic planning and control software. Racing, in this context, is a mechanism to create a competitive environment where the quality of the chosen vehicle configuration has a clear measure – lap time. While nominal conditions may be handled even with poorly integrated components (*e.g.* a pure pursuit controller which works even when accidentally used on the wrong robot [1]), racing conditions severely punish sub-optimal vehicle dynamics, racing strategy (path and speed selection) and controller parameters.

This paper introduces the notion that component reuse and adaptation is analogous to creating a compiler-like tool that targets computational, physical, and external environmental details of a robot's operational domain. In general, the goal of a compiler is to validate and then transform a source program from one language to another (usually lower-level *e.g.* assembly) which is suitable for the target domain [2]. Modern optimizing compilers [3] also seek to improve the performance of the transformed program. To concretize the analogy, we define the source program as a parameterized description of the vehicle dynamics, tracking controllers, and local planning method which we wish to transform to perform safely and efficiently in the operational environment represented by a map, physical laws, and sensing capabilities.

We propose a solution to this parameter search problem analogous to the concept of *superoptimization* [4], [5], [6], [7], a technique that searches the space of equivalent and correct programs for the most performant instance rather than applying a sequence of optimization passes which attack specific performance bottlenecks. Despite the success of superoptimization approaches in narrow domains [8], [9], viewing the autonomous vehicle as the compilation target creates entirely new issues due to the cyber-physical nature of the platform. First, no simulator or model can perfectly emulate the target thus creating a noisy performance measure and potentially falsifying correctness claims. Second, addressing this reality gap by instead executing the proposed program transformations on the vehicle is dangerous, slow, and expensive. The vehicle may get damaged due to aggressive strategies, tests cannot proceed faster than real-time, and evaluations cannot be easily parallelized. In response to these challenges, our solution provides a validated, deterministic, and parallel simulation environment as well as a method of adapting derived strategies to reality via an efficient online optimization component.

This work has three primary contributions. First, we provide a toolchain for superoptimization of autonomous racers called TUNERCAR (see Figure 1). This toolchain includes target hardware, modular software, and a calibrated simulator. Second, we describe a methodology for *tuning* a high-dimensional set of hyperparameters spanning control, planning, dynamics, and strategy; it is a general approach for adapting and optimizing heterogeneous components to new robots and domains. Finally, we validate our solution on an AV software stack and $1/10$th-scale open-source vehicle developed for this project [`f1tenth.org`].

In what follows, we demonstrate that TUNERCAR achieves improvement in lap time relative to other approaches tested given a fixed computation budget, and that our solution exceeds the performance of crowd-sourced expert solutions by up to 21%. Section II describes the problem setup and optimization pipeline. Section III places our solution in context with previous approaches. Section IV describes the modular autonomy stack, simulator, and hardware utilized for large-scale experiments detailed in Section V.

*Authors contributed equally to this work

[1]Authors are with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, USA

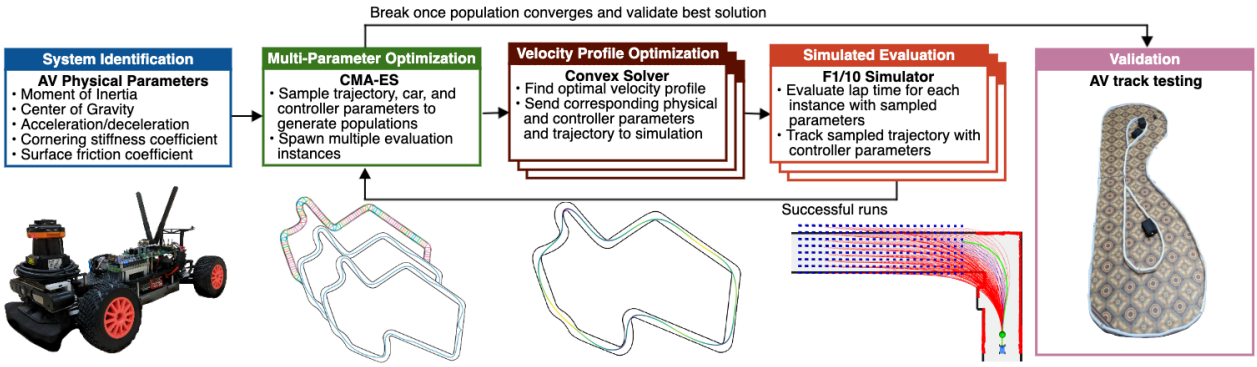

Fig. 1: The TUNERCAR toolchain which jointly optimizes racing strategy, planning methods, control algorithms, and vehicle parameters for an autonomous racecar.

II. METHODOLOGY

A. Problem statement

Our goal is to determine the best vehicle parameters, racing strategy, and controller settings that minimize lap time for a given racing track (also referred to as map). Mathematically, we define the objective function as the lap time, $f(\Theta) : \mathbb{R}^n \to \mathbb{R}$. Here lap time is computed by simulating the system producing a trajectory. The search space parametrized by Θ is the concatenation of three components – vehicle dynamics, racing strategy, and controller. Note that in this work we utilize a simulator (black-box) where the resulting scalar measure is deterministic given an assignment of parameters $\theta \in \Theta$ but may be noisy relative to *in situ* executions. We do not make any other assumptions such as the existence of a gradient. Our approach is shown in Figure 1. In what follows we define a parameterization of the autonomous racer, describe an evaluation criteria, and detail a method to sample and optimize realizations of the parameters.

B. Search space

The physical parameters Θ_p are defined as the mass, m, the location of the center of gravity in the longitudinal direction, l_g, the friction coefficient, μ_s, the height of the center of gravity, h_g, front cornering stiffness, C_{α_f}, and rear cornering stiffness, C_{α_r}. We limit the range of these parameters such that they are physically achievable without major modifications. The nominal values and their ranges are based on system identification performed on the actual vehicle, for further details see Appendix A in the technical report [10].

The strategy parameters Θ_s are defined by the nominal path (x, y) and the velocity profile (v_x, v_y); the size varies depending on the track, for example, $\Theta_s \in \mathbb{R}^{260}$ on *Philadelphia Track A*. The path and velocity profile combined represent the largest portion of the search. To improve the convergence of our proposed optimization method, we note that a deterministic assignment of the optimal velocity profile is possible given a path x, y, reducing the size of the search space by $1/3$. Specifically, the optimal velocity profile can be computed in polynomial time by solving a convex minimum-time parametric optimization problem. Due to space constraints, we refer the reader to [11] and Appendix B in the technical report [10] for full details of the minimum time path traversal formulation.

Finally, the driver parameters Θ_d represent aspects of the control algorithms used. Manipulating these parameters affects how the vehicle tracks the raceline. There are three components: the waypoint lookahead gain g_w, the planning horizon d_l, and the speed gain v_g (a precise definition is given in Section IV-C).

C. Evaluation criteria

For a given set of parameters, θ, i.e. waypoints with velocities, vehicle dynamics and controllers, we calculate the lap time, $f(\theta)$, using our simulator detailed in Section IV. When the simulator receives a new sample, the parameters are updated, the environment is reset, and the virtual vehicle attempts to traverse the track. The result, simulated lap time, is used as the objective function which sorts a set of samples (or population) into quantiles. Solutions are determined to be feasible if the trajectory does not intersect with the boundaries of the track (accounting for the car's length and width). Infeasible solutions are rejected and replaced at the sampling stage described in the next subsection.

D. Optimization

Our proposed optimization problem is high-dimensional, non-convex, non-smooth, and lacks a closed-form expression of the dynamics. Thus, we use gradient-free black-box optimization techniques. A potential pitfall of heuristic optimization methods applied to this class of problem is the inherent tension between exploration of the search space and exploitation of solutions due to the potential existence of local minima. We utilize a method known as covariance matrix adaptation evolution strategies (CMA-ES), [12]; our implementation is described in Algorithm 1. CMA-ES is known to perform well in challenging regimes with many local minima because it explicitly balances exploration with hill-climbing. CMA-ES adapts the covariance matrix (increases the L^2 norm) of the proposal distribution when the top performers of the population are far from the rest of the population, and conversely narrows the spread of the proposal distribution as the population becomes less diverse. Additionally, this approach also allows massively parallel

Algorithm 1 Covariance matrix adaptation evolution strategy (CMA-ES) [12]

input: population size λ, parent number μ
randomly initialize population: θ_k for $k = 1, ..., \lambda$
calculate population means $\hat{\theta}^g$ of the parameters
while termination criterion not met **do**
  evaluate current population θ^g using $f(\theta)$
  sort θ from smallest to largest objective, $f(\theta)$
  isolate top $K = \mu\lambda$ individuals: θ_k for $k = 1, .., K$
  estimate the covariance matrix $C^{(g+1)}$ using $\hat{\theta}^g$
  calculate the means $\hat{\theta}^{(g+1)}$ of θ_k for $k = 1, .., \mu$
  sample λ individuals via $\hat{\theta}^{(g+1)}$ and $C^{(g+1)}$
end while

evaluation of solutions, the most computationally expensive aspect of this pipeline.

As described in Algorithm 1, we first randomly initialize a population of feasible solutions using uniform sampling. Once the population has been evaluated, the top $\mu\lambda$ solutions are isolated. In Section V, we detail the results of a grid search to identify high-performing, generalizable settings for μ and the population size, λ. The top quantile of solutions is used to fit a multivariate Gaussian distribution from which the next generation of samples is drawn. We repeat this process, terminating when the L^2-norm of the covariance matrix, C^g, is less than $\epsilon = 0.01$ (see Section V and [12]).

III. RELATED WORK

This paper presents a general approach to heterogeneous component integration and optimization suitable for robotic systems that interact with a physical environment. A complete discussion of superoptimization and compiler literature has been omitted. However, specific approaches such as program sketching, population-based training, and program synthesis are of interest to the robotics community. We also consider a narrower view of this work relative to numerous methods for optimizing specific vehicle components and software within the racecar engineering community.

Sketching [6] utilizes fragments of programs which capture macroscopic details about the structure of the solution in order to synthesize the low-level details. In contrast, program synthesis [13] attempts to construct a correct program from scratch using only formal specifications. Neither approach inherently considers optimality, only correctness. Examples of sketching [14] and program synthesis [15] techniques applied to robotics generally fail to address the gap between models of robotic systems and realizable interactions between the program and the world. Even still, a variety of computational complexity and undecidability results [16] remain a barrier to applying these methods to realistic systems. Another closely related approach, population-based training (PBT) [17] has recently been utilized to search for efficient neural network designs; however, earlier work [18] focused only on physical robot design, a subset of the problem addressed in this paper. Likewise, [19] directly addresses the synthesis of a controller and planner interface but does not address the real-world applicability of the approach, specifically with respect to the (lack-of) realism in the simulator and vehicle dynamics. Like [19] the approach presented in Section II utilizes a black-box optimization method, CMA-ES [20]. We note that most literature in superoptimization employs similar search algorithms.

In general, race car optimization has historically been divided into two main categories: vehicle parameter optimization and racing trajectory optimization. The former uses a fixed track and fixed trajectory while varying the car parameters while the latter uses a fixed track and a fixed car while varying the trajectories. In [21], 500 pairs of roll stiffness and weight distribution values are evaluated using Pareto minimum analysis to optimize for the fastest lap time. More recently, [22], [23], [24], feature expanded design spaces, parallel computation, and utilize simple genetic algorithms.

In raceline optimization, several approaches have been proposed. An iterative two-step algorithm that separates the longitudinal and lateral control components is explored in [25]. Alternatively, in this paper, we use a population-based guided search using CMA-ES to determine the best raceline. In this process, we use the result from [11] that calculates both lateral and longitudinal forces required to minimize time to traverse a *fixed trajectory*. Concurrent work uses Bayesian optimization in place of CMA-ES [26]. An alternative approach [27], attempts to separate the autonomous racecar performance into two steps. First an approximate solution to the high-level path planning problem is computed then the lap time is minimized using a model predictive controller (MPC) in a receding horizon fashion. Similarly [28] formulates raceline optimization as an MPC problem using a learned dynamics model. Like these works we also perform system identification and incorporate model predictive control; however, we generalize the raceline optimization problem to explicitly consider the system as a whole.

IV. SIMULATOR AND HARDWARE

TUNERCAR includes a complete set of system components which are utilized to demonstrate the approach. In this section we outline the vehicle hardware, simulator, and vehicle software. The target hardware, an open-source 1/10th scale autonomous vehicle created by the authors, is mapped to a validated, deterministic simulator capable of modeling both the dynamics and sensors included on the vehicle [`f1tenth.org`]. The simulator includes a wrapper which enables a distributed approach to optimization of the source program with low communication overhead. In addition, the toolchain provides performance oriented implementations of the core algorithms (see Section IV-C) and a method to update their parameters.

A. Hardware

The vehicle, shown in Figure 2, is designed around a ready-to-run RC car chassis [29]. A power board used to manage the onboard compute and sensors as well as mounting plate design are provided. All aspects – hardware,

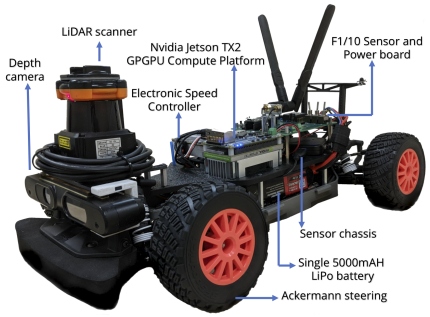

Fig. 2: Target 1/10th scale vehicle.

mechanical design, software – of these additional parts are open source. Computation occurs on the onboard NVIDIA TX2, a modern embedded system on a chip (SoC) which contains a conventional multicore ARM CPU in addition to a power-efficient GPU. The main sensor is a planar laser scanner (or LIDAR) which can capture range measurements. The LIDAR enables the vehicle to localize, estimate odometry, and create maps. Due to the operating environment (typically corridors with few features), we supplement the LIDAR measurements with additional odometry information from the electronic speed controller [30]. An optional RGB-D camera provides additional sensing modalities, but is not used in these experiments.

B. Simulator

The simulator uses a lightweight 2D physics engine written in C++ which implements the single track vehicle model described in [31]. System identification was performed to determine vehicle parameters – mass, center of mass moment of inertia, friction coefficient, cornering stiffness, and maximum acceleration/deceleration rates, Appendix A in [10] contains further details. The moment of inertia was estimated using the bifilar (two-wire) pendulum method [32]. Tire parameters were found using the PAC2002 Magic-formula model [33] and the cornering stiffness coefficient was back-calculated using [34]. A force scale was used to measure the kinetic friction coefficient between the rubber tires and linoleum floor as the vehicle was dragged laterally at a constant velocity. In addition to modeling vehicle dynamics, the simulator detects collisions between the vehicle and obstacles in the environment in linear-time using a pre-computed lookup table of range measurements [35]. This method is also used to simulate the vehicle's LIDAR.

The simulator differs significantly from existing ROS-based tools. To create deterministic rollouts, the C++ executable is wrapped in the OpenAI Gym [36] environment which explicitly steps time when all inputs have been computed. A further benefit of this approach is the ability to take advantage of faster than real-time execution. On commodity hardware, simulations proceed at approximately 6x real-time. Finally, the simulation environment provides a method for the members of the vehicle dynamics parameter class to be modified by TUNERCAR at the beginning of each rollout. Last, the simulation toolchain include a front end for generating random maps and pre-processing occupancy grid maps created using SLAM.

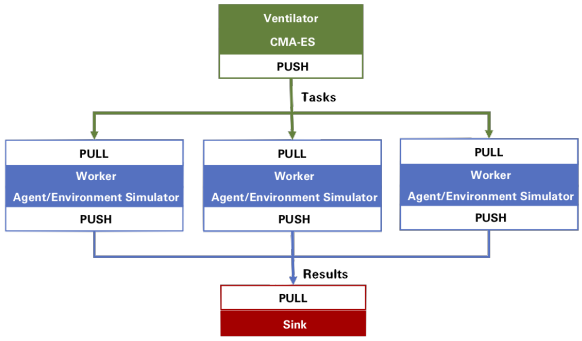

Fig. 3: MapReduce pattern of the search implementation.

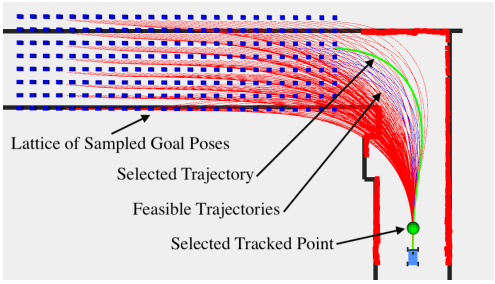

Fig. 4: Adjusting the horizon of the path planner shifts the lattice of samples (blue markers), the green marker represents the pure pursuit trackers selected lookahead distance.

The TUNERCAR toolchain employs a MapReduce [37] messaging pattern implemented using ZeroMQ [38], a high-performance asynchronous messaging library. Figure 3 shows many 'workers' spawned in parallel, each equipped with a simulator, velocity optimizer, and full vehicle stack. Samples from the CMA-ES search node are then broadcast to a pool of worker nodes. On receipt of a new sample, the worker instantiates vehicle and environment parameters and simulates a rollout to compute the lap time. In order to update the searcher's sampling distribution, the worker simply transmits the lap time and task index to a sink node which collects the worker results and synchronizes the search epochs.

C. Vehicle Software

Some algorithms are considered adaptable; others such as the simultaneous localization and mapping package (SLAM) are only used offline to create a model of the environment and, thus, are not tuned. The complete set of non-adaptable algorithms and software includes Google Cartographer [39] for SLAM, a particle filter for pure localization [40], and a behavior controller which manages communication between the planning nodes and the motor controller.

TUNERCAR compiles parameterized versions of two processes on the vehicle: a geometric path tracking controller based on pure pursuit [1], [41] and a path planning module which utilizes a sampling-based non-linear model predictive control [42], [43]. While the reference trajectory is computed by TUNERCAR it is differentiated from these modules as it is not updated online.

1) Path Tracker: The goal of path tracker is to compute steering inputs which allow the autonomous vehicle to follow a sequence of waypoints defined in the map frame. Performance of a path tracker is measured by the feasibility of the inputs, heading error, and lateral offset between the path and vehicle's position [41]. We utilize the pure pursuit path tracker [1], a geometric method which has been shown to be effective if properly tuned for the expected operating conditions. In particular, the lookahead distance, which is used to select a point on the path ahead of the vehicle significantly affects the performance. Too small of a lookahead and the vehicle oscillates; too large and corners are cut. Thus, the lookahead distance of the path tracker computed as $g_w d_l$ (where the tracking lookahead gain is $g_w < 1$ to ensure the tracking lookahead doesn't exceed the path planning horizon, d_l), is included in the search space as a tuneable parameter.

2) Path Planner: The goal of the path planner is to generate kinematically and dynamically feasible trajectories that can take the vehicle from its current pose to a sampled set of goal poses, see Figure 4. Each trajectory is represented as a set of cubic spirals, $p = [s, a, b, c, d]$ where s is the total arc length of the trajectory and (a, b, c, d) are equispaced knot points encoding the path curvature. For each trajectory, gradient descent is used to find spline parameters which minimize the error between the goal pose and the calculated end point given by a forward simulation of the vehicle dynamics. Real-time performance of the system is improved by creating a dense lookup table of pre-computed goal, solution pairs as described in [44]. Thus, online, once a goal point has been chosen, all that remains is to sample N equidistant points corresponding to the spline parameters stored in the lookup-table and check them against an occupancy grid for feasibility. In order to adapt the method to the operational domain the horizon of the planner is adjusted to account for track geometry.

V. CASE STUDY

In our experiments we explore the assignment of hyperparameters of the CMA-ES algorithm as well as the effectiveness of the method in both simulation and reality. As noted in Section II, CMA-ES requires only three hyperparameters: ϵ, the threshold of the L^2 norm of the covariance matrix for the sampling distribution, λ, the population size, and μ, the quantile of samples to retain between epochs. Figure 5 shows an experiment in which ϵ is determined. We repeat this experiment on multiple hyperparameters selections, observing that the lowest lap time does not decrease significantly after the L^2 norm reaches the determined threshold for all experiments. Figure 6 shows the performance of the CMA-ES population for a selection of quantiles μ and population sizes λ. We observe that the best final minimum lap time is achieved as λ increases and μ decreases. We omit results from some combinations of hyperparameters for the sake of clarity in Figures 5 and 6. Based on these experiments we select a μ of 0.01 and λ of 10000 for best overall performance in terms of lap time (here, convergence rate is less of a concern due to the offline nature of our approach).

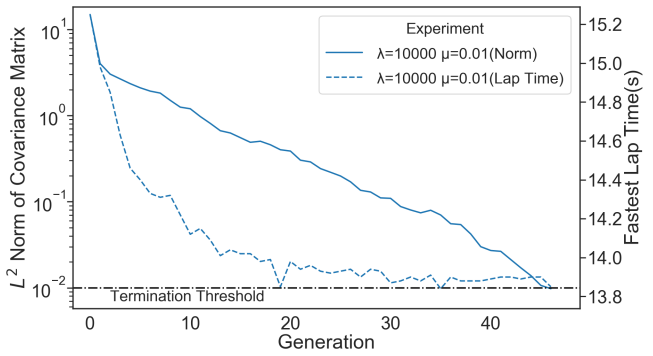

Fig. 5: Convergence of L^2 norm of covariance matrix to determine the threshold ϵ. The termination threshold for the norm is reached when the optimization converges to a local minimum.

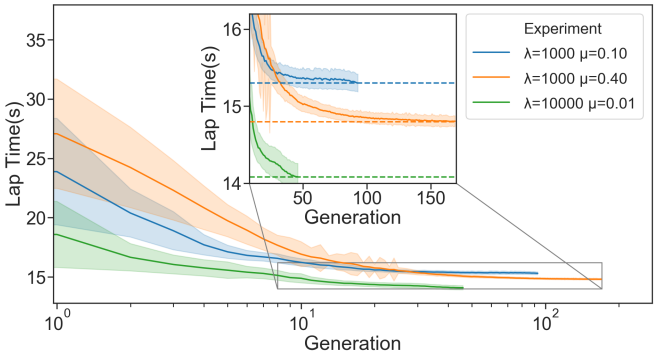

Fig. 6: Effect of population size λ and quantile μ on performance. The combination of λ at 10000, and μ at 0.01 has the best performance in terms of the balance between best lap time and convergence rate.

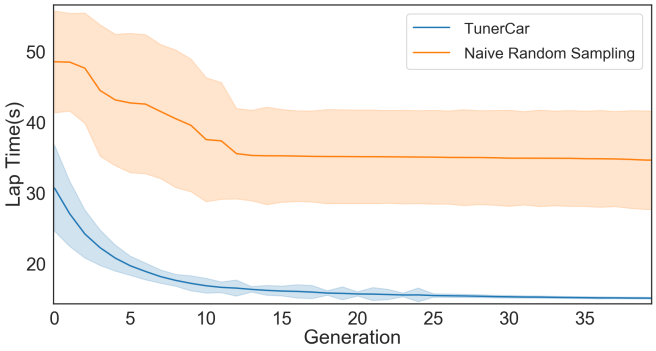

Fig. 7: Performance on Philadelphia Track A with fixed computation.

To further evaluate the CMA-ES implementation in TUNERCAR, we compare the mean and variance of the top 100 lap times relative to a naive random sampling for fixed computational budgets between 10,000 and 200,000 simulations. The results, shown in Figure 7 show that our methodology converges to a set of solutions with lap times significantly lower than that of naive random sampling.

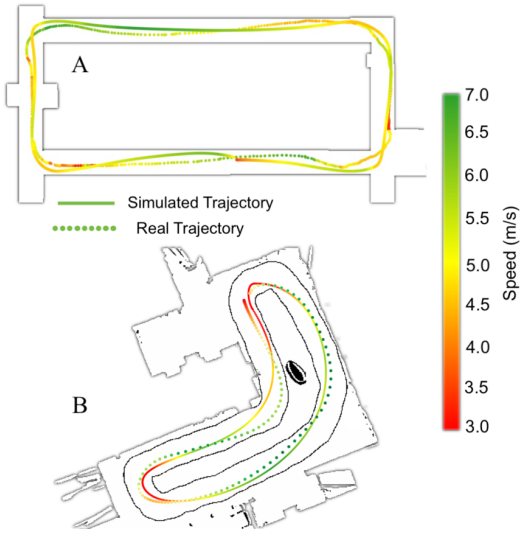

Fig. 8: Comparison of simulated versus real performance: (A) Philadelphia Track A, (B) Philadelphia Track B

TABLE I: TUNERCAR performance in simulation and reality

Philadelphia Track A		
Method	**Lap Time (s, sim)**	**Lap Time (s, real)**
TUNERCAR	16.2376 ± 0.2161	16.19
Pure Pursuit	17.2307 ± 0.0620	17.00
Expert Solutions	N/A	22.2291 ± 3.5958
Philadelphia Track B		
Method	**Lap Time (s, sim)**	**Lap Time (s, real)**
TUNERCAR	13.2424 ± 0.7782	14.03
Pure Pursuit	13.8256 ± 0.8848	16.80

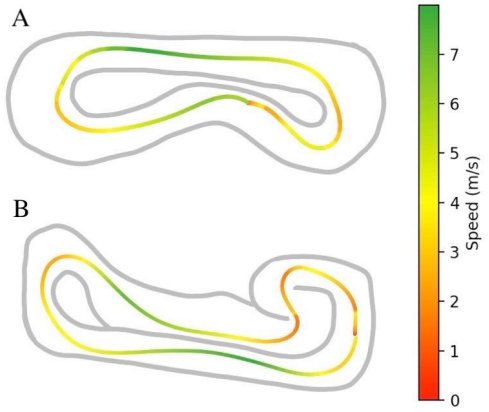

Fig. 9: Agent solutions on the tracks: (A) Porto, (B) Torino

TABLE II: TUNERCAR vs. expert-tuned agents in simulation

Method	**Lap Time (Porto, s)**	**Lap Time (Torino, s)**
TUNERCAR	8.11 ± 0.7372	17.96 ± 0.1214
Pure Pursuit	11.9 ± 0.8342	20.52 ± 0.1367
Best Expert Solution	9.37	19.19

We designed three sets of experiments to evaluate the effectiveness of our toolchain. First, we conducted experiments to validate the results of our approach using the hardware detailed in Section IV. These experiments are conducted on *Philadelphia Track A, B* which are challenging due to the 90 degree turns and narrow drivable surface areas (as seen in Fig. 8). The list of physical car parameters to tune was limited to an easily modifiable subset that avoided major structural changes to the car's components: total car mass and longitudinal location of the car's center of mass. After the pipeline has computed an optimal trajectory and set of parameters, we physically modified the car's parameters and deployed the tuned system, using the optimized waypoints and velocities.

Our second set of experiments was conducted in simulation and compared against historical racing times recorded in the environments. We chose to run the optimization process on a set of two tracks – *Porto* [45] and *Torino* [46] – where we have hosted competitive racing events in order to benchmark the performance of TUNERCAR relative to hand-tuned solutions. Due to the variety of modifications various entrants utilized (*e.g.* springs, suspension, geometry, and custom chassis), we do not restrict the set of physical parameters.

Lastly, we evaluate the reality gap in light of the proposed online optimization strategy by comparing vehicle performance in simulation to that which was achievable in reality. Figure 9 and Table II details the solutions found by TUNERCAR which exceed expert tuning-performance on (8.A) *Torino* and (8.B) *Porto*. Figure 8 and Table I show a comparison of the relative performance between top solutions deployed on the car and in the simulator on (9.A) *Philadelphia Track A* and (9.B) *Philadelphia Track B* as well as a comparison to expert engineered controllers (both crowd-sourced and created by the authors)

VI. CONCLUSIONS

We introduce TUNERCAR, a superoptimization toolchain for jointly optimizing racing strategy, vehicle parameters, planning methods, and control algorithms for an autonomous racecar. We detail the target hardware, software, simulators, and systems infrastructure necessary for implementation. Our methodology deploys a modern evolution strategy onto a massively parallelized software stack that enables simulations to proceed 6x times faster than real-world rollouts, achieving up to 21% faster lap times compared to expert solutions on real world race tracks. We validate our solution on an AV software stack and 1/10th-scale open-source vehicle developed for this project [`f1tenth.org`].

While the method as described in this paper has not been rigorously tested in head-to-head racing, the online path-planning component is capable of navigating the vehicle in the presence of other agents. Future work should investigate the feasibility of deploying this methodology utilizing an expanded search space that encodes the behaviors of other racers. A clear limitation of the presentation of this work is a lack of comparisons to other black-box optimizations, we mitigate this deficiency by measuring the toolchains performance against a variety of hand-tuned solutions entered in F1/10 competitions. Given that the experimental results show that we can find significant performance improvements over the other competition entries, it is important to explore whether other search strategies can create even faster agents.

REFERENCES

[1] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[2] A. V. Aho, R. Sethi, and J. D. Ullman, "Compilers, principles, techniques," *Addison Wesley*, vol. 7, no. 8, p. 9, 1986.

[3] C. A. Lattner, "Llvm: An infrastructure for multi-stage optimization," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2002.

[4] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic superoptimization," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 305–316.

[5] H. Massalin, "Superoptimizer: a look at the smallest program," in *ACM SIGARCH Computer Architecture News*, vol. 15, no. 5. IEEE Computer Society Press, 1987, pp. 122–126.

[6] A. Solar-Lezama, "Program sketching," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 5-6, pp. 475–495, 2013.

[7] P. Liang, M. I. Jordan, and D. Klein, "Learning programs: A hierarchical bayesian approach," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 639–646.

[8] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, "Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions," *arXiv preprint arXiv:1802.04730*, 2018.

[9] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," in *Acm Sigplan Notices*, vol. 48, no. 6. ACM, 2013, pp. 519–530.

[10] M. O'Kelly, A. Jain, H. Zheng, J. Auckley, K. Luong, and R. Mangharam, "Tech Report: TunerCar: A Superoptimization Toolchain for Autonomous Racing," University of Pennsylvania, Tech. Rep., September 2019.

[11] T. Lipp and S. Boyd, "Minimum-time speed optimisation over a fixed path," *International Journal of Control*, vol. 87, no. 6, pp. 1297–1311, 2014.

[12] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[13] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive (1) designs," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2006, pp. 364–380.

[14] T. Campos, J. P. Inala, A. Solar-Lezama, and H. Kress-Gazit, "Task-based design of ad-hoc modular manipulators," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6058–6064.

[15] J. Liu, N. Ozay, U. Topcu, and R. M. Murray, "Synthesis of reactive switching protocols from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 58, no. 7, pp. 1771–1785, 2013.

[16] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, 2000.

[17] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan *et al.*, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.

[18] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding," *ACM SIGEVOlution*, vol. 7, no. 1, pp. 11–23, 2014.

[19] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, pp. 2451–2463, https://worldmodels.github.io. [Online]. Available: https://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution

[20] N. Hansen, "The CMA evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[21] E. M. Kasprzak, K. E. Lewis, and D. L. Milliken, "Steady-state vehicle optimization using pareto-minimum analysis," *SAE transactions*, pp. 2624–2631, 1998.

[22] K. Hacker, K. Lewis, and E. M. Kasprzak, "Racecar optimization and tradeoff analysis in a parallel computing environment," SAE Technical Paper, Tech. Rep., 2000.

[23] F. Castellani and G. Franceschini, "Use of genetic algorithms as an innovative tool for race car design," SAE Technical Paper, Tech. Rep., 2003.

[24] K. Hayward, *Application of Evolutionary Algorithms to Engineering Design*. University of Western Australia, 2007.

[25] N. R. Kapania, J. Subosits, and J. C. Gerdes, "A sequential two-step algorithm for fast generation of vehicle racing trajectories," *Journal of Dynamic Systems, Measurement, and Control*, vol. 138, no. 9, p. 091005, 2016.

[26] A. Jain and M. Morari, "Computing the racing line using bayesian optimization," *arXiv preprint arXiv:2002.04794*, 2020.

[27] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[28] U. Rosolia and F. Borrelli, "Learning how to autonomously race a car: a predictive control approach," *IEEE Transactions on Control Systems Technology*, 2019.

[29] Traxxas, "Fiesta® ST Rally, 1/10 Scale Ford Fiesta AWD Rally Car."

[30] B. Vedder, "Vedder electronic speed controller." [Online]. Available: https://vesc-project.com/documentation

[31] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 719–726.

[32] H. A. Soule and M. P. Miller, "The experimental determination of the moments of inertia of airplanes," 1934.

[33] "Adams/tire help," MSC Software, 2 MacArthur Place, Santa Ana, CA 92707, April 2011.

[34] "Discussion: Cornering stiffness," McHenry Software Manual, M-smac Input, 2002.

[35] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance transforms of sampled functions," *Theory of computing*, vol. 8, no. 1, pp. 415–428, 2012.

[36] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[37] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[38] P. Hintjens, *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.

[39] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1271–1278.

[40] C. Walsh and S. Karaman, "Cddt: Fast approximate 2d ray casting for accelerated localization," vol. abs/1705.01167, 2017. [Online]. Available: http://arxiv.org/abs/1705.01167

[41] J. M. Snider *et al.*, "Automatic steering methods for autonomous automobile path tracking," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[42] T. M. Howard, *Adaptive model-predictive motion planning for navigation in complex environments*. Carnegie Mellon University, 2009.

[43] M. McNaughton, "Parallel algorithms for real-time motion planning," 2011.

[44] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[45] R. Mangharam, M. Behl, H. Abbas, and M. O'Kelly, "2018 Portuguese Grand Prix," in *F1/10 Workshop and Competition as part of the 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018.

[46] R. Mangharam, M. Behl, H. Abbas, M. Bertonga, and M. O'Kelly, "2018 Italian Grand Prix," in *F1/10 Workshop and Competition as part of ESWEEK 2018, Torino, Italy*, 2018.