

Residual Descent Differential Dynamic Game (RD3G) – A Fast Newton Solver for Constrained General Sum Games

Zhiyuan Zhang¹ and Panagiotis Tsiotras²

Abstract— We present Residual Descent Differential Dynamic Game (RD3G), a Newton-based solver for constrained multi-agent game-control problems. The proposed solver seeks a local Nash equilibrium for games where agents are coupled through their rewards and state constraints. By maintaining a dynamic set of active constraints, combined with a barrier function on satisfied constraints and a backtracking line search, the proposed method is able to satisfy state constraints while keeping the dimension of the Newton descent direction problem to a minimum. We compare the proposed method against state-of-the-art techniques and showcase the computational benefits of the RD3G algorithm on several example problems. The RD3G is up to 4X faster and has 2X higher convergence rate than existing approaches in higher dimensional games.

I. INTRODUCTION

Differential dynamic games (DDG) have a wide range of applications in robotics. Many complex problems in multi-agent motion planning in particular, such as drone racing [1], autonomous driving [2], satellite control [3], etc., can be modeled as DDGs, necessitating effective numerical methods to find their solutions.

Game-theoretic control seeks to find an equilibrium to a control problem modeled as a DDG. In game-theoretic problems, it is common to assume decoupled dynamics, that is, each agent's control affects only that agent's state. The agents interact with each other through mutual state constraints and through the game objective. Some commonly used solution concepts for game-theoretic control include Nash equilibrium [4], Stackleberg equilibrium [5], and Correlated equilibrium [6]. While the most suitable solution concept depends on the particular problem, Nash equilibrium remains the most studied and widely used equilibrium concept because it does not assume the order of action taken between the agents (e.g., Stackleberg equilibrium), nor does it require some form of external coordination (e.g., Correlated equilibrium). It is therefore more suitable for uncooperative interactions, which is common in many robotic control problems.

II. RELATED WORK

Various approaches have been proposed to find the Nash equilibrium of a DDG. A common approach is to discretize the action space and convert the differential dynamic game into a bimatrix game [7]. Since the action space can be

large, good heuristics are needed to choose a manageable set of candidate trajectories. A key advantage of this approach is that the trajectories may be computed in parallel on a Graphics Processing Unit (GPU), and then the bimatrix game can be solved using existing methods. This approach works well in practice but it relies heavily on heuristics. Furthermore, this method does not find the equilibrium of the original game, which has a continuous action space.

The Iterated Best Response (IBR) [8] iteratively solves the single-agent optimal control problem for each agent in the game while holding the policy of the other agents static. There is no guarantee of convergence, but if IBR converges for a particular problem, it does converge to the Nash equilibrium [8]. IBR, however, suffers in terms of convergence speed since agent interaction has to manifest through mutual constraints in multiple iterations.

Sensitivity-Enhanced IBR (SE-IBR) [9] is a recent improvement over IBR with a modified cost function to encourage the agent to move in the direction of the agent collision constraint, therefore discovering potential interactions, thus accelerating convergence. Still, SE-IBR involves solving multiple constrained optimal control problems at each time step when used in a receding horizon fashion, and depends on a high-performance solver to work in real-time.

While the equilibrium for a DDG with continuous action space and nonlinear dynamics is generally hard to find, a Linear Quadratic Game (LQ Game) admits an analytic solution [10]. Therefore, several methods approximate a nonlinear DDG with a multi-stage LQ Game and use Differential Dynamic Programming (DDP) to find an open- or closed-loop equilibrium control policy. Minimax DDP [11] treats each stage as a minimax game, while iLQGame [12] and OPTGame3 [13] extend Minimax DDP to general sum LQ Games. iLQGame, in particular, has been used successfully in motion control problems like autonomous car racing [14], [15].

The approaches discussed above are homotopy-type methods that approximate the original game using a series of easier-to-solve problems whose solutions converge to the equilibrium of the original game. However, it is also possible to directly solve for the necessary conditions for a Nash equilibrium [16]–[18]. Earlier methods such as [17], [18] use a single shooting approach and choose a descent step in the direction of dL/du , where L is the Lagrangian and u is the action variable. ALGame [16] is a multiple-shooting Newton method that finds the root of the KKT necessary conditions equations, and uses a dual ascent update step to manage the inequality constraints. For some robotic problems, ALGame is reportedly faster than iLQGame [16].

This work is sponsored by ONR awards N00014-23-2308 and N00014-23-2353 and NSF award IIS-2008686

¹ School of Aerospace Engineering, Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA, Email: zzhang615@gatech.edu

² School of Aerospace Engineering, Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA, Email: tsiotras@gatech.edu

In this work, we propose a novel Newton-based method to solve numerically game-theoretic control problems. In most Newton methods, solving for the descent direction is the most time-consuming part of the algorithm [19]. The proposed RD3G algorithm reduces the scale of the linear problem associated with finding the descent direction by partitioning the collision constraints and by removing the dual variables associated with the inactive constraints from the problem. In addition, it uses an interior point method along with a barrier function to maintain solution feasibility for the inactive constraints. This technique decreases the time spent in each iteration step by a large margin. In addition, we use a multiple shooting technique and perform gradient descent on the states and the controls simultaneously to help convergence.

Many existing methods for game-theoretic control are single-shooting methods [9], [12], [17], [18], which treat the state trajectory purely as derived variables of the control variables. This technique eliminates the need for dynamic constraints and results in simpler formulations. However, the derivatives of the state variables with respect to the control variable suffer from the diminishing gradient problem [20]. Moreover, an auto-gradient library is often needed to provide gradients due to the complex structure of the compute graph.

The proposed method uses indirect multiple shooting and takes gradient descent steps to solve the necessary conditions for a Nash equilibrium, which improves numerical stability and convergence, particularly for longer time horizons or stiff problems. It also mitigates issues related to the sensitivity of the solution to initial conditions, as it avoids the accumulation of numerical errors over the entire time horizon [21].

The remainder of the paper is organized as follows: Section III formulates the game, and Section IV describes the proposed solution method in detail. Section V shows several example problems, benchmark results, and physical experiments, followed by a conclusion and a short discussion on future works in Section VI.

III. PROBLEM FORMULATION

Let $x_k^i \in \mathbb{R}^n$ and $u_k^i \in \mathbb{R}^m$ denote the state and control vectors, respectively, for agent i at timestep k . To keep the notation simple, we assume that all agents have the same dimension for the state and the control. However, the approach can be easily extended to the case of heterogeneous agents. We use $[a..b]$ to denote all the integers between a and b , including a and b . We use parentheses to denote the concatenation of vector variables, i.e., $(a_1, a_2, \dots, a_n) = [a_1^T, a_2^T, \dots, a_n^T]^T$. When we omit the superscript of a variable, it holds for all agents. For example, $x_k = (x_k^1, x_k^2, \dots, x_k^N)$ is the concatenated states of all agents at time step k . The same notation applies for omitted subscripts, denoting every time step, that is, $x^i = (x_1^i, x_2^i, \dots, x_T^i)$ denotes the state trajectory for agent i at all time steps. The game state x is the concatenated state of all agents, i.e., $x = (x^1, x^2, \dots, x^N)$. Finally, u^{-i} denotes the concatenated control of all agents except agent i . Let N be the number of agents and T be the horizon length.

We are interested in finding the generalized Nash equilibrium to the following game:

$$\min_{u^i} J^i(x, u^i), \quad \forall i \in [1..N], \quad (1a)$$

$$x_{k+1}^i = f^i(x_k^i, u_k^i), \quad \forall k \in [0..T-1], \forall i \in [1..N], \quad (1b)$$

$$h(x_k^i, x_k^j) \leq 0, \quad \forall i \neq j, i, j \in [1..N], \forall k \in [1..T], \quad (1c)$$

where,

$$J^i(x, u^i) = \sum_{k=0}^{T-1} J_k^i(x_k, u_k^i) + \phi^i(x_T), \quad (1d)$$

denotes the total cost for agent i , which is the sum of step cost $J_k^i(x_k, u_k^i)$ and the terminal cost $\phi^i(x_T)$. In (1b) $f^i(x_k^i, u_k^i)$ denotes the dynamics for agent i , and in (1c) $h(x_k^i, x_k^j)$ is the agent-to-agent interaction constraint, which, without loss of generality, is assumed to be the same for all agents.

We seek the open-loop Nash equilibrium to the Generalized Nash Equilibrium Problem (GNEP) described in (1), that is, a set of control actions for all agents such that no agent can decrease its cost by unilaterally changing its control action. We denote the Nash equilibrium control as u^* .

IV. SOLUTION APPROACH

The proposed method contains four steps. First, partition the inequality state constraints in (1c). Second, find the descent direction of (1). Finally, determine the step size with a line search.

Our algorithm starts with an initial guess of the control and state trajectory. This initial guess does not have to satisfy the state constraints in (1c) but it should satisfy the dynamics constraints in (1b).

A. Constraint Partition

The number of inequality constraints $h(x_k^i, x_k^j)$ grows quadratically with the number of agents and linearly with the time horizon. Each constraint adds another dual variable, increasing the size of the decision variables, thus negatively impacting performance. To reduce the number of constraints, we separate the inequality constraints into two types: those currently not satisfied, which we need to rectify, and those already satisfied, which we maintain using a barrier function.

Let us denote the index set of active and currently violated constraints as $H_{i,k}^+ = \{j | h(x_k^i, x_k^j) \geq 0, i, j \in [1..N], j \neq i\}$, and the set of inactive constraints as $H_{i,k}^- = \{j | h(x_k^i, x_k^j) < 0, i, j \in [1..N], j \neq i\}$. We define the step Lagrangian for agent i as

$$L_k^i = J_k^i(x_k, u_k^i) + \lambda_k^{i,T} [f^i(x_k^i, u_k^i) - x_{k+1}^i] + \sum_{j \in H_{i,k}^+} \mu_k^{i,j} h(x_k^i, x_k^j) + \sum_{j \in H_{i,k}^-} B(h(x_k^i, x_k^j)), \quad (2)$$

where $B(h) = -\rho \log(-h)$ is the one-sided barrier function, and $\rho > 0$ is a homotopy parameter in (2), and $i \in [1..N], k \in [0..T-1]$. In (2) λ_k^i is the Lagrange multiplier for the i^{th} agent dynamics constraint. Define the variable

$\lambda \in \mathbb{R}^{TN}$ as the concatenation of all λ_k^i for all i and k , i.e., $\lambda = (\lambda_0, \lambda_1, \dots, \lambda_{T-1})$, where $\lambda_k = (\lambda_k^1, \lambda_k^2, \dots, \lambda_k^N)$. Similarly, $\mu_k^{i,j}$ is the Lagrange multiplier for the inequality state constraint, and let $\mu \in \mathbb{R}^{TN^2}$ be the concatenation of all $\mu_k^{i,j}$, defined by $\mu = (\mu_1, \mu_2, \dots, \mu_T)$, where $\mu_k = (\mu_k^{1,1}, \mu_k^{2,1}, \dots, \mu_k^{N,1}, \mu_k^{1,2}, \mu_k^{2,2}, \dots, \mu_k^{N,2}, \dots, \mu_k^{N,N})$.

The Lagrangian for agent i is

$$\mathcal{L}^i = \sum_{k=1}^{T-1} L_k^i + J_0^i(x_0^i, u_0^i) + \lambda_0^{i,T} [f^i(x_0^i, u_0^i) - x_1^i] + \phi^i(x_T^i) + \sum_{j \in H_{i,T}^+} \mu_k^{i,j} h(x_T^i, x_T^j) + \sum_{j \in H_{i,T}^-} B(h(x_T^i, x_T^j)), \quad (3)$$

where, for notational simplicity, we retain the term $J_0^i(x_0, u_0^i)$ to penalize the control at the first time step for notational simplicity, even though x_0 is not part of the decision variables.

B. Residual Descent

The necessary conditions for the generalized Nash equilibrium of the converted problem for agent i are [22]

$$\frac{\partial \mathcal{L}^i}{\partial x^i} = 0, \quad (4a)$$

$$\frac{\partial \mathcal{L}^i}{\partial u^i} = 0, \quad (4b)$$

$$f^i(x_k^i, u_k^i) - x_{k+1}^i = 0, \quad \forall k \in [0..T-1], \quad (4c)$$

$$\mu_k^{i,j} > 0, \quad h(x_k^i, x_k^j) = 0, \quad \forall j \in H_{i,k}^+, \forall k \in [1..T]. \quad (4d)$$

Next, let $F^i = (f^i(x_0^i, u_0^i) - x_1^i, f^i(x_1^i, u_1^i) - x_2^i, \dots, f^i(x_{T-1}^i, u_{T-1}^i) - x_T^i)$, and $h^{i,+} = (h_1, h_2, \dots, h_T)$, where $h_k = (h(x_k^i, x_k^{j_1}), h(x_k^i, x_k^{j_2}), \dots, h(x_k^i, x_k^{j_M}))$, where $j_1, \dots, j_M$ are the enumeration of elements in $H_{i,k}^+$, with M the number of elements in $H_{i,k}^+$. Define the residual of agent i as

$$r^i(x, u, \lambda, \mu) = \left(\frac{\partial \mathcal{L}^i}{\partial x^i}, \frac{\partial \mathcal{L}^i}{\partial u^i}, F^i, h^{i,+} \right), \quad (5)$$

and let the concatenated residual for all agents be $r = (r^1, r^2, \dots, r^N)$. Denote the concatenated decision variables as $y = (x, u, \lambda, \mu^+)$, where μ^+ stands for all entries of μ that are associated with an active or violated inequality state constraint. The objective of the Newton method is to find a solution to the set of equations $r^i(x, u, \lambda, \mu) = 0$ for all $i \in [1..N]$ by minimizing the norm $\|r(x, u, \lambda, \mu)\|^2$.

To this end, let y_ℓ be the current value for y , where ℓ denotes the current iteration step. We find the descent direction for the primal-dual pair y by solving for $\Delta y = (\Delta x, \Delta u, \Delta \lambda, \Delta \mu^+)$ from the first-order Taylor expansion for the combined residual, as in the following equation:

$$r(y_\ell) + \nabla r(y_\ell) \Delta y = 0, \quad (6)$$

where $\nabla r(y_\ell)$ is the Jacobian matrix of r evaluated at y_ℓ . Note that we only retain the entries in μ associated with the active or currently violated constraints to reduce the dimension of Δy . The solution to (6) is the change to y_ℓ that drives the residual $r(y_\ell + \Delta y)$ to zero, given the first order approximation, hence a descent direction.

C. Backtracking Line Search

We perform a backtracking line search in the descent direction to ensure $\|r(y_\ell + \Delta y)\|$ decreases with the chosen step size, and that the descent step does not increase the state constraint residual. It is important to check that a previously satisfied state constraint does not get violated when we take a descent step. We define the sum of all violated state constraint residuals as $\mathbf{h}(x) = \sum_{\{i,j,k|h(x_k^i, x_k^j) > 0\}} h(x_k^i, x_k^j)$. Algorithm 1 describes the line search in detail.

Algorithm 1 Backtracking line search

Define constants $\beta \in (0, 1), \alpha \in (0, 1)$, set variable $t = 1$
 Define $(x, u, \lambda, \mu) \leftarrow y_\ell, (\Delta x, \Delta u, \Delta \lambda, \Delta \mu) \leftarrow \Delta y$
 $y_t = [x + t\Delta x, u + t\Delta u, \lambda + t\Delta \lambda, \mu + t\Delta \mu]$
while $\|r(y_t)\| > (1 - \alpha t)\|r(y_\ell)\|$ or $\mathbf{h}(x + t\Delta x) > \mathbf{h}(x)$
do
 $t \leftarrow \beta t$
end while

D. Summary

We update the homotopy parameter ρ with the strategy proposed in [23], which is used in the popular solver IPOPT [24]. The update rule is:

$$\rho_{\ell+1} = \max \left\{ \frac{\epsilon_{\text{tol}}}{10}, \min \left\{ \kappa \rho_\ell, \rho_\ell^\beta \right\} \right\}, \quad (7)$$

where ϵ_{tol} is the desired tolerance, and κ, β are tuning parameters.

With this strategy, the homotopy parameter eventually decreases at a superlinear rate but does not get too small to cause numerical stability problems. The choice of ρ in (7) is proven to give rise to superlinear convergence under standard second-order sufficient conditions [23]. Algorithm 2 summarizes the proposed algorithm.

Algorithm 2 Residual Descent Differential Dynamic Game

Initialize $u, \lambda = 0, \mu = 0$, compute initial x from u
repeat
 Evaluate $h(x_k^i, x_k^j)$, build sets $H_{i,k}^+, H_{i,k}^-$ for all agents
 Calculate gradient of residual $\nabla r(y_\ell)$
 Solve for descent direction Δy from (6)
 Perform line search, find step size t using Algorithm 1
 Update $y_{\ell+1} \leftarrow y_\ell + t\Delta y$
 Update homotopy parameter ρ with (7)
until $\|r(y_\ell)\| < \epsilon_{\text{tol}}$
return y_ℓ

Partitioning the inequality constraints allows us to treat active and inactive constraints differently. The main advantage of this classification is that it allows us to identify the dual variables associated with the inactive constraints and therefore remove them from the residual. This reduces the dimension of the optimization problem for finding the descent direction, which is the most time-consuming operation.

Without these considerations, the dimension of the residual r in (5) is $N(Tn + Tm + Tn + TN) = O(TN^2)$. Let c be

the average number of agents an agent interacts with, and k be the average number of interaction steps. We can reduce the dimensionality of the residual to $N((Tn + kcn) + Tm + Tn + kn) = O(TNn)$, since $k \ll T, n \ll N$. This reduction in dimensionality has a significant influence on the solution time given that many linear solvers have a complexity of $O(d^3)$ where d is the dimension of the problem [25], [26].

V. EXPERIMENTS

A. Implementation

To evaluate the algorithm, the solver was implemented using Python and C++, with Eigen3 [27] for matrix manipulation in C++. The Least Squares Conjugate Gradient (LSCG) [28] was used as the main solver for the descent direction. In case LSCG fails to converge, the implementation falls back to SparseQR [29], which takes longer to converge but is more robust.

All experiments were performed on a desktop computer with an Intel i7-7700k 4.2GHz CPU and 64GB memory, running Ubuntu 22.04.

B. Numerical Examples

1) *Car Merging*: The traffic merging problem is a classic example of a differential game [30]. The game starts with several cars traveling in two adjacent lanes, each with a different longitudinal speed. The cars in the right lane attempt to merge into the left lane. Each car follows the kinematic bicycle model [31] with $[x, y, v, \theta]^T$ as the states, corresponding to the longitudinal position, lateral position, speed, and heading, respectively. The control for each car $u = [\delta_T, \delta_S]^T$ is the acceleration and the steering angle, respectively.

Each agent admits a linear quadratic cost that penalizes the deviation from the reference state and the control effort as follows

$$J_k^i(x_k, u_k^i) = (x_k^i - x_{\text{ref}}^i)^T Q_r (x_k^i - x_{\text{ref}}^i) + x_k^{iT} Q x_k^i + u_k^{iT} R u_k^i, \quad (8)$$

where Q_r penalizes the deviation from the target lateral position (lane) and deviation from the reference speed, and Q penalizes excessive heading values¹. The reference state x_{ref}^i specifies i^{th} 's car target longitudinal velocity, which is set randomly at initialization, and the target lateral position, which is the center of the left lane for all cars. The collision function $h(x^i, x^j)$ is set to the Cartesian distance between two cars. For this problem, the horizon is configured to 20 steps, and the initial reference control is zero. Figure 1 shows a merging maneuver with three cars.

Note that in Figure 1, the original gap between the yellow car and the purple car is not sufficient to fit the merging red car, so all cars have to adjust their speed for a successful merge, highlighting the “game” aspect of the problem.

Figure 2 shows the same example with more cars, and Table I shows the average solution time for different total number of cars. The initial states of the vehicles are randomly generated, and the algorithm terminates when the residual is

¹For the merging simulations, $T = 20$, $dt = 0.2$, $Q_r = [0, 0.1; 0.01, 0]$, $Q = [0, 0; 0, 1]$, $R = 0.3I_2$

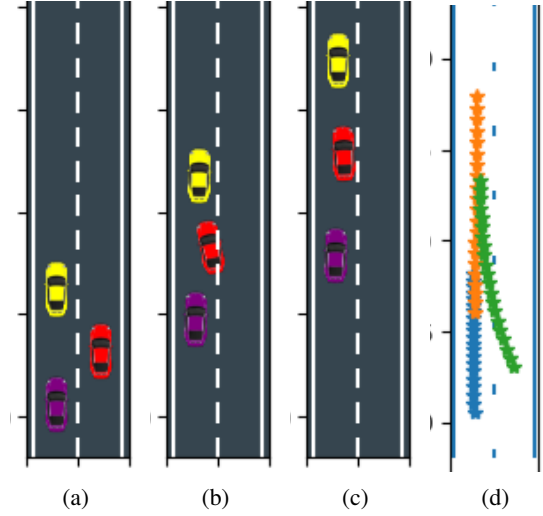


Fig. 1: Merging example with three cars, (a)-(c): Snapshots during merge, (d) Trajectory plot.

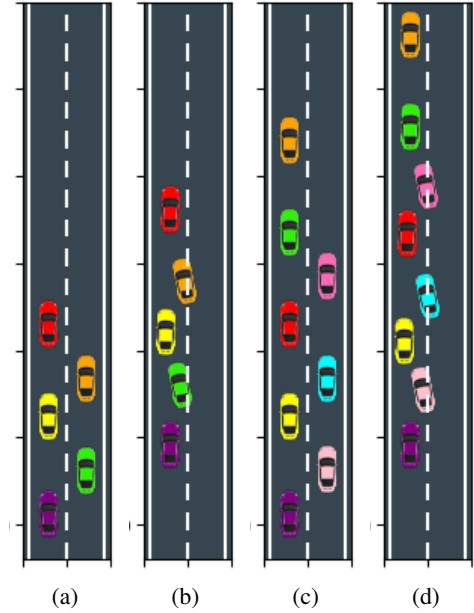


Fig. 2: Merging example with 5 and 8 cars.

smaller than 5×10^{-4} with no collision violation. The time shown is the average taken over 100 runs and only converged runs are included. For the three-car merging problem, the convergence rate is 91% for RD3G and 37% for iLQGame. Table I shows a clear performance advantage over the current state-of-the-art methods, especially when more agents are involved.

Game-theoretic controllers are often used in a receding horizon fashion and therefore must meet strict computation time budgets. To achieve this, designers of the control pipeline may be unable to afford to model all agents in the environment as responsive agents. As empirically shown in Table I, RD3G is up to 4X faster than the current state-of-the-art. This means that with RD3G, interactions with more

TABLE I: Average solution time (ms) of merging problems with different total car counts. N/C stands for “No Convergence”.

Car Count	RD3G	ALGame [16]	iLQGame [12]
2	18	48	324
3	29	95	505
4	83	198	657
5	137	366	829
6	226	653	1037
7	339	1103	2253
8	430	1833	N/C

agents can be modeled, leading to safer and more efficient planning outcomes.

2) *Car Racing*: The next simulation applies RD3G to an adversarial two-car racing game against a non-game-theoretic, interaction-agnostic MPC. Both agents share the kinematic bicycle model in Section V-B.1, with the following cost function

$$J_k^i(x_k, u_k^i) = Q_d(x_k^j - x_k^i) + (x_k^i - x_{\text{ref}}^i)^T Q_r(x_k^i - x_{\text{ref}}^i) + x_k^{i,T} Q x_k^i + u_k^i R u_k^i, \quad (9)$$

where x_k^i is the state of the ego agent, and x_k^j is the state of the opponent agent. The term $Q_d(x_k^j - x_k^i)$ is a cost item that rewards the lead distance of the ego agent against the opponent agent.

The agents are placed at random lateral and longitudinal starting positions on a Nascar-styled track about one car length away from each other. The agents are identical except for different initial speeds. Each race concludes when one car has completed a full lap. Table II shows the win rate of RD3G against a naïve MPC controller that shares an identical cost function, averaged over 200 simulated races with random initial states. “Start position” indicates whether the RD3G-controlled car is initially in the front or in the rear, and “Speed” indicates whether its initial speed is faster or slower than the non-ego agent. The proposed solver shows a clear advantage over non-game-theoretic MPC. In 72% of the cases, the agent using RD3G successfully blocks a faster opponent from passing.

TABLE II: Win rate of RD3G in a two-car racing game.

Start position \ Speed	Fast	Slow
Front	1.0	0.72
Rear	0.96	0.16

C. Physical Experiments

To test RD3G’s real-time performance on physical robotic systems, we implemented RD3G in a competitive racing scenario using the BuzzRacer platform [32].

BuzzRacer is a scaled autonomous vehicle platform for validating multi-agent control algorithms. Figure 3 shows the system components of BuzzRacer. The system contains

a $3\text{ m} \times 5\text{ m}$ reconfigurable race track and utilizes a visual tracking system for state measurements. BuzzRacer can support concurrent operations of multiple 1/28 scale cars. The miniature cars can reach up to 1g cornering acceleration and 3.5 m/s top speed.

Due to the time and effort involved with physical experiments and the difficulty in controlling initial conditions, we did not attempt to replicate the simulation results in Table II. We only seek to validate RD3G when used as a receding horizon controller, namely, that is sufficiently fast for the real-time control tasks of a physical system, and also that its solutions are indeed feasible. In the physical experiment, we configured one car to use a collision avoidance MPC with a reduced top speed, and another to use the proposed RD3G.

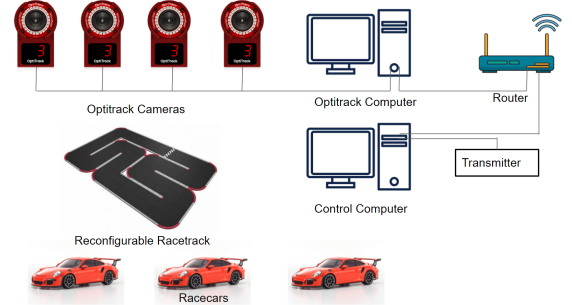


Fig. 3: BuzzRacer system components.

Figure 4 shows snapshots of an overtake during the experiment, where the vehicle controlled by RD3G overtakes the MPC-controlled car without collision. The average control loop frequency during the experiment is 29 Hz. Videos of the experiments and the simulations are available on our project website: <https://www.github.com/Nick-Zhang1996/RD3G>.

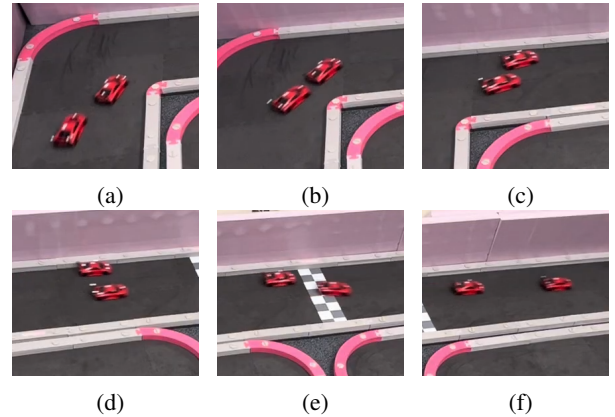


Fig. 4: Video screenshots of an overtake during the experiment.

The control loop frequency observed in the live experiments is consistent to the results in the benchmark simulations with the car merging problem, which is similar in scale compared to the racing problem. To this end, the benchmark results are representative of the solver’s performance in

practice. However, we see larger variations in the solution time during the physical experiment. In particular, the solver is slower in sections of the track that require large steering inputs. Given that we use a zero reference control as an initial guess, we speculate that when the equilibrium is far from the reference, it takes more iterations for RD3G to converge. While this can be helped with random initial guesses or reusing the solution from the last time step as an initial guess, more careful consideration is warranted in future works.

Another potential problem lies in the nature of a local NE solver; this is not specific to RD3G. During experiments, the solver consistently finds a local NE in the vicinity of the initial guess but is unable to switch to a categorically different NE having a potentially better social utility. For example, if the ego car starts at the rear right side of the non-ego car, it will remain on the right side even though it may be beneficial to switch to the left side. Human drivers frequently make decisive actions that qualitatively affect the interaction in a race, but this is currently unattainable with local NE solvers.

VI. CONCLUSION

We have presented a novel solver for differential dynamic games using iterative Newton descent on primal and dual residuals. By treating active and inactive inequality constraints differently and by ignoring the dual variables associated with the inactive constraints when solving for the descent direction, we were able to significantly reduce the size of the optimization problem. The proposed solver performed satisfactorily on different problems and compared favorably with current state-of-art methods.

Several directions can extend this work. First, existing root-finding libraries could be explored to improve the efficiency of the Newton descent step. Additionally, a sampling-based method could be developed to find global Nash equilibria, using RD3G as a foundational algorithm. Given the challenges of running the algorithm in real-time with an increasing number of agents, future work could investigate hierarchical solution approaches. Lastly, parallelizable algorithms leveraging GPU acceleration could be explored to enhance computational efficiency.

REFERENCES

- [1] R. Spica, E. Cristofalo, Z. Wang, E. Montijano, and M. Schwager, "A Real-Time Game Theoretic Planner for Autonomous Two-Player Drone Racing," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1389–1403, 2020.
- [2] M. Wang, Z. Wang, J. Talbot, J. C. Gerdes, and M. Schwager, "Game-Theoretic Planning for Self-Driving Cars in Multivehicle Competitive Scenarios," *IEEE Transactions on Robotics*, vol. 37, no. 4, pp. 1313–1325, Aug. 2021.
- [3] J. Zhou, J. Cheng, S. Wang, Y. Wang, and L. Zhao, "Input-delay Satellite Optimal Tracking Control Based on Differential Games," in *Chinese Control Conference*, Guangzhou, China, 9–12 Jul. 2019, pp. 1941–1945.
- [4] J. Nash, "Non-cooperative Games," *Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.
- [5] X. He, A. Prasad, and S. P. Sethi, "A Survey of Stackelberg Differential Game Models in Supply and Marketing Channels," *Journal of Systems Science and Systems Engineering*, vol. 16, no. 4, pp. 385–413, 2007.
- [6] J. Duffy and N. Feltoch, "Correlated Equilibria, Good and Bad: An Experimental Study," *International Economic Review*, vol. 51, no. 3, pp. 767–796, Aug. 2010.
- [7] A. Liniger and J. Lygeros, "A Noncooperative Game Approach to Autonomous Racing," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 3, pp. 884–897, 2020.
- [8] S. D. Bopardikar, A. Speranzon, and C. Langbort, "Convergence Analysis of Iterated Best Response for a Trusted Computation Game," *Automatica*, vol. 78, pp. 88–96, 2017.
- [9] M. Wang, Z. Wang, J. Talbot, J. C. Gerdes, and M. Schwager, "Game-Theoretic Planning for Self-Driving Cars in Multivehicle Competitive Scenarios," *IEEE Transactions on Robotics*, vol. 37, no. 4, pp. 1313–1325, Aug. 2021.
- [10] T. Başar and G. J. Olsder, *Dynamic Noncooperative Game Theory*, 2nd Edition. Society for Industrial and Applied Mathematics, 1998.
- [11] J. Morimoto, G. Zeglin, and C. Atkeson, "Minimax Differential Dynamic Programming: Application to a Biped Walking Robot," in *International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, Las Vegas, 1–5 Oct. 2003, pp. 1927–1932.
- [12] D. Fridovich-Keil, V. Rubies-Royo, and C. J. Tomlin, "An Iterative Quadratic Method for General-Sum Differential Games with Feedback Linearizable Dynamics," in *IEEE International Conf. on Robotics and Automation*, May 31–Aug. 31 2020, pp. 2216–2222.
- [13] D. Blueschke, R. Neck, and D. A. Behrens, "OPTGAME3: A Dynamic Game Solver and an Economic Example," in *Advances in Dynamic Games*, ser. Annals of the International Society of Dynamic Games, V. Křivan and G. Zaccour, Eds. Birkhäuser, 2013, vol. 13, pp. 23–46.
- [14] M. Rowold, "A Preview of Open-Loop and Feedback Nash Trajectories in Racing Scenarios," arXiv preprint arXiv:2310.00766, 2023. [Online]. Available: <https://arxiv.org/abs/2310.00766>
- [15] M. Rowold, A. Langmann, B. Lohmann, and J. Betz, "Open-Loop and Feedback Nash Trajectories for Competitive Racing with iLQGames," arXiv preprint arXiv:2402.01918, 2024. [Online]. Available: <https://arxiv.org/abs/2402.01918>
- [16] S. L. Cleach, M. Schwager, and Z. Manchester, "ALGAMES: A Fast Solver for Constrained Dynamic Games," in *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, July 12–16, 2020.
- [17] Y. Zhou, W. Jin, and X. Wang, "Distributed Differentiable Dynamic Game for Multi-robot Coordination," arXiv preprint arXiv:2207.08892, 2023. [Online]. Available: <https://arxiv.org/abs/2207.08892>
- [18] B. Di and A. Lamperski, "Newton's Method and Differential Dynamic Programming for Unconstrained Nonlinear Dynamic Games," in *Conference on Decision and Control (CDC)*, Nice, France, Dec. 11–13, 2019, pp. 4073–4078.
- [19] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
- [20] S. Basodi, C. Ji, H. Zhang, and Y. Pan, "Gradient Amplification: An Efficient Way to Train Deep Neural Networks," *Big Data Mining and Analytics*, vol. 3, no. 3, pp. 196–207, 2020.
- [21] H. Bock and K. Plitt, "A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems," *IFAC Proceedings Volumes*, vol. 17, no. 2, pp. 1603–1608, 1984, 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2–6 July 1984.
- [22] F. Facchinei, A. Fischer, and V. Piccialli, "Generalized Nash equilibrium problems and Newton methods," *Mathematical Programming*, vol. 117, pp. 163–194, Mar. 2009.
- [23] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A Trust Region Method Based on Interior Point Techniques for Nonlinear Programming," *Mathematical Programming*, vol. 89, pp. 149–185, 2000.
- [24] A. Wächter and L. T. Biegler, "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006.
- [25] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.
- [26] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997.
- [27] G. Guennebaud and B. Jacob, (2010) Eigen v3. Accessed: Aug. 2024. [Online]. Available: <http://eigen.tuxfamily.org>
- [28] M. R. Hestenes and E. Stiefel, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain," 2004, accessed: Aug. 2024. [Online]. Available: <https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf>
- [29] G. Guennebaud and B. Jacob, (2014) Eigen SparseQR. Accessed: Aug. 2024. [Online]. Available: https://eigen.tuxfamily.org/dox/classEigen_1_1SparseQR.html
- [30] X. Liao, X. Zhao, Z. Wang, K. Han, P. Tiwari, M. J. Barth, and G. Wu, "Game Theory-Based Ramp Merging for Mixed Traffic With Unity-SUMO Co-Simulation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 9, pp. 5746–5757, 2022.

- [31] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and Dynamic Vehicle Models for Autonomous Driving Control Design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015, pp. 1094–1099.
- [32] Z. Zhang and P. Tsiotras, "BuzzRacer: A Palm-Sized Autonomous Vehicle Platform for Testing Multi-Agent Adversarial Decision-Making," in *IEEE Int. Conf. on Intelligent Robots and Systems*, Abu Dhabi, UAE, 14-18 Oct. 2024.