Ben-Gurion University of the Negev
The Faculty of Natural Sciences
The Department of Computer Science

Differential Games for Compositional Handling of Competing Control Tasks

September 2022

Joshua Shay Kricheli

Under the supervision of
Prof. Gera Weiss, The Department of Computer Science
and **Dr. Shai Arogeti**, The Department of Mechanical Engineering

Introduction

- We introduce a divide and conquer control design methodology for single-agent, multi-objective dynamical systems

Introduction

- We introduce a divide and conquer control design methodology for single-agent, multi-objective dynamical systems
- The approach starts with associating each pre-described control objective with a corresponding virtual input that is presumed to act upon the system

Introduction

- We introduce a divide and conquer control design methodology for single-agent, multi-objective dynamical systems
- The approach starts with associating each pre-described control objective with a corresponding virtual input that is presumed to act upon the system
- Then we associate a virtual cost functional to each virtual input, providing each objective a set of weighting parameters

Introduction

- Next we consider a non-cooperative, non-zero-sum differential game between the virtual inputs who represent the players

Introduction

- Next we consider a non-cooperative, non-zero-sum differential game between the virtual inputs who represent the players
- Finally, guarantying a Nash Equilibrium between the players allows a modular, yet simple design of complex controllers

Introduction

- Next we consider a non-cooperative, non-zero-sum differential game between the virtual inputs who represent the players
- Finally, guarantying a Nash Equilibrium between the players allows a modular, yet simple design of complex controllers
- In order to demonstrate the method motivation and application, we will now show a simple introductory example

Main Contributions

This study provides the following core contributions:

1. Novel formulation for controllers that apply for single-agent, multi-objective dynamic systems, by solving non-cooperative, non-zero-sum differential games for their Nash Equilibria, in continuous-time control systems

Main Contributions

This study provides the following core contributions:

1. Novel formulation for controllers that apply for single-agent, multi-objective dynamic systems, by solving non-cooperative, non-zero-sum differential games for their Nash Equilibria, in continuous-time control systems

2. Extending the aforementioned theoretical basis and formal mathematical formulation of the technique of single-agent multi-objective Nash Equilibria, for direct-design discrete-time control systems

3. Development of an open-source Python package named *PyDiffGame*, implementing the proposed method, both for the continuous and discrete-time case

3. Development of an open-source Python package named *PyDiffGame*, implementing the proposed method, both for the continuous and discrete-time case

4. Derivation of a novel method for solving matrix algebraic Riccati equations (AREs) by converting them to differential Riccati equations (DREs) and solving them repetitively until convergence

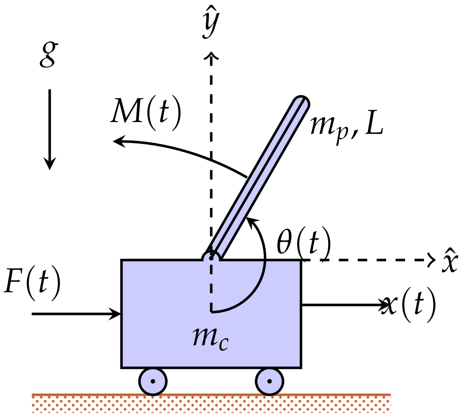
3. Development of an open-source Python package named *PyDiffGame*, implementing the proposed method, both for the continuous and discrete-time case

4. Derivation of a novel method for solving matrix algebraic Riccati equations (AREs) by converting them to differential Riccati equations (DREs) and solving them repetitively until convergence

5. Implementing the method of solving AREs by reduction to DREs in the Python package *PyDiffGame*

Motivating Example

Consider the following modified inverted pendulum system:



For any $t \in \mathbb{R}^{\geq 0}$:

- $x(t) \in \mathbb{R}$ - cart position
- $F(t) \in \mathbb{R}$ - linear force
- $\theta(t) \in \mathbb{R}$ - pendulum angle
- $M(t) \in \mathbb{R}$ - pure torque
- $m_c, m_p \in \mathbb{R}^+$ - cart and pendulum masses
- $L \in \mathbb{R}^+$ - pendulum length
- g - gravity constant

System State Vector

The number of variables required to define the system is $n = 4$ and thus let the state vector $\mathbf{x}(t) \in \mathbb{R}^n = \mathbb{R}^4$ of the system be defined as such:

$$\mathbf{x}(t) := \begin{bmatrix} x(t) \\ \theta(t) \\ \dot{x}(t) \\ \dot{\theta}(t) \end{bmatrix}$$

for any $t \in \mathbb{R}^{\geq 0}$

System Initial Condition

For simplicity, let us assume a zero initial condition for the system:

$$\mathbf{x}(0) = \begin{bmatrix} x(0) \\ \theta(0) \\ \dot{x}(0) \\ \dot{\theta}(0) \end{bmatrix} := \mathbf{0_n} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

System Terminal Requirements

Let us assume it is required to converge to a specific terminal state vector $\mathbf{x}_\infty$ with desirable values for x and θ and zero velocities, i.e., we require:

$$\mathbf{x}_\infty := \lim_{t \to \infty} \mathbf{x}(t) = \lim_{t \to \infty} \begin{bmatrix} x(t) \\ \theta(t) \\ \dot{x}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} x_\infty \\ \theta_\infty \\ 0 \\ 0 \end{bmatrix}$$

for some constants $x_\infty \in \mathbb{R}, \theta_\infty \in [0, 2\pi]$

System Input

The number of non-dependant actuators acting upon the system is $m = 2$ and thus let the input vector $\mathbf{u}(t) \in \mathbb{R}^m = \mathbb{R}^2$ of the system be defined as such:

$$\mathbf{u}(t) := \begin{bmatrix} F(t) \\ M(t) \end{bmatrix}$$

for any $t \in \mathbb{R}^{\geq 0}$

Linearized System Model

In this study we show the state space model of the described system can be linearized to adhere the following Linear Time-Invariant (LTI) model:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$$

for any $t \in \mathbb{R}^{\geq 0}$

Linearized System Model

In this study we show the state space model of the described system can be linearized to adhere the following Linear Time-Invariant (LTI) model:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$$

for any $t \in \mathbb{R}^{\geq 0}$ and with[1]:

- $A \in \mathbb{R}^{n \times n} = R^{4 \times 4}$ being the dynamics matrix
- $B \in \mathbb{R}^{n \times m} = R^{4 \times 2}$ being the input matrix

[1] Both A and B of the described system are formally derived in this study

Linearized System Model Matrices

The matrices A and B are of the following form:

$$A := \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{3}{1+4\frac{m_c}{m_p}} g & 0 & 0 \\ 0 & \frac{6}{1+\frac{3}{1+\frac{1}{\frac{m_c}{m_p}}}} \frac{g}{L} & 0 & 0 \end{bmatrix} \; ; \; B := \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{4}{1+4\frac{m_c}{m_p}} \frac{1}{m_p} & \frac{6}{1+4\frac{m_c}{m_p}} \frac{1}{L m_p} \\ \frac{6}{1+4\frac{m_c}{m_p}} \frac{1}{L m_p} & \frac{6}{\frac{1}{2}+\frac{3}{1+\frac{m_p}{m_c}}} \frac{1}{L^2 m_p} \end{bmatrix}$$

Linearized System Model Matrices

Let us denote:

$$A := \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \underbrace{\frac{3}{1+4\zeta}g}_{a_{32}} & 0 & 0 \\ 0 & \underbrace{\frac{6}{1+\frac{3}{1+\frac{1}{\zeta}}}\frac{g}{L}}_{a_{42}} & 0 & 0 \end{bmatrix} ; B := \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \underbrace{\frac{4}{1+4\zeta}\frac{1}{m_p}}_{b_{31}} & \underbrace{\frac{6}{1+4\zeta}\frac{1}{Lm_p}}_{b_{32}} \\ \underbrace{\frac{6}{1+4\zeta}\frac{1}{Lm_p}}_{b_{32}} & \underbrace{\frac{6}{\frac{1}{2}+\frac{3}{1+\frac{1}{\zeta}}}\frac{1}{L^2m_p}}_{b_{42}} \end{bmatrix}$$

with $\zeta := \frac{m_c}{m_p}$

System Virtual Decomposition

We decompose the system using the following virtual inputs:

$$v_x(t) := \underbrace{\begin{bmatrix} b_{31} & b_{32} \end{bmatrix}}_{M_x} \underbrace{\begin{bmatrix} F(t) \\ M(t) \end{bmatrix}}_{\mathbf{u}(t)} = b_{31}F(t) + b_{32}M(t);$$

$$v_\theta(t) := \underbrace{\begin{bmatrix} b_{32} & b_{42} \end{bmatrix}}_{M_\theta} \underbrace{\begin{bmatrix} F(t) \\ M(t) \end{bmatrix}}_{\mathbf{u}(t)} = b_{32}F(t) + b_{42}M(t)$$

System Virtual Decomposition

We decompose the system using the following virtual inputs:

$$v_x(t) := \underbrace{\begin{bmatrix} b_{31} & b_{32} \end{bmatrix}}_{M_x} \underbrace{\begin{bmatrix} F(t) \\ M(t) \end{bmatrix}}_{\mathbf{u}(t)} = b_{31}F(t) + b_{32}M(t);$$

$$v_\theta(t) := \underbrace{\begin{bmatrix} b_{32} & b_{42} \end{bmatrix}}_{M_\theta} \underbrace{\begin{bmatrix} F(t) \\ M(t) \end{bmatrix}}_{\mathbf{u}(t)} = b_{32}F(t) + b_{42}M(t)$$

The intention is that $v_x(t) \in \mathbb{R}^{m_x} = \mathbb{R}^1 = \mathbb{R}$ is related to the dynamics of $x(t)$ and $v_\theta(t) \in \mathbb{R}^{m_\theta} = \mathbb{R}^1 = \mathbb{R}$ is related to the dynamics of $\theta(t)$

Augmented Virtual Inputs Vector

- Writing the virtual inputs in vector form:

$$\underbrace{\begin{bmatrix} v_x(t) \\ v_\theta(t) \end{bmatrix}}_{\mathbf{v}(t)} = \underbrace{\begin{bmatrix} M_x \\ M_\theta \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} F(t) \\ M(t) \end{bmatrix}}_{\mathbf{u}(t)}$$

Augmented Virtual Inputs Vector

- Writing the virtual inputs in vector form:

$$\underbrace{\begin{bmatrix} v_x(t) \\ v_\theta(t) \end{bmatrix}}_{\mathbf{v}(t)} = \underbrace{\begin{bmatrix} M_x \\ M_\theta \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} F(t) \\ M(t) \end{bmatrix}}_{\mathbf{u}(t)}$$

- We refer to $\mathbf{v}(t) \in \mathbb{R}^{\sum_{q\in\{x,\theta\}} m_q} = \mathbb{R}^2$ as the augmented virtual inputs vector of the equivalent decomposed system

Augmented Virtual Inputs Vector

- Writing the virtual inputs in vector form:

$$\underbrace{\begin{bmatrix} v_x(t) \\ v_\theta(t) \end{bmatrix}}_{\mathbf{v}(t)} = \underbrace{\begin{bmatrix} M_x \\ M_\theta \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} F(t) \\ M(t) \end{bmatrix}}_{\mathbf{u}(t)}$$

- We refer to $\mathbf{v}(t) \in \mathbb{R}^{\sum_{q\in\{x,\theta\}} m_q} = \mathbb{R}^2$ as the augmented virtual inputs vector of the equivalent decomposed system
- The augmented virtual inputs vector of the satisfies:

$$\mathbf{v}(t) = M\mathbf{u}(t)$$

Virtual Controller Design

- We refer to $M \in \mathbb{R}^{m \times \sum_{q \in \{x,\theta\}} m_q} = \mathbb{R}^{2\times 2}$ as the augmented division matrix of the aforementioned virtual decomposition[2]

[2] Notice each decomposition induces a (possibly) different value for M

Virtual Controller Design

- We refer to $M \in \mathbb{R}^{m \times \sum_{q \in \{x,\theta\}} m_q} = \mathbb{R}^{2\times 2}$ as the augmented division matrix of the aforementioned virtual decomposition[2]
- We can now compute a controller with regards to $\mathbf{v}(t)$, then roll back to $\mathbf{u}(t)$, under the condition that M is invertible, in which case we refer to the system as Inversely Designable or ID

[2]Notice each decomposition induces a (possibly) different value for M

Virtual Controller Design

- We refer to $M \in \mathbb{R}^{m \times \sum_{q \in \{x,\theta\}} m_q} = \mathbb{R}^{2\times 2}$ as the augmented division matrix of the aforementioned virtual decomposition[2]
- We can now compute a controller with regards to $\mathbf{v}(t)$, then roll back to $\mathbf{u}(t)$, under the condition that M is invertible, in which case we refer to the system as Inversely Designable or ID
- It can be shown that for any values of m_c, m_p and L, the modified inverted pendulum is always ID

[2] Notice each decomposition induces a (possibly) different value for M

Virtual Controller Design

- Our approach is more easy to implement when the system is ID, meaning defining $\mathbf{v}(t)$ guarantees a unique value for $\mathbf{u}(t)$

Virtual Controller Design

- Our approach is more easy to implement when the system is ID, meaning defining $\mathbf{v}(t)$ guarantees a unique value for $\mathbf{u}(t)$
- In the case where the system is not ID, then $\mathbf{u}(t)$:

Virtual Controller Design

- Our approach is more easy to implement when the system is ID, meaning defining $\mathbf{v}(t)$ guarantees a unique value for $\mathbf{u}(t)$
- In the case where the system is not ID, then $\mathbf{u}(t)$:
  - Either has no solution that satisfies $\mathbf{v}(t) = M\mathbf{u}(t)$

Virtual Controller Design

- Our approach is more easy to implement when the system is ID, meaning defining $\mathbf{v}(t)$ guarantees a unique value for $\mathbf{u}(t)$
- In the case where the system is not ID, then $\mathbf{u}(t)$:
  - Either has no solution that satisfies $\mathbf{v}(t) = M\mathbf{u}(t)$
  - Or it has infinitely many solutions

Virtual Controller Design

- Our approach is more easy to implement when the system is ID, meaning defining $\mathbf{v}(t)$ guarantees a unique value for $\mathbf{u}(t)$
- In the case where the system is not ID, then $\mathbf{u}(t)$:
    - Either has no solution that satisfies $\mathbf{v}(t) = M\mathbf{u}(t)$
    - Or it has infinitely many solutions
- In the first case, a designer must choose a different value for M

Virtual Controller Design

- Our approach is more easy to implement when the system is ID, meaning defining $\mathbf{v}(t)$ guarantees a unique value for $\mathbf{u}(t)$
- In the case where the system is not ID, then $\mathbf{u}(t)$:
  - Either has no solution that satisfies $\mathbf{v}(t) = M\mathbf{u}(t)$
  - Or it has infinitely many solutions
- In the first case, a designer must choose a different value for M
- In the second case, any value of $\mathbf{u}(t)$ satisfying $\mathbf{v}(t) = M\mathbf{u}(t)$ will suffice[3]

[3] A solution to $\mathbf{v}(t) = M\mathbf{u}(t)$ when M is singular can be found using numerical methods, such as computing the psuedoinverse of M

Equivalent Decomposed System

Using $\mathbf{v}(t)$, we get an equivalent decomposed system of the form:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{B_x} v_x(t) + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{B_\theta} v_\theta(t)$$

Equivalent Decomposed System

Using $\mathbf{v}(t)$, we get an equivalent decomposed system of the form:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \underbrace{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}_{B_x} v_x(t) + \underbrace{\begin{bmatrix}0\\0\\0\\1\end{bmatrix}}_{B_\theta} v_\theta(t)$$

which satisfies:

$$B\mathbf{u}(t) = B_x v_x(t) + B_\theta v_\theta(t) = \sum_{q\in\{x,\theta\}} B_q v_q(t)$$

Differential Game

- By designating appropriate cost functionals, the decomposed system induces a set of differential games

Differential Game

- By designating appropriate cost functionals, the decomposed system induces a set of differential games

- In each game, the players are the functions that define the virtual actuators $\left(v_q(\cdot)\right)_{q\in\{x,\theta\}}$, that compete by minimizing their own respective assigned virtual cost functional

Differential Game

- By designating appropriate cost functionals, the decomposed system induces a set of differential games
- In each game, the players are the functions that define the virtual actuators $\left(v_q(\cdot)\right)_{q\in\{x,\theta\}}$, that compete by minimizing their own respective assigned virtual cost functional
- We compute a Nash Equilibrium that balances between the objectives thus obtaining a set of optimal virtual inputs, as described in detail in this study

Virtual Cost Functionals

For $q \in \{x, \theta\}$, let us consider infinite horizon quadratic cost functionals of the following form:

$$J_q\Big(v_x(\cdot), v_\theta(\cdot)\Big) := \int_0^\infty \Big[\tilde{\mathbf{x}}(\tau)^T Q_q \tilde{\mathbf{x}}(\tau) + \tilde{v}_q^T(\tau) r_q \tilde{v}_q(\tau)\Big] \mathrm{d}\tau$$

Virtual Cost Functionals

For $q \in \{x, \theta\}$, let us consider infinite horizon quadratic cost functionals of the following form:

$$J_q\Big(v_x(\cdot), v_\theta(\cdot)\Big) := \int_0^\infty \Big[\tilde{\mathbf{x}}(\tau)^T Q_q \tilde{\mathbf{x}}(\tau) + \tilde{v}_q^T(\tau) r_q \tilde{v}_q(\tau)\Big] \mathrm{d}\tau$$

where:

- $\tilde{\mathbf{x}}(\tau) := \mathbf{x}_\infty - \mathbf{x}(\tau)$ is the vector state error for any $\tau \in \mathbb{R}^{\geq 0}$
- $Q_q \in \mathbb{R}^{n \times n} = \mathbb{R}^{4 \times 4}$ are semi-positive definite state weights
- $\tilde{v}_q(\cdot) := v_{q_\infty} - v_q(\cdot)$ where v_{q_∞} is the input law required to maintain $\mathbf{x}_\infty$, as in: $\lim_{\tau \to \infty} v_q(\tau) = v_{q_\infty}$, and $\dot{\mathbf{x}}_\infty = \mathbf{0} = A\mathbf{x}_\infty + \sum_{\psi \in \{x, \theta\}} B_\psi v_{q_\infty}$
- $r_q \in \mathbb{R}^{m_q \times m_q} = \mathbb{R}$ are positive virtual input weights

Open-Loop Nash Equilibrium

For the modified inverted pendulum, a pair of virtual inputs $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$ constitutes an Open-Loop Nash Equilibrium if for all $q \in \{x, \theta\}$ it is not possible to decrease the value of the cost functional $J_q\Big(v_x(\cdot), v_\theta(\cdot)\Big)$ only by changing its corresponding chosen virtual input $v_q^*(\cdot)$ to some other input $v_q(\cdot)$, while leaving $v_\psi(\cdot)$ intact, when $\psi \neq q$

Open-Loop Nash Equilibrium

Formally, the pair $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$ satisfies:

$$
\begin{aligned}
\forall v_x(\cdot)\ ;\ J_x\Big(v_x^*(\cdot), v_\theta^*(\cdot)\Big) &\leq J_x\Big(v_x(\cdot), v_\theta^*(\cdot)\Big);\\
\forall v_\theta(\cdot)\ ;\ J_\theta\Big(v_x^*(\cdot), v_\theta^*(\cdot)\Big) &\leq J_\theta\Big(v_x^*(\cdot), v_\theta(\cdot)\Big)
\end{aligned}
$$

Open-Loop Nash Equilibrium

Formally, the pair $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$ satisfies:

$$\forall v_x(\cdot)\ ;\ J_x\Big(v_x^*(\cdot), v_\theta^*(\cdot)\Big) \leq J_x\Big(v_x(\cdot), v_\theta^*(\cdot)\Big);$$
$$\forall v_\theta(\cdot)\ ;\ J_\theta\Big(v_x^*(\cdot), v_\theta^*(\cdot)\Big) \leq J_\theta\Big(v_x^*(\cdot), v_\theta(\cdot)\Big)$$

where for all $q \in \{x, \theta\}$, equality for J_q is obtained only when $v_q(\cdot) \equiv v_q^*(\cdot)$

Nash Equilibrium Solution

This study shows the Open-Loop Nash Equilibrium problem is solved by closed-loop constant feedback control policies $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$ of the following form:

$$v_q^*(\cdot) := -K_q^*\mathbf{x}^*(t)$$

Nash Equilibrium Solution

This study shows the Open-Loop Nash Equilibrium problem is solved by closed-loop constant feedback control policies $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$ of the following form:

$$v_q^*(\cdot) := -K_q^* \mathbf{x}^*(t)$$

where:

Nash Equilibrium Solution

This study shows the Open-Loop Nash Equilibrium problem is solved by closed-loop constant feedback control policies $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$ of the following form:

$$v_q^*(\cdot) := -K_q^* \mathbf{x}^*(t)$$

where:

- $K_q^* \in \mathbb{R}^{m_q \times n} = \mathbb{R}^{1\times 4}$ is a constant controller with respect to time defined as: $K_q^* := \frac{1}{r_q} B_q^T P_q^*$

Nash Equilibrium Solution

This study shows the Open-Loop Nash Equilibrium problem is solved by closed-loop constant feedback control policies $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$ of the following form:

$$v_q^*(\cdot) := -K_q^* \mathbf{x}^*(t)$$

where:

- $K_q^* \in \mathbb{R}^{m_q \times n} = \mathbb{R}^{1\times 4}$ is a constant controller with respect to time defined as: $K_q^* := \frac{1}{r_q} B_q^T P_q^*$
- $\mathbf{x}^*(t)$ is a game optimal state trajectory

Nash Equilibrium Solution

This study shows the Open-Loop Nash Equilibrium problem is solved by closed-loop constant feedback control policies $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$ of the following form:

$$v_q^*(\cdot) := -K_q^* \mathbf{x}^*(t)$$

where:

- $K_q^* \in \mathbb{R}^{m_q \times n} = \mathbb{R}^{1\times 4}$ is a constant controller with respect to time defined as: $K_q^* := \frac{1}{r_q} B_q^T P_q^*$
- $\mathbf{x}^*(t)$ is a game optimal state trajectory
- $P_q^* \in \mathbb{R}^{n\times n} = \mathbb{R}^{4\times 4}$ is a constituent of a positive-definite solution to a set of equations arising from the problem

Nash Equilibrium Solution

More specifically:

Nash Equilibrium Solution

More specifically:

1. $\mathbf{x}^*(t)$ is a game optimal state trajectory with regards to the Nash Equilibrium optimal control problem described, i.e. it is a solution to the model of the decomposed system when assigned with the Nash Equilibrium optimal policies $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$, so for all $t \in \mathbb{R}^{\geq 0}$ it satisfies:

Nash Equilibrium Solution

More specifically:

1. $\mathbf{x}^*(t)$ is a game optimal state trajectory with regards to the Nash Equilibrium optimal control problem described, i.e. it is a solution to the model of the decomposed system when assigned with the Nash Equilibrium optimal policies $\left(v_q^*(\cdot)\right)_{q\in\{x,\theta\}}$, so for all $t \in \mathbb{R}^{\geq 0}$ it satisfies:

$$\dot{\mathbf{x}}^*(t) = A\mathbf{x}^*(t) + \sum_{\psi\in\{x,\theta\}} B_\psi v_\psi^*(t)$$

2. The matrices $(P_q^*)_{q\in\{x,\theta\}}$ are the unique positive-definite solution[4] to the Game Continuous Algebraic Riccati Equations (GCAREs):

[4] In the study we show:

- If the set of GCAREs has a finite amount of solutions, then it is of order $O(2^N)$, with N being the number of objectives, and thus in this case $O(2^2) = O(4)$
- A solution that stabilizes the closed loop dynamics is one where each matrix P_q is positive-definite, and a unique such solution exists under certain conditions of detectability and stabilizability

2. The matrices $(P_q^*)_{q\in\{x,\theta\}}$ are the unique positive-definite solution[4] to the Game Continuous Algebraic Riccati Equations (GCAREs):

$$\forall q \in \{x,\theta\}\ ;\ P_q A_{cl} + A_{cl}^T P_q + Q_q + \frac{1}{r_q} P_q B_q B_q^T P_q = 0$$

[4] In the study we show:

- If the set of GCAREs has a finite amount of solutions, then it is of order $O(2^N)$, with N being the number of objectives, and thus in this case $O(2^2) = O(4)$
- A solution that stabilizes the closed loop dynamics is one where each matrix P_q is positive-definite, and a unique such solution exists under certain conditions of detectability and stabilizability

2. The matrices $(P_q^*)_{q\in\{x,\theta\}}$ are the unique positive-definite solution[4] to the Game Continuous Algebraic Riccati Equations (GCAREs):

$$\forall q \in \{x,\theta\}\ ;\ P_q A_{cl} + A_{cl}^T P_q + Q_q + \frac{1}{r_q} P_q B_q B_q^T P_q = 0$$

with $A_{cl} := A - \sum_{\psi\in\{x,\theta\}} \frac{1}{r_\psi} B_\psi B_\psi^T P_\psi$

[4] In the study we show:

- If the set of GCAREs has a finite amount of solutions, then it is of order $O(2^N)$, with N being the number of objectives, and thus in this case $O(2^2) = O(4)$
- A solution that stabilizes the closed loop dynamics is one where each matrix P_q is positive-definite, and a unique such solution exists under certain conditions of detectability and stabilizability

Simulation Overview

- We will now present numerical simulation results for the system to illustrate the method effectiveness

Simulation Overview

- We will now present numerical simulation results for the system to illustrate the method effectiveness
- The simulation was conducted using a Python package we developed for the purpose of implementing the general method this motivating example is a private case of

Simulation Overview

- We will now present numerical simulation results for the system to illustrate the method effectiveness
- The simulation was conducted using a Python package we developed for the purpose of implementing the general method this motivating example is a private case of
- The Package is called `PyDiffGame`[5], is fully covered in this study and can be found with extensive documentation at

https://github.com/krichelj/PyDiffGame

[5] The package has awarded the '*Starstruck*' achievement due to it being a 'repository that has many stars'

Simulation

- We will compare the results of our method with those of a regular Linear Quadratic Regulator (LQR) for the continuous infinite horizon case
- The infinite horizon LQR cost functional is of the following form:

$$J_{LQR}\left(\mathbf{u}(\cdot)\right) := \int_0^\infty \left[\tilde{\mathbf{x}}(\tau)^T Q_{LQR} \tilde{\mathbf{x}}(\tau) + \tilde{\mathbf{u}}^T(\tau) R_{LQR} \tilde{\mathbf{u}}(\tau)\right] \mathrm{d}\tau$$

where:

  - $Q_{LQR} \in \mathbb{R}^{4\times 4}$ is the LQR state weight matrix with $Q \geq 0$
  - $R_{LQR} \in \mathbb{R}^{2\times 2}$ is the LQR input weight matrix with $R > 0$
  - $\tilde{\mathbf{u}}(\cdot) := \mathbf{u}_\infty - \mathbf{u}(\cdot)$ where $\mathbf{u}_\infty$ is the input required to maintain $\mathbf{x}_\infty$, as in: $\lim_{\tau\to\infty} \mathbf{u}(\tau) = \mathbf{u}_\infty$, and $A\mathbf{x}_\infty + B\mathbf{u}_\infty = \mathbf{0}$

Simulation Game State Weights

Consider the following state weight matrices for J_x and J_θ:

$$Q_x := q \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \; ; \; Q_\theta := q \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 4 \end{bmatrix}$$

for some $q \in \mathbb{R}^+$

Simulation Game State Weights

- For $\psi \in \{x, \theta\}$:

Simulation Game State Weights

- For $\psi \in \{x, \theta\}$:
  - One can see Q_ψ affects only $\psi(t)$ and $\dot{\psi}(t)$

Simulation Game State Weights

- For $\psi \in \{x, \theta\}$:
  - One can see Q_ψ affects only $\psi(t)$ and $\dot{\psi}(t)$
  - The multiset of eigenvalues of Q_ψ, i.e. its spectrum is $\sigma(Q_\psi) = \{5q, 0, 0, 0\}$ with an algebraic multiplicity of 3 for 0

Simulation Game State Weights

- For $\psi \in \{x, \theta\}$:
  - One can see Q_ψ affects only $\psi(t)$ and $\dot{\psi}(t)$
  - The multiset of eigenvalues of Q_ψ, i.e. its spectrum is $\sigma(Q_\psi) = \{5q, 0, 0, 0\}$ with an algebraic multiplicity of 3 for 0
  - Since all these eigenvalues $\lambda \in \sigma(Q_\psi)$ satisfy $\lambda \geq 0$, Q_ψ is positive semi-definite[6]

[6] A well-known theorem elaborated on in this study

Simulation Game State Weights

- For $\psi \in \{x, \theta\}$:
  - One can see Q_ψ affects only $\psi(t)$ and $\dot{\psi}(t)$
  - The multiset of eigenvalues of Q_ψ, i.e. its spectrum is $\sigma(Q_\psi) = \{5q, 0, 0, 0\}$ with an algebraic multiplicity of 3 for 0
  - Since all these eigenvalues $\lambda \in \sigma(Q_\psi)$ satisfy $\lambda \geq 0$, Q_ψ is positive semi-definite[6]
- This setting for the weight matrices assures that each objective weights its associated state variables, while accounting more for the velocity, to reduce fluctuations

[6] A well-known theorem elaborated on in this study

LQR State Weights

- Let the LQR weight matrix then be the sum of Q_x and Q_θ

LQR State Weights

- Let the LQR weight matrix then be the sum of Q_x and Q_θ i.e.:

$$Q_{LQR} := Q_x + Q_\theta = q \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 2 & 0 & 4 & 0 \\ 0 & 2 & 0 & 4 \end{bmatrix}$$

LQR State Weights

- Let the LQR weight matrix then be the sum of Q_x and Q_θ i.e.:

$$Q_{LQR} := Q_x + Q_\theta = q \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 2 & 0 & 4 & 0 \\ 0 & 2 & 0 & 4 \end{bmatrix}$$

- The multiset of eigenvalues of Q_{LQR} is $\sigma(Q_{LQR}) = \{5q, 5q, 0, 0\}$ with an algebraic multiplicity of 2 for 0 and $5q$

LQR State Weights

- Let the LQR weight matrix then be the sum of Q_x and Q_θ i.e.:

$$Q_{LQR} := Q_x + Q_\theta = q \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 2 & 0 & 4 & 0 \\ 0 & 2 & 0 & 4 \end{bmatrix}$$

- The multiset of eigenvalues of Q_{LQR} is $\sigma(Q_{LQR}) = \{5q, 5q, 0, 0\}$ with an algebraic multiplicity of 2 for 0 and $5q$
- This setting for the LQR weight matrix accounts for attempting to capture the weighting considerations of both Q_x and Q_θ

Input Weights

- Consider the following values for the input weights:

Input Weights

- Consider the following values for the input weights:

$$r_x = r_\theta := r$$

for some $r \in \mathbb{R}^+$

Input Weights

- Consider the following values for the input weights:

$$r_x = r_\theta := r$$

for some $r \in \mathbb{R}^+$

- Correspondingly, let:

$$R_{LQR} := r \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

which is of course positive definite

Input Weights

- Consider the following values for the input weights:

$$r_x = r_\theta := r$$

  for some $r \in \mathbb{R}^+$

- Correspondingly, let:

$$R_{LQR} := r \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

  which is of course positive definite

- This setting along with that of the state weights allows us to set $r := 1$ and then just consider a value for q

Agnostic Costs

We will compare between LQR and PyDiffGame by comparing the following expressions for both instances:

$$J_{agnostic} := \int_0^\infty \left[||\tilde{\mathbf{x}}(\tau)||^2 + ||\tilde{\mathbf{u}}(\tau)||^2 \right] \mathrm{d}\tau$$

Simulation Hyperparameters Values

Let us consider the following simulation code:

```python
from itertools import product

epsilon = 10 ** (-3)
x_Ts = [10 ** p for p in [1, 2]]
theta_Ts = [pi / 2 + t for t in [pi / 2, pi / 4]]
m_cs = [10 ** p for p in [1, 2]]
m_ps = [10 ** p for p in [0, 1, 2]]
p_Ls = [10 ** p for p in [1, 2]]
qs = [10 ** p for p in [-4, -3, -2, -1, 0, 1]]
params = [x_Ts, theta_Ts, m_cs, m_ps, p_Ls, qs]
all_combos = list(product(*params))
```

There are 288 combinations, each inducing a differential game

Simulation Code

```
wins = []

for (x_T, theta_0, m_c, m_p, p_L, q) in all_combos:
    x_T = np.array([x_T, theta_0, 0, 0])
    x_0 = np.zeros_like(x_T)
    inverted_pendulum_comparison = \
        InvertedPendulumComparison(m_c=m_c, m_p=m_p, p_L=p_L, q=q,
                                   x_0=x_0, x_T=x_T, epsilon=epsilon)
    is_max_lqr = \
        inverted_pendulum_comparison(plot_state_spaces=False,
                                     run_animations=False,
                                     print_costs=True,
                                     non_linear_costs=True,
                                     agnostic_costs=True)
    wins += [int(is_max_lqr)]

wins = np.array(wins)
print(wins.sum() / len(wins) * 100)
```

Simulation Results

- We achieved success in $167/288$ games which is 57.986 percent of all the games played
- Our method is best when the number of objectives increases
- In such case weighting of the overall system becomes more difficult
- As the results show, even for this simple case where $N = 2$, in about half of the cases individual weighting incurred overall less effort