2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

An Iterative Quadratic Method for General-Sum Differential Games with Feedback Linearizable Dynamics

David Fridovich-Keil*, Vicenç Rubies-Royo*, and Claire J. Tomlin

***Abstract*— Iterative linear-quadratic (ILQ) methods are widely used in the nonlinear optimal control community. Recent work has applied similar methodology in the setting of multi-player general-sum differential games. Here, ILQ methods are capable of finding local equilibria in interactive motion planning problems in real-time. As in most iterative procedures, however, this approach can be sensitive to initial conditions and hyperparameter choices, which can result in poor computational performance or even unsafe trajectories. In this paper, we focus our attention on a broad class of dynamical systems which are feedback linearizable, and exploit this structure to improve both algorithmic reliability and runtime. We showcase our new algorithm in three distinct traffic scenarios, and observe that in practice our method converges significantly more often and more quickly than was possible without exploiting the feedback linearizable structure.**

I. Introduction

In robotics, a wide variety of decision making problems, including low-level control, motion planning, and task planning, are often best expressed as optimal control problems. Specific algorithms and solution strategies may differ depending upon factors such as system dynamics and cost structure; yet, modern methods such as model predictive control have proven extremely effective in many applications of interest. Still, optimal control formulations are fundamentally limited to solving decision problems for *a single agent*.

Dynamic game theory—the study of games played over time—provides a natural extension of optimal control to the multi-agent setting. For example, nearby vehicles at an intersection (e.g., Fig. 1) mutually influence one another as they attempt to balance making forward progress in a desired direction while avoiding collision. Abstractly, dynamic games provide each agent, or "player," a separate input to the system, and allow each player to have a different cost function which they wish to optimize. Players may wish to cooperate with one another in some situations and not cooperate in others, leading to complicated, coupled optimal play. Moreover, different players may know different pieces of information at any point in time. Optimal play depends strongly upon this information structure; a player with an informational advantage can often exploit that knowledge to the detriment of any competitors. For example, in poker a player who cheats and looks at the top card in the deck is more certain of who will win the next hand, and hence can assume less risk in betting.

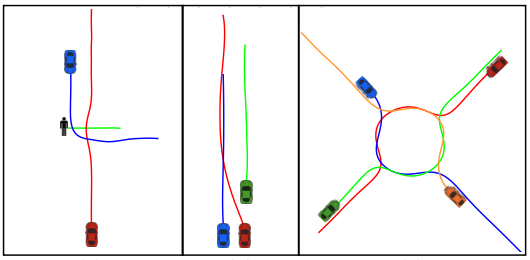

Fig. 1: Three traffic scenarios—(a) intersection, (b) high speed overtaking, and (c) roundabout merging—which we formulate as differential games and use to benchmark the performance of our method and a baseline [1].

In this paper, we consider dynamic games played in continuous time, or differential games. Historically, differential games were first studied in zero-sum (perfectly competitive) settings such as pursuit-evasion problems [2]. Here, optimal play is described by the Hamilton-Jacobi-Isaacs (HJI) PDE, in which the Hamiltonian includes a minimax optimization problem that encodes the instantaneous preferences of both players. These results extend to general-sum games as well, in which optimal play follows coupled HJ equations [3, 4]. Unfortunately, numerical solutions to these coupled PDEs often prove intractable because they operate on a densely discretized state space. Approximate dynamic programming methods [5] offer a promising alternative; still, computational efficiency remains a significant challenge.

Recently, the robotics community has shown renewed interest in dynamic and differential games [6–8], with a variety of new approximate algorithms for identifying locally optimal play. For example, Sadigh et al. [9] optimize the behavior of a self-driving car while accounting for the reaction of a human driver. Wang et al. [10] demonstrate a real-time iterative best response algorithm for planning competitive trajectories in a 6-player drone racing game. Building upon the earlier sequential linear-quadratic method of [11] and the well-known iterative linear-quadratic regulator [12, 13], our own prior work [1] solves warm-started 3-player differential games in under 50 ms each, operating single-threaded on a consumer laptop.

In this paper, we extend and improve upon our previous work [1] by exploiting the structure in a broad class of dynamical systems. Many systems, including quadcopters and the planar unicycle and bicycle models commonly used to model automobiles, are feedback linearizable. That is,

Department of EECS, UC Berkeley, {dfk, vrubies, tomlin}@eecs.berkeley.edu. * indicates equal contribution.

This research is supported by an NSF CAREER award, the Air Force Office of Scientific Research (AFOSR), NSF's CPS FORCES and VeHICaL projects, the UC-Philippine-California Advanced Research Institute, the ONR BRC grant for Multibody Systems Analysis, a DARPA Assured Autonomy grant, and the SRC CONIX Center. D. Fridovich-Keil is also supported by an NSF Graduate Research Fellowship.

there exists a (nonlinear) control law which renders the closed-loop input-output dynamics of these nonlinear systems *linear*. Here, we develop an algorithm for identifying locally optimal play in differential games with feedback linearizable dynamics. We establish theoretical equivalence between the solutions identified using this algorithm and those which do not exploit the feedback linearizable structure. By exploiting the structure, however, our algorithm is able to take much larger steps at each iteration and generally converge to an equilibrium more quickly and more reliably than was previously possible. Experimental results in Section VI confirm these computational advantages for the interactive traffic scenarios shown in Fig. 1.

II. Related Work

To put our work in context, here we provide a brief summary of iterative linear-quadratic (ILQ) methods of solving differential games, other approximate techniques of solving games, and common ways in which feedback linearization is used to accelerate motion planning.

A. ILQ methods and other approximate techniques

Iterative linear-quadratic (ILQ) methods are increasingly popular in the nonlinear model predictive control (MPC) and motion planning communities [12, 14]. These algorithms refine an initial control law at each iteration by forming a Jacobian linearization of system dynamics and a quadratic approximation of cost, and solving the resulting LQR subproblem. Because LQ games also offer an efficient solution, this approach has also been applied in the context of two-player zero-sum differential games by [11] and recently extended in [1] to the N-player general-sum setting. ILQ methods are *local*. For optimal control problems, this means that they generally converge to local optima; for differential games, if they converge, they converge to local equilia. Importantly, these methods scale favorably with state dimension (cubic), number of players (cubic), and time horizon (linear), and [1] reports real-time operation for several three-player examples.

Iterative best response (IBR) algorithms comprise another class of methods for solving games. Here, in each iteration players sequentially solve (or approximately solve) the optimal control problem which results when all other players' strategies are fixed. IBR has been demonstrated in a wide variety of settings, including congestion games [15], drone racing [10], and autonomous driving [16]. As in the case of ILQ methods, convergence is not generally guaranteed for arbitrary initializations. At best, IBR converges to local Nash equilibria (e.g., [10]). However, by reducing the game to a sequence of optimal control problems, IBR algorithms can take advantage of existing MPC and planning tools.

B. Feedback linearization in motion planning

Feedback linearization is a popular differential geometric control technique which renders a class of nonlinear systems' input-output response *linear*. We provide a brief technical overview of feedback linearization in Section IV; here, we summarize its relevance to motion planning. One of the early successes of feedback linearization was its effectiveness in planning for chained systems, e.g., a car with multiple trailers [17, 18]. Feedback linearization (and the related notion of differential flatness) is also commonly used for minimum snap control of quadrotors [19, 20]. Here, the differentially flat structure of the underlying system dynamics allows planners to generate piecewise polynomial trajectories which the system can track exactly. This concept is extended to the case of differential games in [10], where each iteration of IBR yields a new spline trajectory.

III. Problem Formulation

We consider N-player, general-sum differential games with control-affine dynamics. That is, we presume that the game state $x \in \mathbb{R}^n$ evolves as

$$\dot{x} = f(x) + \sum_{i=1}^{N} g_i(x)u_i \ , \tag{1}$$

where $u_i \in \mathbb{R}^{k_i}$ is the control input of player i. In our examples (Section VI), x will be the concatenated states of multiple subsystems, but this is not strictly necessary. We assume that (1) is full-state feedback linearizable, i.e. there exist outputs $y = h(x)$ such that y and finitely many of its time derivatives evolve linearly as a function of some auxiliary inputs $z_i \in \mathbb{R}^{k_i}$, for some control law $u_i := u_i(x, z_i)$. A brief review of feedback linearization may be found in Section IV.

Next, we suppose that each player i wishes to minimize a running cost ℓ_i over finite time horizon T:

$$J_i(u_1, \ldots, u_N) := \int_0^T \ell_i(t, x, u_1, \ldots, u_N)dt \ . \tag{2}$$

We shall require ℓ_i to be C^2 in $x, u_j, \forall j$, uniformly in time t. Player i's total cost J_i then depends explicitly upon each player's control input signal $u_i(\cdot)$ and implicitly upon the initial condition $x(0)$.

Finally, we presume that each player i has access to the state x at every time t, but *not* other players' control inputs $u_j, j \neq i$, i.e.

$$u_i(t) \equiv \gamma_i\big(t, x(t)\big) \tag{3}$$

for some measurable function $\gamma_i : [0, T] \times \mathbb{R}^n \to \mathbb{R}^{k_i}$. We shall denote the set of such functions Γ_i. For clarity, we shall also overload the notation of costs $J_i(\gamma_1; \ldots; \gamma_N) \equiv J_i\big(\gamma_1(\cdot, x(\cdot)), \ldots, \gamma_N(\cdot, x(\cdot))\big)$.

Thus equipped with dynamics (1), costs (2), and information pattern (3), in principle, we seek Nash equilibria of the game.

Definition 1: (Nash equilibrium, [21, Chapter 6]) A set of strategies $(\gamma_1^*, \ldots, \gamma_N^*)$ constitute a Nash equilibrium if no player has a unilateral incentive to deviate from his or her strategy. Precisely, the following inequality must hold for each player i:

$$\begin{aligned} J_i^* &\equiv J_i(\gamma_1^*, \ldots, \gamma_{i-1}^*, \gamma_i^*, \gamma_{i+1}^*, \ldots, \gamma_N^*) \\ &\leq J_i(\gamma_1^*, \ldots, \gamma_{i-1}^*, \gamma_i, \gamma_{i+1}^*, \ldots, \gamma_N^*), \forall \gamma_i \in \Gamma_i \ . \end{aligned}$$

In practice, we may only be able to check if these conditions are satisfied locally in the neighborhood of strategy γ_i^*, i.e., find *local* Nash equilibrium. As noted in [1] ILQ methods for games are known *not* to find even local Nash equilibria, but still be competitive and, importantly, efficient to compute.

IV. Background: Feedback Linearization

This section provides a brief review of feedback linearization, a geometric control technique popularly used across a wide range of robotic applications including manipulation, quadrotor flight, and autonomous driving.

Recall dynamics (1), and define the matrix $g(x) := [g_1(x), \ldots, g_N(x)] \in \mathbb{R}^{n \times k}$ and vector $u^T = [u_1^T, \ldots, u_N^T] \in \mathbb{R}^k$, with $k = \sum_i k_i$ the total control dimension. Thus, (1) may be rewritten as

$$\dot{x} = f(x) + g(x)u, \; y = h(x) \tag{4}$$

where y is the output of the system, and the functions f, g, and h are sufficiently smooth.

A. Mechanics

Suppose that (4) has well-defined vector relative degree $(r_1, \ldots, r_k)$ [22, Definition 9.15] and is full-state feedback linearizable. Then, there exists a matrix $M(x)$ and vector $m(x)$ such that the time derivatives of the outputs y follow

$$[y_1^{(r_1)}, \ldots, y_k^{(r_k)}]^T = M(x)u + m(x). \tag{5}$$

Presuming the invertibility of the so-called "decoupling matrix" $M(x)$, we may design the following feedback linearizing control law as a function of both state x and an *auxiliary input* $z \in \mathbb{R}^k$:

$$u(x, z) = M^{-1}(x)(z - m(x)), \tag{6}$$

which renders the input-output dynamics linear in the new auxiliary inputs z:

$$[y_1^{(r_1)}, \ldots, y_k^{(r_k)}]^T = z. \tag{7}$$

Note that, as for u in (1) we shall consider $z^T = [z_1^T, \ldots, z_N^T]$ to be a concatenation of auxiliary inputs for each player, with $z_i \in \mathbb{R}^{k_i}$.

B. Change of coordinates

We have seen how a careful choice of feedback linearizing controller $u(x, z)$ renders the dynamics of the output y and its derivatives linear. Define the state of this linear system as $\xi := [y_1, \ldots, y_1^{(r_1-1)}, \ldots, y_k, \ldots, y_k^{(r_k-1)}]^T$. Just as there is a bijective map (6) between control u and auxiliary input z whenever $M(x)$ is invertible, there is also a bijection between state x and linear system state $\xi, x = \lambda(\xi)$ [22] because (4) is full-state feedback linearizable. We shall use both bijective maps (and their derivatives) in Section V to rewrite costs (2) in terms of the linearized dynamics (7).

V. Methods

In this section we present our main contribution, a computationally stable and efficient algorithm for identifying local equilibria of general-sum games with feedback linearizable dynamics. We begin in Section V-A by computing a feedback linearizing controller for unicycle dynamics, which we shall use as a running example throughout the paper. Then, in Section V-B we show how to transform the costs for each player to depend upon linear system state ξ and auxiliary inputs z_i rather than state x and controls u_i. In Section V-C we introduce the main algorithm, and finally in Section V-D we summarize the effects of using feedback linearization.

A. Feedback linearization by example

Consider the following (single player) 4D unicycle dynamical model:

$$\dot{x} = \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ w \\ a \end{bmatrix}, \; y = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{8}$$

representing the evolution of the positions p_x and p_y, the orientation θ, and speed v. The inputs w and a represent the angular rate and the acceleration. By taking time derivatives of the output y following the procedure from Section IV, we obtain the new set of states $\xi = [p_x, \dot{p}_x, p_y, \dot{p}_y]^T$ for the linearized system. Differentiation reveals that

$$\begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \end{bmatrix} = \begin{bmatrix} -v\sin\theta & \cos\theta \\ v\cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} w \\ a \end{bmatrix}. \tag{9}$$

From this result, we compute the inverse decoupling matrix and drift term as

$$M^{-1}(x) = \begin{bmatrix} -\sin\theta/v & \cos\theta/v \\ \cos\theta & \sin\theta \end{bmatrix}, \; m(x) = 0. \tag{10}$$

Finally, we can also explicitly derive the state conversion map $\lambda(\xi)$

$$\lambda(\xi) = \begin{bmatrix} p_x \\ p_y \\ \sqrt{\dot{p}_x^2 + \dot{p}_y^2} \\ \tan^{-1}(\frac{\dot{p}_y}{\dot{p}_x}) \end{bmatrix}. \tag{11}$$

Now, consider a differential game with two players, each of whom independently follows dynamics (8). The inverse decoupling matrix $M^{-1}(x)$ and the Jacobian of the state conversion map λ for the full system will be block diagonal.

B. Transforming costs

So far, we have introduced feedback linearization and shown how to derive the mappings from auxiliary input z to control u and linearized system state ξ to state x. To exploit the feedback linearizable structure of (4) when solving the game, we must rewrite running costs $\ell_i(t, x, u_1, \ldots, u_N)$ in terms of ξ and z. Overloading notation, we shall denote the transformed running costs as

$$\begin{aligned} &\ell_i(t; \xi; z_1; \ldots; z_N) \equiv \\ &\quad \ell_i\Big(t; \lambda(\xi); u_1\big(\lambda(\xi), z_1\big); \ldots; u_N\big(\lambda(\xi), z_N\big)\Big), \end{aligned} \tag{12}$$

Algorithm 1: Feedback Linearized Iterative LQ Games

Input: initial linearized system state $\xi(0)$ and control strategies $\{\gamma_i^0\}_{i\in\{1,\dots,N\}}$, time horizon T

Output: converged control strategies $\{\gamma_i^*\}_{i\in\{1,\dots,N\}}$ for the linearized system

1 **for** *iteration* $p = 1, 2, \dots$ **do**
2 $\quad \mu^p \equiv \{\hat{\xi}(t), \hat{z}_i(t)\}_{i\in\{1,\dots,N\},t\in[0,T]} \leftarrow$
3 $\quad\quad$ getTrajectory$\big(\xi(0), \{\gamma_i^{p-1}\}\big)$;
4 $\quad \{l_i(t), Q_i(t), R_{ij}(t)\} \leftarrow$ quadraticizeCost$\big(\mu^p\big)$;
5 $\quad \{\tilde{\gamma}_i^p\} \leftarrow$ solveLQGame$\big(\{l_i(t), Q_i(t), R_{ij}(t)\}\big)$;
6 $\quad \{\gamma_i^p\} \leftarrow$ stepToward$\big(\{\gamma_i^{p-1}, \tilde{\gamma}_i^p\}\big)$;
7 $\quad$ **if** *converged* **then**
8 $\quad\quad$ **return** $\{\gamma_i^p\}$

where $u_i(\lambda(\xi), z_i)$ is given in (6).

Section V-C presents our main algorithm; a core step will be to compute first and second derivatives of each player's running cost with respect to the new state ξ and inputs z_i. This may be done efficiently using the chain rule and exploiting known sparsity patterns for particular systems and costs. For completeness, however, we shall ignore sparsity and illustrate computing the first derivative of ℓ_i with respect to the j^{th} dimension of ξ, denoted ξ_j:

$$\frac{\partial \ell_i}{\partial \xi_j} = \sum_{p=1}^{n} \frac{\partial \ell_i}{\partial x_p}\frac{\partial x_p}{\partial \xi_j} + \sum_{n=1}^{N}\sum_{p=1}^{k_n} \frac{\partial \ell_i}{\partial u_{n,p}} \sum_{q=1}^{n} \frac{\partial u_{n,p}}{\partial x_q}\frac{\partial x_q}{\partial \xi_j} \quad (13)$$

where $u_{n,p}$ is the p^{th} entry of the n^{th} player's control input.

Second derivatives may be computed similarly, though again we stress that for specific dynamics and cost functions it is often much more efficient to exploit the *a priori* known sparsity of partial derivatives. Interestingly, we also observe that the terms arising from the second sum in (13), which account for the state-dependence of the feedback linearizing controllers (6), are often negligible in practice and may be dropped without significant impact on solution quality.

C. Core algorithm

Like the original iterative LQ game algorithm, we proceed from a set of initial strategies γ_i for each player—understood now to map from (t, ξ) to z_i—and iteratively refine them by solving LQ approximations. Our main contribution, therefore, lies in the transformation of the game itself into the coordinates ξ, z_i which correspond to feedback linearized dynamics. As we shall see in Section VI, iterative LQ approximations are much more stable in the transformed coordinates and converge at least as quickly.

Algorithm 1 outlines the major steps in the resulting algorithm. We begin at the given initial condition $\xi(0)$ for the linearized system and strategies γ_i^0 for each player. Note that these strategies define control laws *for the linearized system*, i.e. $z_i(t) \equiv \gamma_i\big(t, \xi(t)\big)$.

At each iteration, we first (Algorithm 1, line 3) integrate the linearized dynamics (7) forward to obtain the current operating point $(\hat{\xi}(\cdot), \{\hat{z}_i(\cdot)\})$. Then (Algorithm 1, line 4), we compute a quadratic approximation to each player's running cost in terms of the variations $\delta\xi := \xi - \hat{\xi}$ and $\delta z_j := z_j - \hat{z}_j$

$$\ell_i(t;\xi;z_1;\dots;z_N) - \ell_i(t;\hat{\xi};\hat{z}_1;\dots;\hat{z}_N) \approx \delta\xi^T l_i(t) + \frac{1}{2}\delta\xi^T Q_i(t)\delta\xi + \frac{1}{2}\sum_{j=1}^{N} \delta z_j^T (R_{ij}(t)\delta z_j + 2r_{ij}(t)), \quad (14)$$

using the chain rule as in (13) to compute the terms l_i, Q_i and R_{ij} for each player.

Equipped with linear dynamics (7) and quadratic costs (14), the solution of the resulting general-sum LQ game is given by a set of coupled Riccati differential equations, which may be derived from the first order necessary conditions of optimality for each player [21, Chapter 6]. In practice (Algorithm 1, line 5), we numerically solve these equations in discrete-time using a time step of Δt. If a solution exists at the p^{th} iteration, it is known to take the form

$$\tilde{\gamma}_i^p(t,\xi) \equiv \hat{z}_i(t) - P_i^p(t)\big(\xi(t) - \hat{\xi}(t)\big) - \alpha_i^p(t) \quad (15)$$

for matrix $P_i^p(t)$ and vector $\alpha_i^p(t)$ [21, Corollary 6.1].

We cannot simply use these strategies at the $(p+1)^{\text{th}}$ iteration or we risk diverging, however, without further assumptions on the curvature and convexity of running costs ℓ_i. In fact, these costs are generally nonconvex when expressed in terms of ξ and z_j (12), which necessitates some care in updating strategies. To address this issue (Algorithm 1, line 6), we follow a common practice in the ILQR and sequential quadratic programming literature (e.g., [23]) and introduce a step size parameter $\eta \in (0, 1]$:

$$\gamma_i^p(t,\xi) = \hat{z}_i(t) - P_i^p(t)\big(\xi(t) - \hat{\xi}(t)\big) - \eta\alpha_i^p(t). \quad (16)$$

Observe that, taking $\eta = 0$ and recalling that $\xi(0) = \hat{\xi}(0)$, we recover the previous open-loop control signal $\gamma_i^p(t,\xi) = \hat{z}_i, \forall t \in [0, T]$. Taking $\eta = 1$, we recover the LQ solution from this iteration (15). As is common in the literature, we perform a backtracking linesearch on η, starting with initial value η_0 and terminating when the trajectory that results from (16) satisfies a trust region constraint at level ϵ. In our experiments, we use an L_∞ constraint, i.e.

$$\|\xi(t) - \hat{\xi}(t)\|_\infty < \epsilon, \forall t, \quad (17)$$

and check that M^{-1} exists at each time.

D. Effect of feedback linearization

In comparison to the non-feedback linearizable case, the linearized dynamics (7) are independent of trajectory (and hence also of iteration). That is, in the non-feedback linearizable case [1], each iteration begins by constructing a Jacobian linearization of dynamics (1); this is superfluous in our case. As a consequence, large changes in auxiliary input z between iterations—which lead to large changes in state trajectory—are trivially consistent with the feedback linearized dynamics (7). By contrast, a large change in control u may take the nonlinear dynamics (1) far away from the previous Jacobian linearization, which causes the algorithm from [1] to be fairly

sensitive to step size η and trust region size ϵ. We study this sensitivity more carefully in Section VI.

Finally, it is important to note that while many systems of interest (e.g., manipulators, cars, and quadrotors) are feedback linearizable, this is not true of all systems. Additionally, there are two drawbacks of our algorithm that deserve mention. First, we must take care to avoid regions in which M^{-1} does not exist. We accomplish this by designing costs that penalize proximity to singularities. While this can potentially limit the range of behaviors, many motion problems naturally incorporate these costs. Second, the transformed costs $\ell_i(t;\xi;\dots)$ may have much more varied, extreme curvature than the original costs $\ell_i(t,x,\dots)$. In some cases, this can make Algorithm 1 sensitive to linesearch parameters η_0 and ϵ, even offsetting the benefits mentioned above. We defer further discussion and empirical study for Section VI.

VI. Results

In this section, we study the empirical performance of Algorithm 1. In Section VI-A, we quantify the improvements in algorithmic stability from Section V-D for an intersection scenario. In Section VI-B, we discuss a case in which the extreme curvature of the transformed cost $\ell_i(t;\xi;\dots)$ alluded to also in Section V-D which causes Algorithm 1 to converge very slowly. In practice, however, this is not necessarily a serious problem. In Section VI-C, we redesign this problematic cost function to depend explicitly upon ξ rather than x without changing the semantic character of equilibria.

A. Improvements in solver stability

To showcase the benefits of our feedback linearization-based approach, we study the empirical sensitivity of solutions to the initial step size η_0 and trust region size ϵ hyperparameters from Section V-C. We shall consider a three-player intersection example and compare the strategies identified by Algorithm 1 with those identified on the original dynamics, using the algorithm from [1]. Here, two cars, modeled with bicycle dynamics

$$\begin{bmatrix}\dot{p}_x\\ \dot{p}_y\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix} v\cos\theta\\ v\sin\theta\\ (v/L)\tan\phi\end{bmatrix}, \quad \begin{bmatrix}\dot{v}\\ \dot{\phi}\\ \dot{a}\end{bmatrix} = \begin{bmatrix} a\\ \omega\\ \kappa\end{bmatrix} \tag{18}$$

(with inter-axle distance L and inputs ω controlling front wheel rate $\dot{\phi}$ and κ controlling jerk), and a pedestrian modeled with dynamics (8) navigate an intersection. Like (8), bicycle dynamics (18) are feedback linearizable in the outputs (p_x, p_y). We place quadratic penalties on each player's distance from the appropriate lane center and from a fixed goal location, as well as on the difference between speed v and a fixed nominal speed $\bar{v}$. Players are also penalized quadratically within a fixed distance of one another [1].

[1]For details concerning weighing of different cost terms we refer the reader to our github repository at `https://github.com/HJReachability/ilqgames`

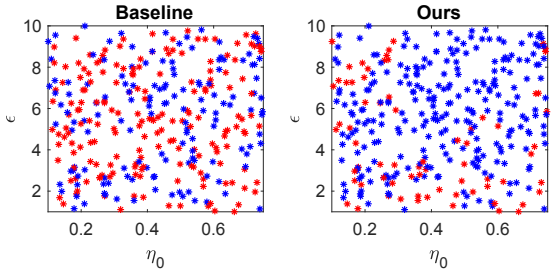

Fig. 2: Distribution of pairs (η_0, ϵ) colored by quality metric q. Pairs with low q are colored blue, and high q pairs are colored red.

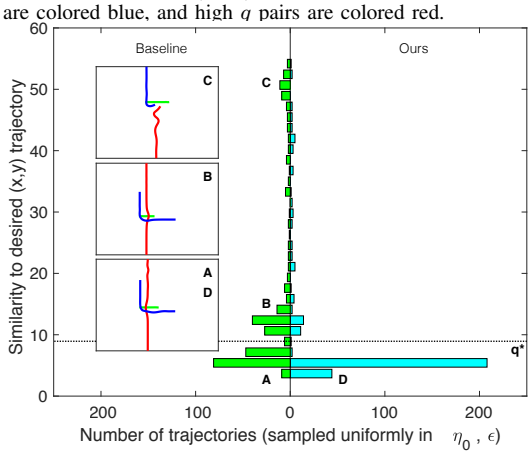

Fig. 3: Comparison of the proposed algorithm with the state of the art [1] for a three player intersection game. Histograms (left, baseline; right, ours) show that our method is much more numerically stable and converges more frequently. Insets labelled {A, B, C, D} show a typical trajectory for the associated bin. The dotted horizontal line shows threshold q^*, used to distinguish samples from Fig. 2.

In order to assess the quality of a trajectory $\mu = (\xi, z_1, \dots, z_N)$ generated by a particular algorithm, we define the *similarity metric* to the desired trajectory to be:

$$q(\mu, \tilde{\mu}) := \max_{t\in[0,T]} \|\xi(t) - \tilde{\xi}(t)\|_{2,(p_x,p_y)}. \tag{19}$$

Here, we take $\tilde{\mu} := (\tilde{\xi}, \tilde{z}_1, \dots, \tilde{z}_N)$ to be the equilibrium trajectory which that algorithm ideally converges to. The norm measures Euclidean distance only in the (p_x, p_y) dimensions. Trajectories that diverge or converge to unreasonable solutions yield high values for q, while trajectories that closely match $\tilde{\mu}$ incur low values.

We fix the initial conditions and cost weights identically for both algorithms. Thus, any trajectory μ identified by the solver will solely be a function of the initial step size η_0 and trust region size ϵ. Therefore, we will overload the penalty metric notation as $q(\eta_0;\epsilon)$. Given this metric we study the quality of solutions over the ranges $\eta_0 \in [0.1, 0.75]$ and $\epsilon \in [1.0, 10.0]$, and test 324 uniformly sampled (η_0, ϵ) pairs.

Fig. 2 displays the sampled pairs over the space of η_0 and ϵ. For clarity, we set a *success threshold* q^* and color "successful" pairs with $q(\eta_0;\epsilon) \le q^*$ blue, and "unsucessful" pairs red. Fig. 3 shows histograms of solution quality q for each algorithm, with a horizontal line denoting threshold q^*. We observe that solving the game using feedback linearization converges much more reliably than solving it for the original nonlinear system. Moreover, for converged trajectories with low q-value, the average computation time

was 0.3982 ± 0.3122 s (mean $\pm$ standard deviation) for our method and 0.8744 ± 0.9582 s for the baseline.

B. Sensitivity to transformed cost landscape

Unfortunately, these results do not generalize to all games. As per Section V-D, in some cases the cost landscape gets much more complicated when expressed in linearized system coordinates ξ, z_i. For example, a simple quadratic penalty on a single player's speed difference from nominal $\bar{v}$ in (8) is nonconvex and non-smooth near the origin when expressed as a function of linearized system state ξ:

$$(v - \bar{v})^2 \iff \left(\bar{v} - \sqrt{\dot{p}_x^2 + \dot{p}_y^2}\right)^2. \tag{20}$$

Consequences vary; the effect is negligible in the intersection example from Fig. 3, but it is more significant in the roundabout example below in Section VI-C, where cars must slow down before turning into the roundabout.

C. Designing costs directly for the linearized system

Fortunately, in practical settings of interest it is typically straightforward to design smooth, semantically equivalent costs explicitly as functions of the linearized system coordinates ξ. For example, we can replace the nominal speed cost of (20) with a time-varying quadratic penalty in that player's position (p_x, p_y):

$$(v - \bar{v})^2 \implies \big(p_x(t) - \bar{p}_x(t)\big)^2 + \big(p_y(t) - \bar{p}_y(t)\big)^2, \tag{21}$$

where $\big(\bar{p}_x(\cdot), \bar{p}_y(\cdot)\big)$ defines the point on the lane center a distance $\bar{v}t$ from the initial condition.

We demonstrate the effectiveness of this substitution in two examples—merging into a roundabout, and overtaking a lead vehicle—in which the original cost (20) led to instability in Algorithm 1. In both cases, we also use simple quadratic penalties for z_i (rather than transforming $\|u_i\|^2$ into linearized coordinates), albeit with different weightings. Results for the roundabout merging and overtaking examples are shown in Figures 4 and 5, respectively. From the 324 samples in each (drawn from expanded ranges $\eta_0 \in [0.1, 1.0], \epsilon \in [1, 50]$), we see that Algorithm 1 converged more frequently than the method of [1]. Moreover, when successful, the average computational time in the roundabout example was 0.2797 ± 0.1274 s for our method and 0.4244 ± 0.5259 s for the baseline. Runtimes for the overtaking example were 0.5112 ± 0.3228 s (ours) and 0.4417 ± 0.4142 s (baseline). Observe how runtimes for our approach cluster more tightly around the mean, indicating a more reliable convergence rate.

VII. Conclusion

We have presented a novel algorithm for identifying local equilibria in differential games with feedback linearizable dynamics. Our method works by repeatedly solving LQ games in the linearized system coordinates, rather than in the original system coordinates. By working with the linearized system, our algorithm becomes less sensitive to parameters such as initial step size and trust region size, which often leads it to converge faster. Our method is fully general, i.e.

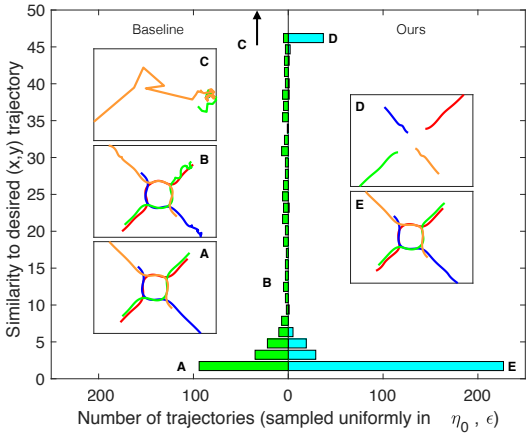

Fig. 4: Comparison for a roundabout merging example with four cars.

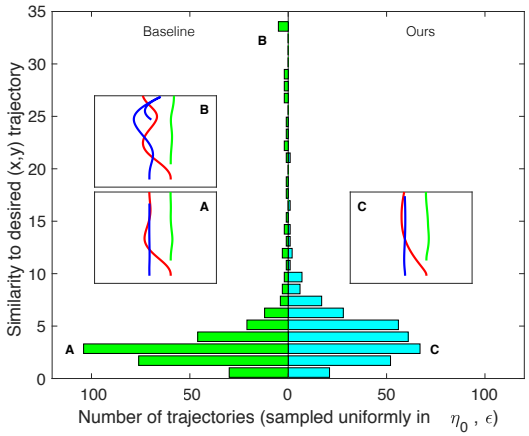

Fig. 5: Comparison for a three vehicle high speed overtaking maneuver.

any cost expressed in terms of nonlinear system coordinates may also be expressed in terms of linearized coordinates. However, in some cases transforming costs in this way makes the cost landscape extremely complicated. In such cases, it is often possible to design semantically equivalent replacement costs directly in the linearized coordinates. We test our method in a variety of competitive traffic scenarios. Using appropriately redesigned costs when necessary, our experiments confirm the computational stability and efficiency of our approach.

References

[1] D. Fridovich-Keil et al. "Efficient Iterative Linear-Quadratic Approximations for Nonlinear Multi-Player General-Sum Differential Games". *arXiv preprint arXiv:1909.04694* (2019).

[2] R. Isaacs. *Games of pursuit*. Tech. rep. Rand Corporation, 1951.

[3] A. W. Starr and Y.-C. Ho. "Nonzero-sum differential games". *Journal of Optimization Theory and Applications* 3.3 (1969).

[4] A. Starr and Y.-C. Ho. "Further properties of nonzero-sum differential games". *Journal of Optimization Theory and Applications* 3.4 (1969).

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific Belmont, MA, 1996.

[6] A. Dreves and M. Gerdts. “A generalized Nash equilibrium approach for optimal control problems of autonomous cars”. *Optimal Control Applications and Methods* 39.1 (2018).

[7] A. Dreves. “A best-response approach for equilibrium selection in two-player generalized Nash equilibrium problems”. *Optimization* 68.12 (2019).

[8] G. Williams et al. “Best response model predictive control for agile interactions between autonomous ground vehicles”. *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018.

[9] D. Sadigh et al. “Planning for autonomous cars that leverage effects on human actions.” *Robotics: Science & Systems*. Ann Arbor, MI, USA. 2016.

[10] Z. Wang, R. Spica, and M. Schwager. “Game Theoretic Motion Planning for Multi-robot Racing”. *Distributed Autonomous Robotic Systems*. Springer, 2019.

[11] H. Mukai et al. “Sequential linear quadratic method for differential games”. *Proc. 2nd DARPA-JFACC Symposium on Advances in Enterprise Control*. Citeseer. 2000.

[12] W. Li and E. Todorov. “Iterative linear quadratic regulator design for nonlinear biological movement systems.” *ICINCO*. 2004.

[13] D. H. Jacobson and D. Q. Mayne. “Differential dynamic programming” (1970).

[14] J. van den Berg. “Iterated LQR smoothing for locally-optimal feedback control of systems with non-linear dynamics and non-quadratic cost”. *American Control Conference (ACC)*. IEEE. 2014.

[15] A. Jonsson and M. Rovatsos. “Scaling up multiagent planning: A best-response approach”. *International Conference on Automated Planning and Scheduling*. 2011.

[16] M. Wang et al. “Game Theoretic Planning for Self-Driving Cars in Competitive Scenarios”. *Robotics: Science & Systems*. 2019.

[17] R. M. Murray and S. S. Sastry. “Nonholonomic motion planning: Steering using sinusoids”. *Transactions on Automatic Control* 38.5 (1993).

[18] P. Rouchon et al. “Flatness, motion planning and trailer systems”. *Conference on Decision and Control (CDC)*. Vol. 3. IEEE. 1993.

[19] D. Mellinger and V. Kumar. “Minimum snap trajectory generation and control for quadrotors”. *International Conference on Robotics and Automation*. IEEE. 2011.

[20] C. Richter, A. Bry, and N. Roy. “Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments”. *Robotics Research*. Springer, 2016.

[21] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. SIAM, 1999.

[22] S. Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Springer, 1999.

[23] Y. Tassa, N. Mansard, and E. Todorov. “Control-limited differential dynamic programming”. *International Conference on Robotics and Automation (ICRA)*. IEEE. 2014.