Solving Multi-Player Interactions using Linear Quadratic and Iterative Linear Quadratic Games

Mohamed Safwat **Ahmed Khalil** **Henry Chang**
CSE 579: Intelligent Control Through Learning & Optimization

Abstract

In many settings where multiple agents interact, each agent's optimal choices depend heavily on others' choices. For successful decentralized multi-agent robot collaboration, each agent must be able to reason about their decisions' impact on others' decisions. Differential game theory provides a principled formalism for expressing these coupled interactions, however, traditional numerical solution techniques scale poorly with state dimension and are therefore not utilized in real-time applications. As inspired by the iterative quadratic regulator (ILQR), recent work offers efficient approximations to solve these problems. We implemented a Julia package, inspired by ilqgames [1], for solving nonlinear multi-player general-sum differential games using linear-quadratic approximations. The package is open-source and can be found at `https://github.com/itsahmedkhalil/iLQGameSolver.jl`

1 Introduction and Background

Ever since the industrial revolution, the impact of manufacturing technologies has profoundly affected our society. We were able to quickly and efficiently mass-produce essential products at a fraction of the cost. One of the most successful technologies introduced was automation, and this was one of the earliest applications of robotic systems. Today, industrial robots play a vital role in automated assembly lines, enabling rapid and efficient fabrication. Unfortunately, most assembly lines are relatively inflexible as they are designed to produce a particular product or a narrow range of products. For industrial robots to fabricate a wide variety of products, they must be able to quickly and cohesively coordinate their actions. While communication protocols could be set up for small groups of robots, they are unable to scale up to the hundreds or thousands of robots usually found in manufacturing settings.

An especially important factor for efficient robot collaboration is collision avoidance between robots, other agents, and static objects. This constraint is a crucial feature of planning and control for autonomous manufacturing settings as the factories are purposefully designed to maximize efficiency. Additionally, this collision-avoidance applies to whatever cargo the robot is manipulating or transporting. To maximize the output of production, a planner must compute decisions that allow individual robots and teams of robots to quickly and efficiently complete tasks. However, given the large scale of some of these manufacturing settings, a centralized planner would be infeasible due to the so-called "curse of dimensionality" [2].

One proposed method for solving decentralized collaborative robot interactions in manufacturing settings is to reformulate the problems as differential games. Often, these decentralized interactions are decoupled, thus meaning that each autonomous agent must predict the behaviors of others and act appropriately. This decoupling requires strong predictive assumptions on how agents' decisions impact one another. Differential game theory allows multi-agent decision-making problems to explicitly account for the mutual dependence of all agents' decisions without requiring *a priori* predictive assumptions [1].

The algorithm we implemented was proposed by Fridovich-Keil et al. in their paper "Efficient Iterative Linear-Quadratic Approximations for Nonlinear Multi-Player General-Sum Differential Games" [1]. The algorithm aims to find a local solution to the problem to avoid the curse of dimensionality that arises when searching for global Nash equilibria. The algorithm is built upon the iterative linear-quadratic regulator (ILQR) [3], a local method used in smooth optimal control problems [4, 5, 6]. Furthermore, by exploiting the efficient closed-form solution of LQ games [7], the algorithm can solve successive LQ approximations, thereby finding a local solution to the original game in real-time.

2 Derivation of Nash Equilibrium for Linear Quadratic Games with Feedback

The following is a derivation of the discrete linear-quadratic game with feedback information structure.

2.1 Problem Formulation

Suppose an N player discrete nonlinear dynamic game where the transition $f(x, u)$ is differentiable with respect to x, u:

$$x_{t+1} = f(x_t, u_t^1, u_t^2, \dots, u_t^N) \tag{1}$$

Where each of the N players is trying to minimize a stage additive cost with a non-quadratic cost function c that is twice differentiable with respect to both x and u:

$$L_i(u^1, u^2, \dots, u^N) = \sum_{t=0}^{T-1} c_i(x_t, u_t) \tag{2}$$

The global Nash equilibria is satisfied if and only if no player i has an incentive to deviate from its current set of strategies,

$$L_i^* := L_i(\gamma^{1*}, \gamma^{2*}, \dots, \gamma^{i*}, \dots, \gamma^{N*}) \leq L_i(\gamma^{1*}, \gamma^{2*}, \dots, \gamma^{i}, \dots, \gamma^{N*}) \tag{3}$$

Where γ^i is a set of strategies or policies about a trajectory that map states to actions, $\gamma_t^i : X \to U, i \in N, t \in T - 1$.

Note: Subscript t is used to represent time step, superscript i is used to represent a single player, and superscript j is $j \in N$. It should be assumed that any control input u without a superscript actually represents the entire control input commands of all players as a column vector.

2.2 Solution using linear-quadratic approximations

Using a first order Taylor series expansion about a proposed trajectory $(\hat{x}_t, \hat{u}_t)$, the dynamics can be approximated as follows,

$$f(\hat{x}_t + \delta x_t, \hat{u}_t + \delta u_t) \approx f(\hat{x}_t, \hat{u}_t) + \nabla_x f(\hat{x}_t, \hat{u}_t)\delta x_t + \nabla_u f(\hat{x}_t, \hat{u}_t)\delta u_t \tag{4}$$

Where $\nabla_x f(\hat{x}_t, \hat{u}_t) \in \mathbb{R}^{d_x \times d_x}$ is the Jacobian of the dynamics f with respect to the state x and $\nabla_u f(\hat{x}_t, \hat{u}_t) \in \mathbb{R}^{d_x \times d_u}$ is the Jacobian of the dynamics f with respect to the control inputs u.

After linearization for all timesteps, we get a sequence of time-variant linear dynamical systems, where:

$$A_t := \nabla_x f(\hat{x}_t, \hat{u}_t), \quad B_t := \nabla_u f(\hat{x}_t, \hat{u}_t), \quad \forall t = 0, \dots, H-1 \tag{5}$$

Combining the equations, the dynamic system of the game can be written as:

$$\delta x_{t+1} = A_t \delta x_t + B_t \delta u_t$$

Where δx_t and δu_t are changes to the state, $x_t - \hat{x}_t$, and control input trajectories, $u_t - \hat{u}_t$, respectively.

A non-quadratic cost function for a single agent, is approximated using a second order Taylor expansion,

$$\begin{aligned} c(\hat{x}_t + \delta x_t, \hat{u}_t + \delta u_t) &\approx c(\hat{x}_t, \hat{u}_t) + \nabla_{x_t,u_t} c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix} \\ &\quad + \frac{1}{2} \begin{bmatrix} \delta x_t & \delta u_t \end{bmatrix} \nabla^2_{x_t,u_t} c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix} \\ c(\delta x_t, \delta u_t) &= \frac{1}{2}(\delta x_t^\intercal Q_t + 2 l_t^\intercal)\delta x_t + \frac{1}{2}(\delta u_t^\intercal R_t + 2 r_t^\intercal)\delta u_t + \frac{1}{2}\delta x_t^\intercal H_t \delta u_t + \frac{1}{2}\delta u_t^\intercal H_t \delta x_t \end{aligned} \tag{6}$$

Where the hessians are, $Q_t = \frac{\partial^2 c}{\partial x^2}$, $R_t = \frac{\partial^2 c}{\partial u^2}$, $H_t = \frac{\partial^2 c}{\partial u \partial x}$, and the gradients are, $l_t = \nabla_x c$, and $r_t = \nabla_u c$. For the derivation of the LQGame, we neglect, H_t since it is not included in any of our formulated cost functions. Furthermore, any constant terms can also be neglected.

The problem can be formulated as an LQGame problem with an affine term due to the gradients in the cost function. For a dynamic game, the cost function for agent i is given by,

$$c_i(\delta x_t, \delta u_t^1, ..., \delta u_t^N) = \frac{1}{2}(\delta x_t^\intercal Q_t^i + 2 l_t^{i\intercal})\delta x_t + \frac{1}{2}\sum_{j=1}^{N}(\delta u_t^{j\intercal} R_t^{ij} + 2 r_t^{ij\intercal})\delta u_t^j \tag{7}$$

Where R_t^{ij} is the cost of player j's actuation onto player i's cost function.

We can then formulate the value function for player i as the current cost + cost to go,

$$J_i^*(\delta x_t, t) = \min_{\delta u_i}[c_i(\delta x_t, \delta u_t) + J^*(\delta x_{t+1}, t+1)] \tag{8}$$

Assuming that the actuation at the final state is to be driven to zero, then, at the terminal state, $T-1$, the cost is only associated with the state,

$$J_i^*(\delta x_{T-1}, T-1) = \frac{1}{2}(\delta x_{T-1}^\intercal Q_{T-1}^i + 2 l_{T-1}^{i\intercal})\delta x_{T-1} \tag{9}$$

Where the next state value function has the following quadratic form,

$$J_i^*(\delta x_{T-1}, T-1) = \frac{1}{2}(\delta x_{T-1}^\intercal V_{T-1}^i + 2 \zeta_{T-1}^{i\intercal})\delta x_{T-1} \tag{10}$$

At the second to last time step, $T-2$, the value function is,

$$J_i^*(\delta x_{T-2}, T-2) = \min_{\delta u_i}[c(\delta x_{T-2}, \delta u_{T-2}) + J_i^*(\delta x_{T-1}, T-1)] \tag{11}$$

Substituting in the quadratic cost function and the next state value function,

$$\begin{aligned} J_i^*(\delta x_{T-2}, T-2) = \min_{\delta u_i}\Bigg[&\frac{1}{2}(\delta x_{T-2}^\intercal Q_{T-2}^i + 2 l_{T-2}^{i\intercal})\delta x_{T-2} + \frac{1}{2}\sum_{j=1}^{N}(\delta u_{T-2}^{j\intercal} R_{T-2}^{ij} + 2 r_{T-2}^{ij\intercal})\delta u_{T-2}^j \\ &+ \frac{1}{2}(\delta x_{T-1}^\intercal V_{T-1}^i + 2\zeta_{T-1}^{i\intercal})\delta x_{T-1}\Bigg] \end{aligned} \tag{12}$$

Substituting in the linear dynamics and setting $t = T-2$ for notation simplicity,

$$\begin{aligned} J_i^*(x_t, t) = \min_{\delta u_i}\Bigg[&\frac{1}{2}(\delta x_t^\intercal Q_t^i + 2 l_t^{i\intercal})\delta x_t + \frac{1}{2}\sum_{j=1}^{N}(\delta u_t^{j\intercal} R_t^{ij} + 2 r_t^{ij\intercal})\delta u_t^j \\ &+ \frac{1}{2}((A_t \delta x_t + \sum_{j=1}^{N} B_t^j \delta u_t^j)^\intercal V_{t+1}^i + 2\zeta_{t+1}^{i\intercal})(A_t \delta x_t + \sum_{j=1}^{N} B_t^j \delta u_t^j)\Bigg] \end{aligned} \tag{13}$$

We then take the derivative of the value function with respect to player i's deviation from its control input trajectory, and solve for the optimal control,

$$\frac{\partial}{\partial \delta u^i} J_i^*(\delta x_t, t) = 0 \tag{14}$$

$$0 = R_t^{ii}\delta u_t^i + r_t^{ii} + B_t^{i\top} V_{t+1}^i B_t^i \delta u_t^i + B_t^{i\top} V_{t+1}^i A_t \delta x_t + B_t^{i\top} V_{t+1}^i \sum_{j=1}^{N} B_t^j \delta u_t^j + B_t^{i\top}\zeta_{t+1}^i \tag{15}$$

Simplifying and rearranging,

$$(R_t^{ii} + B_t^{i\top} V_{t+1}^i B_t^i)\delta u_t^i + B_t^{i\top} V_{t+1}^i \sum_{j=1}^{N} B_t^j \delta u_t^j = -B_t^{i\top} V_{t+1}^i A_t \delta x_t - B_t^{i\top}\zeta_{t+1}^i \tag{16}$$

Notice the form of the optimal control input that minimizes the objective,

$$\delta u_t^{i*} = -P_t^i \delta x_t - \alpha_t^i \tag{17}$$

Where P_t^i is analogous to an optimal proportional gain, and α_t^i is a feedforward term.
Substituting 17 into 16,

$$(R_t^{ii} + B_t^{i\top} V_{t+1}^i B_t^i)(-P_t^i \delta x_t - \alpha_t^i) + B_t^{i\top} V_{t+1}^i \sum_{j=1}^{N} B_t^j (-P_t^j \delta x_t - \alpha_t^j) = -B_t^{i\top} V_{t+1}^i A_t \delta x_t - B_t^{i\top}\zeta_{t+1}^i \tag{18}$$

the system of linear equations in 18 can be rewritten as,

$$(R_t^{ii} + B_t^{i\top} V_{t+1}^i B_t^i)P_t^i + B_t^{i\top} V_{t+1}^i \sum_{j=1}^{N} B_t^j P_t^j = B_t^{i\top} V_{t+1}^i A_t \tag{19}$$

$$(R_t^{ii} + B_t^{i\top} V_{t+1}^i B_t^i)\alpha_t^i + B_t^{i\top} V_{t+1}^i \sum_{j=1}^{N} B_t^j \alpha_t^j = B_t^{i\top}\zeta_{t+1}^i \tag{20}$$

To solve for P_t^i and α_t^i, we substitute the optimal control input , u_t^{i*}, into the value function (equation 13) to obtain a recursive relation for V_t and ζ_t,

$$\begin{aligned} J_i^*(x_t, t) = {} & \frac{1}{2}(\delta x_t^\top Q_t^i + 2 l_t^{i\top})\delta x_t + \frac{1}{2}\sum_{j=1}^{N}((-P_t^j \delta x_t - \alpha_t^j)^\top R_t^{ij} + 2 r_t^{ij\top})(-P_t^j \delta x_t - \alpha_t^j) \\ & + \frac{1}{2}((A_t \delta x_t + \sum_{j=1}^{N} B_t^j(-P_t^j \delta x_t - \alpha_t^j))^\top V_{t+1}^i + 2\zeta_{t+1}^{i\top})(A_t \delta x_t + \sum_{j=1}^{N} B_t^j(-P_t^j \delta x_t - \alpha_t^j)) \end{aligned} \tag{21}$$

$$\begin{aligned} J_i^*(x_t, t) = {} & \frac{1}{2}(\delta x_t^\top Q_t^i + 2 l_t^{i\top})\delta x_t + \frac{1}{2}\sum_{j=1}^{N}((P_t^j \delta x_t + \alpha_t^j)^\top R_t^{ij} - 2 r_t^{ij\top})(P_t^j \delta x_t + \alpha_t^j) \\ & + \frac{1}{2}((A_t \delta x_t - \sum_{j=1}^{N} B_t^j(P_t^j \delta x_t + \alpha_t^j))^\top V_{t+1}^i + 2\zeta_{t+1}^{i\top})(A_t \delta x_t - \sum_{j=1}^{N} B_t^j(P_t^j \delta x_t + \alpha_t^j)) \end{aligned} \tag{22}$$

Simplifying further to match the form of the value function (equation 10) and neglecting the constant term,

$$\begin{aligned} J_i^*(x_t, t) = {} & \frac{1}{2}\delta x_t^\top (Q_t^i + \sum_{j=1}^{N} P_t^{j\top} R_t^{ij} P_t^j + (A_t \delta x_t - \sum_{j=1}^{N} B_t^j P_t^j)^\top V_{t+1}^i (A_t \delta x_t - \sum_{j=1}^{N} B_t^j P_t^j))\delta x_t \\ & + (l_t^i + \sum_{j=1}^{N}(P_t^{j\top} R_t^{ij}\alpha_t^j - P_t^{j\top} r_t^{ij}) + (A_t \delta x_t - \sum_{j=1}^{N} B_t^j P_t^j)^\top (\zeta_{t+1}^i - V_{t+1}^i \sum_{j=1}^{N} B_t^j \alpha_t^j))^\top \delta x_t \end{aligned} \tag{23}$$

The value function can be updated recursively as,

$$V_t^i \leftarrow Q_t^i + \sum_{j=1}^{N} P_t^{j\top} R_t^{ij} P_t^j + (A_t \delta x_t - \sum_{j=1}^{N} B_t^j P_t^j)^\top V_{t+1}^i (A_t \delta x_t - \sum_{j=1}^{N} B_t^j P_t^j) \quad (24)$$

$$\zeta_t^i \leftarrow l_t^i + \sum_{j=1}^{N} (P_t^{j\top} R_t^{ij} \alpha_t^j - P_t^{j\top} r_t^{ij}) + (A_t \delta x_t - \sum_{j=1}^{N} B_t^j P_t^j)^\top (\zeta_{t+1}^i - V_{t+1}^i \sum_{j=1}^{N} B_t^j \alpha_t^j) \quad (25)$$

3 ILQGame

The general idea of the Iterative LQGame (ILQGame) is to solve for locally optimal changes in trajectories for nonlinear systems by locally approximating the dynamics model with a linear or affine function and the cost with a quadratic function (equation 4 and 6, respectively). The local approximations are used to solve the LQGame iteratively for a locally optimal Nash equilibrium.

3.1 Algorithm

The ILQGame algorithm can be summarized in the following iterative procedure. First, an initial dynamically feasible trajectory is proposed $(\hat{x}_0, \hat{u}_0, ..., \hat{x}_{T-1}, \hat{u}_{T-1})$. Note that we do this step by randomly initiating a sequence of controls, which can sometimes lead to undesired locally optimal solutions or even no convergence. The dynamics and cost function are then linearized and quadriticized about the proposed trajectory and used to solve the affine quadratic game as shown in section 2.2. The obtained feedback and feedforward matrices in the control policy (equation 17) are used to compute the sequence of states $(\hat{x}_0, ..., \hat{x}_{T-1})$ by rolling out the nonlinear dynamics using an explicit RK4. The process is repeated until convergence.

3.2 Regularization

After performing the second order Taylor expansion at some (x_t, u_t), the cost matrices Q_t and R_t might not be positive semi-definite. We attribute the negative semi-definiteness of Q_t to the numerically ill-conditioned penalty method used for collision constraints. Therefore, Q_t must be regularized to solve the affine game appropriately. A pseoducode of the regularization is shown below.

Algorithm 1 Regularization

```
1: Init $\beta > 0$
2: while $Q_t \prec 0$ do
3:     $Q_t \leftarrow Q_t + \beta I$
4: end while
```

3.3 Line Search

Optimizing nonlinear functions can often lead to undesirable behaviors, such as overshooting a local minimum. Therefore, it is required to regulate the direction of gradient descent. We do so by including a hyperparameter that acts as a step size on α in equation 17.

3.4 Convergence

For convergence, we check the norm between the changes to the state $x_t - \hat{x}_t$ trajectories and compare it to a specified tolerance. If it is less than the tolerance, the algorithm is converged.

4 Results

The algorithms were tested on a number of games with different agents, including point-masses, differential drive mobile robots, and quadcopters. The following results detail the algorithm's performance on point-masses and quadcopters. All the tests were intersection games where each agent attempts to reach their pre-specified final goal state and actuation effort without colliding with other agents. Furthermore, the algorithms were implemented in a feedback control strategy manner using a receding horizon controller.

4.1 Cost Function

The cost function for each agent was composed of four different parts: state tracking error, actuation/input effort, terminal state error, and collision avoidance.

$$\min_{\substack{\delta x_{0:T-1}\\ \delta u_{0:T-2}}} \underbrace{\delta x_H^\mathsf{T} Q_N \delta x_H}_{\text{Terminal state error}} + \sum_{t=0}^{T-1} \left[\underbrace{\delta x_t^\mathsf{T} Q \delta x_t}_{\text{State tracking error}} + \underbrace{\delta u_t^\mathsf{T} R^{ij} \delta u_t}_{\text{Actuation effort}} + \underbrace{\rho || \min(d_t^{ij}(x_t) - d_m, 0)||^2}_{\text{Collision Penalty}} \right]$$

Where d_t^{ij} is the sum of distances between the i^{th} agent and all other j agents. For example, in a 3-player 2D game, d_t^{ij}, would be defined as:

$$d_t^{ij} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}$$

The threshold distance that all agents should keep between one another is d_m and any violation to that constraint will lead to a penalty multiplier, ρ.

4.2 Point Mass Games

The states and inputs for the linear 2D point mass dynamics for each agent are defined as:

$$x = [x, y, \dot{x}, \dot{y}]^T \tag{26}$$

$$u = [F_x, F_y]^T \tag{27}$$

And the system model is defined by:

$$\dot{x} = f(x(t), u(t)) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ -\frac{b}{m}\dot{x} + \frac{1}{m}F_x \\ -\frac{b}{m}\dot{y} + \frac{1}{m}F_y \end{bmatrix} \tag{28}$$

Where the variables b, m, and F_{ext} are the damping coefficients [N-s/m], mass of the agent [kg], and the input force [N], respectively.

In this game, three players with point mass dynamics attempt to reach their final goal states, indicated with the cross symbol, without colliding with one another. Figure 1 shows the results of running the algorithm once until convergence over 10 seconds, the entire horizon of the game, with a time step of 0.1 seconds.

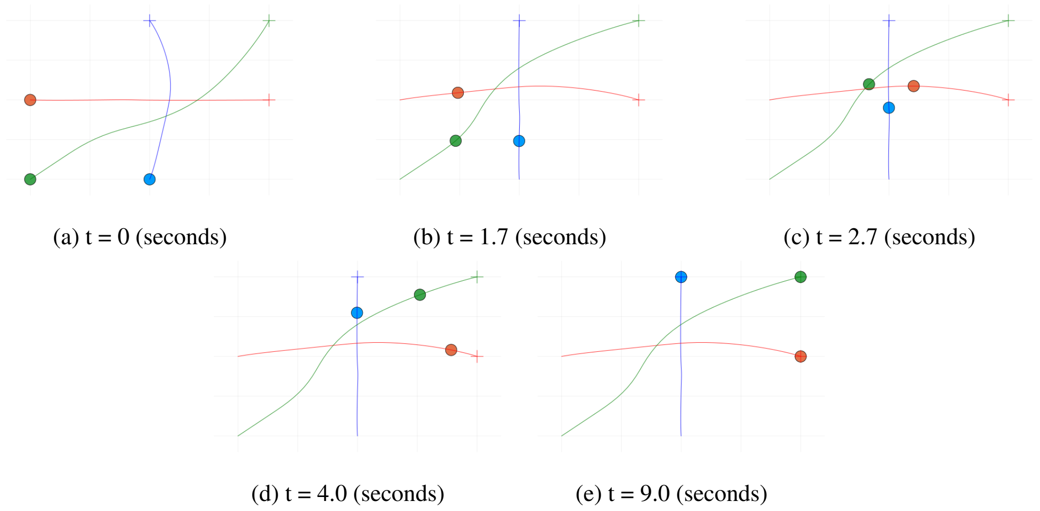

(a) t = 0 (seconds)
(b) t = 1.7 (seconds)
(c) t = 2.7 (seconds)
(d) t = 4.0 (seconds)
(e) t = 9.0 (seconds)

Figure 1: Time-lapse of simulation for a 3-agent point mass game

The algorithm optimized the actions of all players over the entire game duration in an average 23 seconds. The convergence time depended heavily on the open-loop initial strategies each game started with. Figure 2 shows the Monte Carlo results for the same 3-player game shown above with 100 random Gaussian open-loop initial strategies.

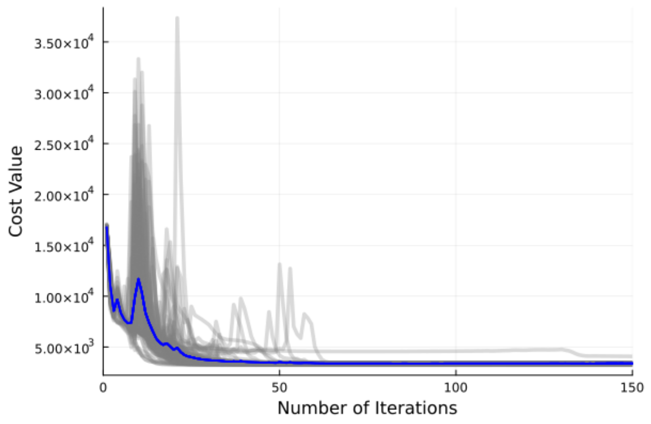


Figure 2: Monte Carlo results for 100 random initial strategies

As shown in Figure 2, the games, on average, tend to converge well within 100 iterations.

4.3 Quadcopter Games

The states and inputs for the nonlinear quadcopter dynamics of each agent are defined as:

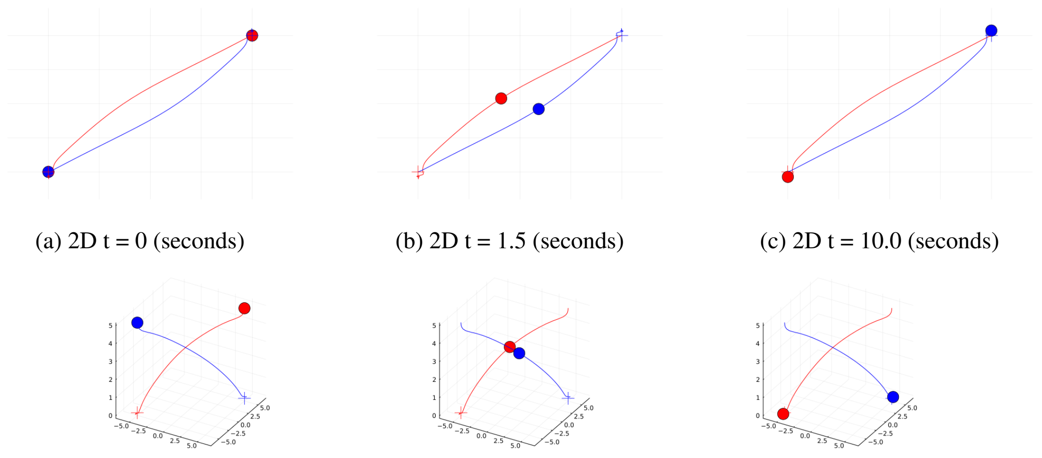
$$x = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \Psi]^T \tag{29}$$

$$u = [\dot{z}_{cmd}, \phi_{cmd}, \theta_{cmd}, \dot{\Psi}_{cmd}]^T \tag{30}$$

And the nonlinear system model is defined by:

$$\dot{x} = f(x(t), u(t)) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \frac{sin(\phi)sin(\Psi)+cos(\phi)sin(\theta)cos(\Psi)(\ddot{z}+g)}{cos(\theta)cos(\phi)} \\ \frac{-sin(\phi)sin(\Psi)+cos(\phi)sin(\theta)sin(\Psi)(\ddot{z}+g)}{cos(\theta)cos(\phi)} \\ \frac{1}{\tau_z}(\dot{z}_{cmd} - \dot{z}) \\ \frac{1}{\tau_\phi}(\phi_{cmd} - \phi) \\ \frac{1}{\tau_\theta}(\theta_{cmd} - \theta) \\ \dot{\Psi}_{cmd} \end{bmatrix} \tag{31}$$

In this game, two players with nonlinear quadcopter dynamics attempt to reach their final goal states, indicated with the cross symbol, without colliding with one another. Unfortunately, running the algorithm once over the entire game duration took too much time to converge. In response, we implemented a receding horizon version of the algorithm, which was significantly faster and more robust. Figure 3 shows the results of running the algorithm with a receding horizon of 2.5 seconds with a time step of 0.1 seconds for an entire game duration of 10 seconds. Figures 3a to 3c show the top view of the quadcopters (x-y plane) and Figures 3d to 3f show the 3D view of the quadcopters.



(a) 2D t = 0 (seconds) (b) 2D t = 1.5 (seconds) (c) 2D t = 10.0 (seconds)

(d) 3D t = 0 (seconds) (e) 3D t = 1.5 (seconds) (f) 3D t = 10.0 (seconds)

Figure 3: Time-lapse of simulation for a 2-agent quadcopter game

With a receding horizon of 2.5 seconds, the controller ran at a rate of 33 Hz, which would allow it to run on hardware in real-time. All computation times are reported for single-threaded operation on a 2021 MacBook Air with an M1 chip, to simulate the performance of the algorithms on an average computer.

5 Conclusion

We implemented a fast and generalizable Julia package that can solve nonlinear multi-agent general-sum differential games using iterative linear-quadratic approximations. The package is capable of handling multiple agents with a variety of dynamic systems, including quadcopters. The primary algorithm linearizes the dynamics, quadraticizes the game's costs, and solves for the Nash equilibrium strategies using backward recursion of the coupled Riccati equations. Additionally, a receding horizon variation of the algorithm was implemented and showed improvements in speed and robustness for systems with large and complex dynamics. Next steps will include testing this algorithm on hardware in a manufacturing setting to evaluate its performance. We would also like to relax some assumptions for real-world applications. For example, our current implementation assumes full state observability and explicitly gives each agent the objectives of others. However, in a practical decentralized setting, agents would need to infer other agents' states and objectives.

References

[1] David Fridovich-Keil, Ellis Ratner, Lasse Peters, Anca D. Dragan, and Claire J. Tomlin. Efficient Iterative Linear-Quadratic Approximations for Nonlinear Multi-Player General-Sum Differential Games, 2019.

[2] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957.

[3] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. volume 1, pages 222–229, 01 2004.

[4] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard. Whole-body model-predictive control applied to the hrp-2 humanoid. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3346–3351, 2015.

[5] Nikita Kitaev, Igor Mordatch, Sachin Patil, and Pieter Abbeel. Physics-based trajectory optimization for grasping in cluttered environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3102–3109, 2015.

[6] Jianyu Chen, Wei Zhan, and Masayoshi Tomizuka. Constrained iterative lqr for on-road autonomous driving motion planning. pages 1–7, 10 2017.

[7] Tamer Başar and Geert Jan Olsder. *Dynamic Noncooperative Game Theory, 2nd Edition*. Society for Industrial and Applied Mathematics, 1998.