Composition of Dynamic Control Objectives Based on Differential Games

Joshua Shay Kricheli, Aviran Sadon, Shai Arogeti, Shimon Regev, and Gera Weiss

Ben Gurion University of the Negev
MED 29th Mediterranean Conference on Control and Automation

June 22-25 2021
Bari, Puglia, Italy

Introduction

- We present a controller design that uses game theory as a mechanism for composing controllers

Introduction

- We present a controller design that uses game theory as a mechanism for composing controllers
- This design allows to easily handle multiple, possibly conflicting, dynamically changing objectives

Introduction

- We present a controller design that uses game theory as a mechanism for composing controllers
- This design allows to easily handle multiple, possibly conflicting, dynamically changing objectives
- The simplicity is in the independent specification of each objective from which the final overall controller is formed

Method Overview

We propose to apply the following steps repeatedly:

1. Assign a ‘virtual input’ to each objective

Method Overview

We propose to apply the following steps repeatedly:

1. Assign a 'virtual input' to each objective

2. Define a differential game where each player controls an input and tries to achieve the corresponding objective

Method Overview

We propose to apply the following steps repeatedly:

1. Assign a 'virtual input' to each objective
2. Define a differential game where each player controls an input and tries to achieve the corresponding objective
3. Compute a Nash equilibrium and use it to construct a controller that balances between the objectives

Virtual Inputs

- We propose a method to transform a linear time-invariant (LTI) system of the form:

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

- To an equivalent system:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \sum_{i=1}^{N} B_i \mathbf{v}_i(t)$$

Virtual Inputs

- We propose a method to transform a linear time-invariant (LTI) system of the form:

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

decomposition of the input

- To an equivalent system:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \boxed{\sum_{i=1}^{N} B_i \mathbf{v}_i(t)}$$

Virtual Inputs

- We propose a method to transform a linear time-invariant (LTI) system of the form:

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

decomposition of the input

- To an equivalent system:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \boxed{\sum_{i=1}^{N} B_i \mathbf{v}_i(t)}$$

- Each $\mathbf{v}_i$ is called a virtual input designed s.t. each input affects mostly one of the objectives

- For nonlinear systems, we use on-the-fly linearization around each state using linear parameter varying (LPV) models

Cost Functions

- For a system of the form:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \sum_{i=1}^{N} B_i \mathbf{v}_i(t)$$

Cost Functions

- For a system of the form:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \sum_{i=1}^{N} B_i \mathbf{v}_i(t)$$

- Each virtual input is associated with objective defined as an independent cost function of the form:

$$J_i = \int_0^\infty \left[\mathbf{x}(t)^T Q_i \mathbf{x}(t) + \sum_{j=1}^{N} \mathbf{v}_j(t)^T R_{ij} \mathbf{v}_j(t) \right] dt$$

where each Q_i specifies the cost of deviations from the zero state and R_{ij} specifies control costs

Movie

Nash Equilibrium

- We consider the feedback control policies $(\mathbf{v}_1^*, \mathbf{v}_2^*, \ldots, \mathbf{v}_N^*)$ where $\mathbf{v}_i^*$ maps the state to the i'th virtual input
- Such a tuple constitutes a Nash Equilibrium iff for all $1 \leq i \leq N$ and any policy $\mathbf{v}_i$ we have:

$$J_i(\mathbf{v}_1^*, \ldots, \mathbf{v}_i^*, \ldots, \mathbf{v}_N^*) \leq J_i(\mathbf{v}_1^*, \ldots, \mathbf{v}_i, \ldots, \mathbf{v}_N^*)$$

Nash Equilibrium

- We consider the feedback control policies $(\mathbf{v}_1^*, \mathbf{v}_2^*, \ldots, \mathbf{v}_N^*)$ where $\mathbf{v}_i^*$ maps the state to the i'th virtual input
- Such a tuple constitutes a Nash Equilibrium iff for all $1 \leq i \leq N$ and any policy $\mathbf{v}_i$ we have:

$$J_i(\mathbf{v}_1^*, \ldots, \boxed{\mathbf{v}_i^*}, \ldots, \mathbf{v}_N^*) \leq J_i(\mathbf{v}_1^*, \ldots, \boxed{\mathbf{v}_i}, \ldots, \mathbf{v}_N^*)$$

replacing $\mathbf{v}_i^*$ by $\mathbf{v}_i$ does not improve J_i

Nash Equilibrium

- We consider the feedback control policies $(\mathbf{v}_1^*, \mathbf{v}_2^*, \ldots, \mathbf{v}_N^*)$ where $\mathbf{v}_i^*$ maps the state to the i'th virtual input

- Such a tuple constitutes a Nash Equilibrium iff for all $1 \leq i \leq N$ and any policy $\mathbf{v}_i$ we have:

$$J_i(\mathbf{v}_1^*, \ldots, \boxed{\mathbf{v}_i^*}, \ldots, \mathbf{v}_N^*) \leq J_i(\mathbf{v}_1^*, \ldots, \boxed{\mathbf{v}_i}, \ldots, \mathbf{v}_N^*)$$

replacing $\mathbf{v}_i^*$ by $\mathbf{v}_i$ does not improve J_i

- We propose to construct the actual input using the virtual inputs satisfying the Nash equilibrium

- This generates a control strategy that forms a dynamic balance between the objectives

Calculating the Nash Equilibrium

- For the Nash Equilibrium, we consider linear feedback controllers of the form:[1]

$$\mathbf{v}_i^*(\mathbf{x}(t)) = -K_i\mathbf{x}(t) = -R_{ii}^{-1}B_i^T P_i\mathbf{x}(t)$$

[1] Vassilis L. Syrmos Frank L. Lewis Draguna L. Vrabie. "Optimal Control". In: chap. 10, pp. 438–460.

Calculating the Nash Equilibrium

- For the Nash Equilibrium, we consider linear feedback controllers of the form:[1]

$$\mathbf{v}_i^*(\mathbf{x}(t)) = -K_i\mathbf{x}(t) = -R_{ii}^{-1}B_i^T P_i \mathbf{x}(t)$$

- where $\{P_i\}_{i=1}^N$ is a solution of the Ricatti equations:

$$0 = P_i A_c + A_c^T P_i + Q_i + \sum_{j=1}^{N} P_j B_j R_{jj}^{-T} R_{ij} R_{jj}^{-1} B_j^T P_j$$

- and where:

$$A_c = A - \sum_{i=1}^{N} B_i R_{ii}^{-1} B_i^T P_i$$

[1] Frank L. Lewis, "Optimal Control".

Quadrotor Example Use Case

- For demonstration, we implemented a quadrotor controller

[2] Amir Shapiro Hanoch Efraim Shai Arogeti and Gera Weiss. "Vision Based Output Feedback Control of Micro Aerial Vehicles in Indoor Environments". In: (2017).

Quadrotor Example Use Case

- For demonstration, we implemented a quadrotor controller
- A quadrotor is a well studied non-linear system with six degrees of freedom

[2] Hanoch Efraim and Weiss, "Vision Based Output Feedback Control of Micro Aerial Vehicles in Indoor Environments".

Quadrotor Example Use Case

- For demonstration, we implemented a quadrotor controller
- A quadrotor is a well studied non-linear system with six degrees of freedom
- We use a simulation of a quadrotor in an indoor environment based on a work previously done by our team[2]

[2] Hanoch Efraim and Weiss, "Vision Based Output Feedback Control of Micro Aerial Vehicles in Indoor Environments".

Quadrotor Example Use Case

- For demonstration, we implemented a quadrotor controller
- A quadrotor is a well studied non-linear system with six degrees of freedom
- We use a simulation of a quadrotor in an indoor environment based on a work previously done by our team[2]
- We implement two levels of control:
  1. Orientation control based on inertial sensors
  2. Positional control based on coordinates of the lines of a corridor

[2] Hanoch Efraim and Weiss, "Vision Based Output Feedback Control of Micro Aerial Vehicles in Indoor Environments".

Quadrotor Example Use Case

- For demonstration, we implemented a quadrotor controller
- A quadrotor is a well studied non-linear system with six degrees of freedom
- We use a simulation of a quadrotor in an indoor environment based on a work previously done by our team[2]
- We implement two levels of control:
  1. Orientation control based on inertial sensors
  2. Positional control based on coordinates of the lines of a corridor
- We simulate the corresponding image that the quadrotor is presumably witnessing according to the given current state

[2] Hanoch Efraim and Weiss, "Vision Based Output Feedback Control of Micro Aerial Vehicles in Indoor Environments".

We define two control objectives:

Quadrotor Objectives

We define two control objectives:

1. The first objective is stabilizing the quadrotor at the center of the corridor, facing forward

Quadrotor Objectives

We define two control objectives:

1. The first objective is stabilizing the quadrotor at the center of the corridor, facing forward
2. The second one is a dynamic objective on the forward velocity, where the objective changes with the distance of the quadrotor to the wall

Quadrotor Objectives

We define two control objectives:

1. The first objective is stabilizing the quadrotor at the center of the corridor, facing forward

2. The second one is a dynamic objective on the forward velocity, where the objective changes with the distance of the quadrotor to the wall
   - If the quadrotor is closer to the center than the wall, then the objective is to reach a (positive) constant velocity

Quadrotor Objectives

We define two control objectives:

1. The first objective is stabilizing the quadrotor at the center of the corridor, facing forward

2. The second one is a dynamic objective on the forward velocity, where the objective changes with the distance of the quadrotor to the wall
    - If the quadrotor is closer to the center than the wall, then the objective is to reach a (positive) constant velocity
    - Otherwise, the objective is to decrease the velocity to zero

There might be a conflict between these two objectives, thus finding a balance between them is beneficial

Quadrotor Control

- Let us consider the state vector of the quadrotor:

$$\mathbf{x} = \begin{bmatrix} \phi \ \dot{\phi} \ \theta \ \dot{\theta} \ \psi \ \dot{\psi} \ z \ \dot{z} \ x \ \dot{x} \ y \ \dot{y} \end{bmatrix}^T$$

- The quadrotor is controlled by applying desired torques using its motors
- The torques are generated by controlling the angular velocity of each motor
- Let us define the following input vector for the thrust and three torques along the axes:

$$\mathbf{u} = \begin{bmatrix} T \ \tau_x \ \tau_y \ \tau_z \end{bmatrix}^T$$

Quadrotor Model

We consider the following rigid-body non-linear model[3] (assuming small angles, symmetry and diagonal inertia matrix):

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{\phi} \\ \frac{I_{yy}-I_{zz}}{I_{xx}}\dot{\theta}\dot{\psi} + \frac{J_r}{I_{xx}}\dot{\theta}\Omega_r + \frac{l}{I_{xx}}\tau_x \\ \dot{\theta} \\ \frac{I_{zz}-I_{xx}}{I_{yy}}\dot{\phi}\dot{\psi} - \frac{J_r}{I_{yy}}\dot{\phi}\Omega_r + \frac{l}{I_{yy}}\tau_y \\ \dot{\psi} \\ \frac{I_{xx}-I_{yy}}{I_{zz}}\dot{\theta}\dot{\phi} + \frac{1}{I_{zz}}\tau_z \\ \dot{z} \\ g - \frac{1}{m} cos\phi \cos\theta \; T \\ \dot{x} \\ \frac{1}{m}\left(\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi\right) T \\ \dot{y} \\ \frac{1}{m}\left(\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi\right) T \end{bmatrix}$$

[3] Samir Bouabdallah. "Design and Control of Quadrotors with Application to Autonomous Flying". In: (2007).

Simulation Tactics I

- We show the performance of the suggested methodology in the context of the quadrotor hierarchical control strategy

Simulation Tactics I

- We show the performance of the suggested methodology in the context of the quadrotor hierarchical control strategy
- The method uses an image-based visual servoing to control micro aerial vehicles (MAVs) in indoor environments

Simulation Tactics I

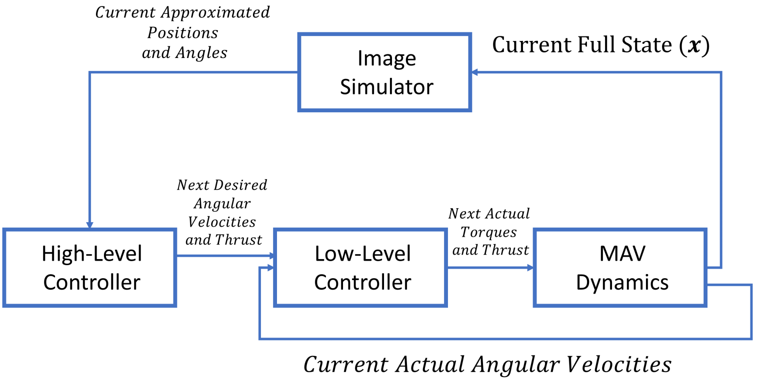
- We show the performance of the suggested methodology in the context of the quadrotor hierarchical control strategy
- The method uses an image-based visual servoing to control micro aerial vehicles (MAVs) in indoor environments
- We assume the state of the system is obtained by a linearization of the model proposed earlier

Simulation Tactics I

- We show the performance of the suggested methodology in the context of the quadrotor hierarchical control strategy
- The method uses an image-based visual servoing to control micro aerial vehicles (MAVs) in indoor environments
- We assume the state of the system is obtained by a linearization of the model proposed earlier
- Then we simulate the corresponding image that the quadrotor is presumably witnessing according to that state

Simulation Tactics II

The following is a high-level representation of the simulated system:



Simulation Tactics III

1. The state $\mathbf{x}$ is obtained by solving $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$

Simulation Tactics III

1. The state $\mathbf{x}$ is obtained by solving $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$

2. This state is fed into the image simulator that produces the current corresponding approximated positions and angles

Simulation Tactics III

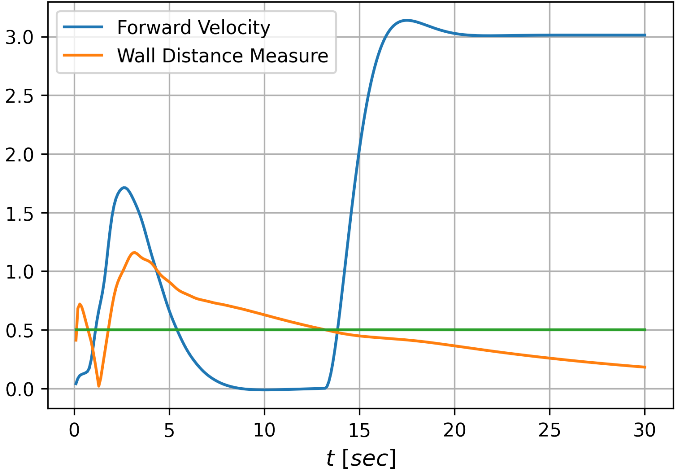
1. The state $\mathbf{x}$ is obtained by solving $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$
2. This state is fed into the image simulator that produces the current corresponding approximated positions and angles
3. Then the high-level controller produces the desired thrust and angular rates

Simulation Tactics III

1. The state $\mathbf{x}$ is obtained by solving $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$
2. This state is fed into the image simulator that produces the current corresponding approximated positions and angles
3. Then the high-level controller produces the desired thrust and angular rates
4. Finally, the low-level controller produces the actual thrust and torques, $\mathbf{u}$, to be applied to the quadrotor

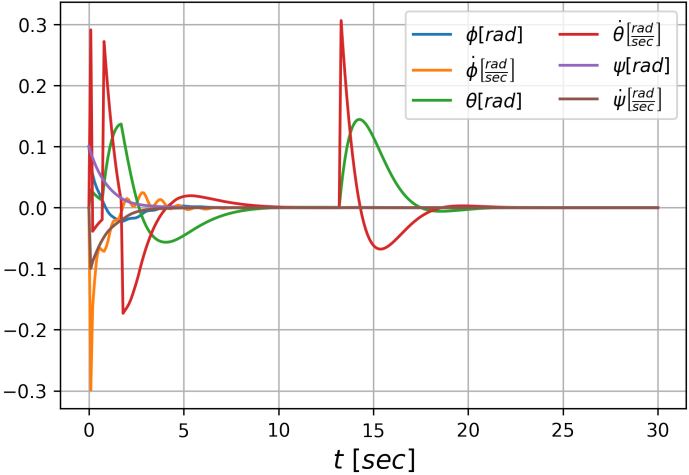
Results I - Objectives Dynamics

The following figure displays the simulation results. Once the wall measure reaches 0.5 - the velocity starts to increase from 0 to 3:



Results II - Angular State Variables

The following figure displays the angles and angular velocities



Conclusions

- We saw our method provides an easy way of finding a plausible balance in between possibly conflicting objectives
- Our simulation provides a quick stabilization of the state variables, even after changes in the interaction
- Handling non-linear systems with multiple objectives can be made quite easy by this method

Python Package

- We designed a Python package that encapsulates the method described
- The package provides a quick way to define objectives and generates the corresponding dynamic differential games
- Then the algorithm solves these games on-the-fly and generates the state variables
- The package is called `PyDiffGame` and is detailed extensively in the article

PyDiffGame

```
git clone https://github.com/krichelj/PyDiffGame.git
```

Thank you for listening!

Acknowledgments

This research was supported in part by:

- The Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Initiative at Ben-Gurion University of the Negev
- The Marcus Endowment Fund at Ben-Gurion University of the Negev
- The Israeli Smart Transportation Research Center (ISTRC)