Modelling and optimization of cooperative multi-robot-systems

Christian Reinl

Simulation and Systems Optimization Group
Technische Universität Darmstadt

Koblenz, January/18/2007
Overview

Introduction

Optimizing a mixed-integer linear program (MILP)

Modelling the cooperative system

Results and potentialities for joint projects
Introduction

Investigated problems

Hybrid and hierarchical automata

Optimizing a mixed-integer linear program (MILP)

Modelling the cooperative system

Results and potentialities for joint projects
Example problems

- *RoboCup* scenario
  - Robots *together* try to score a goal
  - Problem: optimal trajectories and task assignment
Example problems

- **RoboCup scenario**
  - Robots *together* try to score a goal
  - Problem: optimal trajectories and task assignment

- **Unmanned Air Vehicles (UAVs)**
  - UAVs *cooperate* in monitoring an area
  - Fire monitoring, traffic surveillance, ...
  - *COMETS* project
    (www.comets-uavs.org)

source: http://www.comets-uavs.org
Robotic games (Challenges)

- attacker-defender-models
- specific **dynamical** abilities need adjusted tactic planning
- **discrete** roles: go_to_ball, dribble, kick, ...
- **reaction** on **perturbations**
  (e.g. caused by the opponent’s actions)
- many **uncertainties**
Benchmark problems

- coordinated path-planning for unmanned vehicles
  - collision avoidance
  - constraints for communication
  - intention: minimize time or energy

- cooperative task allocation and trajectory planning in the RoboCup
  - two players
  - one (simple) defender
  - intention: improve the attackers chances for a considered time horizon
Describing the system with a hybrid automaton

- initial-conditions
- jump-conditions
- event-conditions (e.g. "kick")

- fixed number of possible transitions at unknown points in time $t_i$
- dynamics in the states is described by differential equations of motion
  \[ \dot{x} = f(x, u, t) \]

Complete physical description of the multi-vehicle system from a global view!
Hybrid optimal control (Markus Glocker)

- modelled with **hybrid automata**
- transformation into a finite dimensional problem with **direct collocation**
- solved with SNOPT
- **Branch-and-bound-techniques**

**Circular tours:**

**RoboCup scenario:**
Clocked hierarchical automaton

- Transitions only occur on a time discretization
- Hierarchy allows description of behavior on different levels

Heuristic rules are implemented:
- To control the behavior,
- To guarantee coordinated actions and
- To control the motions

The sum of these automata describes the system from an internal view!
Introduction

Optimizing a mixed-integer linear program (MILP)
Definition and characteristics of MILP
Techniques to formulate the linear program

Modelling the cooperative system

Results and potentialities for joint projects
Formal definition of MILP

- **mixed integer linear program (MILP)**

\[
\min_{x,z} \quad f_1^T x + f_2^T z \\
\text{subject to} \quad G_1 x + G_2 z \leq b \\
x \in B_1 \subseteq \mathbb{R}^{n_c} \\
z \in B_2 \subseteq \mathbb{N}^{n_d}
\]

- (quite the) simplest form of a mixed-integer optimization problem with constraints

- several approaches to transform a complex problem in this formalism need to be investigated

- modelling techniques are decisive for the quality of the entire procedure in mixed integer programming
Mixed-integer linear programs

▶ applicable for predictive control in uncertain environments
Mixed-integer linear programs

- applicable for predictive control in **uncertain environments**
- **fast** available initial guesses for optimization
Mixed-integer linear programs

- applicable for predictive control in **uncertain environments**
- **fast** available initial guesses for optimization
- solution of linear problems can be **computed directly** without iterating the objective function and derivations
Mixed-integer linear programs

- applicable for predictive control in **uncertain environments**
- **fast** available initial guesses for optimization
- solution of linear problems can be **computed directly** without iterating the objective function and derivations
- even for simple problems a huge number of variables is needed
Mixed-integer linear programs

- applicable for predictive control in **uncertain environments**
- **fast** available initial guesses for optimization
- solution of linear problems can be **computed directly** without iterating the objective function and derivations
- even for simple problems a huge number of variables is needed
- **structure** is more important than the number of variables
Mixed-integer linear programs

- applicable for predictive control in **uncertain environments**
- **fast** available initial guesses for optimization
- solution of linear problems can be **computed directly** without iterating the objective function and derivations
- even for simple problems a huge number of variables is needed
- **structure** is more important than the number of variables
- best available solver **CPLEX**
Handling non-convex environments

- considered problems are (mostly) **non-convex**
Handling non-convex environments

- considered problems are (mostly) non-convex

- use linear approximations to subdivide them in convex subproblems
Handling non-convex environments

- considered problems are (mostly) **non-convex**

- use **linear approximations** to subdivide them in convex subproblems

- **logical constraints** describe the (approximated) region
Introduction

Optimizing a mixed-integer linear program (MILP)

Modelling the cooperative system
  Formulating an objective function
  Introducing the automata
  Modeling the vehicles’ and ball’s dynamic
  Connecting and adding constraints to the model

Results and potentialities for joint projects
The way to optimization with clocked hierarchical automata

1. formulate an **objective function** to be minimized
2. translate the **automata’s structure** into mathematical equations and inequalities using **binary variables**.
3. formulate all the agents’ different dynamics with **finite difference equations**
4. **select** conditions from the automata
5. connect the **switched physical conditions and invariants** to the automata’s structure
6. add **additional constraints**
Formulating an objective function

For a time-horizon \([t_0, t_f]\)
- the position of the (player on the) ball,
- his distance to the defender and
- the position of the supporter and
- the control efforts and
- the incidence of some states (e.g. \texttt{ball\_in\_goal}) should be optimized.
Translation of the automaton

- introducing a **binary variable** $b_s(k)$ for each state $s$ and each timestep $k$
Translation of the automaton

- introducing a **binary variable** $b_s(k)$ for each state $s$ and each timestep $k$

\[
b_s(k + 1) \leq \sum_{p \in S^*} b_p(k)
\]

- transitions:
Translation of the automaton

- introducing a **binary variable** $b_s(k)$ for each state $s$ and each timestep $k$

  $$b_s(k + 1) \leq \sum_{p \in S^*} b_p(k)$$

- transitions:

  $$b_s(k) = \sum_{p \in S} b_p(k)$$

- hierarchies:
Modeling the vehicles’ and ball’s dynamic

- introducing a **sampling time** $t_s$

\[ \dot{x}(t) = f(x(t), u(t)) \quad \Rightarrow \quad x_{k+1} = x_k + t_s(A_k x_k + B_k u_k) \]

- decoupling and linearization must be done carefully
  - additional constraints may be needed
  - **physical characteristics** must be considered
Modeling the vehicles’ and ball’s dynamic

- introducing a **sampling time** $t_s$

\[
\dot{x}(t) = f(x(t), u(t)) \quad \Rightarrow \quad x_{k+1} = x_k + t_s(A_k x_k + B_k u_k)
\]

- decoupling and linearization must be done carefully
  - additional constraints may be needed
  - **physical characteristics** must be considered

- performance of the optimization depend primarily on the binary variables, so that we used

\[
x(k + 1) = x(k) + t_s v(k); \quad v(k + 1) = x(k) + t_s u(k)
\]

ball: \[x_b(k + 1) = x_b(k) + t_s v_b(k)\]

for our first investigations
Selection of conditions for modelling the system

- existing automata include many **heuristics** to control the robots and to guarantee coordinated actions
Selection of conditions for modelling the system

- existing automata include many heuristics to control the robots and to guarantee coordinated actions
- every state is characterized by invariants and conditions
Selection of conditions for modelling the system

- existing automata include many **heuristics** to control the robots and to guarantee coordinated actions
- every state is characterized by **invariants** and **conditions**
- Optimization needs some **latitude**
Selection of conditions for modelling the system

- existing automata include many **heuristics** to control the robots and to guarantee coordinated actions
- every state is characterized by **invariants** and **conditions**
- Optimization needs some **latitude**

Conditions must be **dropped** and **replaced according to the attributes** to be optimized!
Connecting conditions with the automaton’s structure

- [sate s is active \((b_s = 1)\)] \(\Rightarrow\) [condition fulfilled] \(\Leftrightarrow\) \(inv_s = 1\)

\[\forall k, \forall s : b_s(k) \leq inv_s(k)\]
Connecting conditions with the automaton’s structure

- [state $s$ is active ($b_s = 1$)] $\Rightarrow$ [condition fulfilled] $\Leftrightarrow$ $inv_s = 1$

\[ \forall k, \forall s : b_s(k) \leq inv_s(k) \]

- Logical expressions can be translated into linear constraints; e.g. with the widespread \textbf{'big-M'-method}

example:
\[
\begin{align*}
\text{[either } (g_1 \geq b_1) \text{ or } (g_2 \geq b_2) \text{]} & \Leftrightarrow \\
\begin{bmatrix}
g_1 & \leq & b_1 + M\delta_1 \\
g_2 & \leq & b_2 + M\delta_2 \\
\delta_1 + \delta_2 & \leq & 1
\end{bmatrix}
\end{align*}
\]

$\delta_i \in \{0, 1\}$, $M > 0$, $M > \max\{g_1 - b_1\}$, $M > \max\{g_2 - b_2\}$
Connecting conditions with the automaton’s structure

- [sate s is active ($b_s = 1$)] $\Rightarrow$ [condition fulfilled] $\Leftrightarrow$ $inv_s = 1$

$$\forall k, \forall s : b_s(k) \leq inv_s(k)$$

- Logical expressions can be translated into linear constraints; e.g. with the widespread ’big-M’-method

example:

\[
\begin{align*}
\text{either } (g_1 \geq b_1) \text{ or } (g_2 \geq b_2) & \Leftrightarrow \\
& \begin{bmatrix}
  g_1 & \leq & b_1 + M\delta_1 \\
  g_2 & \leq & b_2 + M\delta_2 \\
  \delta_1 + \delta_2 & \leq & 1
\end{bmatrix} \\
\delta_i & \in \{0, 1\}, \ M > 0, \ M > \max\{g_1 - b_1\}, \ M > \max\{g_2 - b_2\}
\end{align*}
\]

- not unique

\[
L_1 \vee (L_2 \land L_3) \Leftrightarrow \delta_1 + \delta_2 \geq 1, \ \delta_1 + \delta_3 \geq 1 \Leftrightarrow 2\delta_1 + \delta_2 + \delta_3 \geq 2
\]

not equivalent for $\delta_i \in \mathbb{R}$!
Adding Constraints for collision Avoidance

- **static obstacles:**

\[
(x - x_{\text{Obst}})^2 + (y - y_{\text{Obst}})^2 > r_{\text{Obst}}^2
\]

\[\text{⇓}\]

\[
\bigvee_{i=1}^{6} \left[ k_{i,1} (x - x_{\text{Obst}}) + k_{i,2} (y - y_{\text{Obst}}) > r \right]
\]

\[
k_{i,1} = \sin \frac{i}{3} \pi \quad k_{i,2} = \cos \frac{i}{3} \pi
\]
Adding Constraints for collision Avoidance

> **static obstacles:**

\[
(x - x_{Obst})^2 + (y - y_{Obst})^2 > r_{Obst}^2
\]

\[\downarrow\]

\[
\bigwedge_{i=1}^{6} \left[k_{i,1}(x - x_{Obst}) + k_{i,2}(y - y_{Obst}) > r\right]
\]

\[
k_{i,1} = \sin \frac{i}{3} \pi \quad k_{i,2} = \cos \frac{i}{3} \pi
\]

> **avoiding collisions with moving objects:**

\[
\bigwedge_{i=1}^{4} \left[k_{i,1}(x_1 - x_2) + k_{i,2}(y_1 - y_2) > d\right]
\]

\[
k_{i,1} = \sin \frac{i}{2} \pi \quad k_{i,2} = \cos \frac{i}{2} \pi
\]
Adding constraints

- The introduction of

\[
[\text{Only one player can dribble the ball}] \iff [b_{dribble,Rob1}(k) + b_{dribble,Rob2}(k) \leq 1]
\]

reduces the computing time to a third!
Adding constraints

- The introduction of
  \[\text{Only one player can dribble the ball} \iff b_{\text{dribble,Rob}1}(k) + b_{\text{dribble,Rob}2}(k) \leq 1\]
  reduces the computing time to a third!

- Dropped conditions in former steps must be replaced by **weaker (state-specific) constraints** on the control and state variables.
Adding constraints

- The introduction of

\[ \text{[Only one player can dribble the ball]} \iff b_{\text{dribble},Rob1}(k) + b_{\text{dribble},Rob2}(k) \leq 1 \]

reduces the computing time to a third!

- Dropped conditions in former steps must be replaced by weaker (state-specific) constraints on the control and state variables.

- Introducing a defender and constraints for the rolling ball
Adding constraints

- The introduction of
  \[ \text{[Only one player can dribble the ball]} \iff \text{[} \ b_{dribble, Rob_1}(k) + b_{dribble, Rob_2}(k) \leq 1 \text{]} \]
  reduces the computing time to a third!

- Dropped conditions in former steps must be replaced by
  **weaker (state-specific) constraints** on the control and state variables.

- Introducing a **defender** and constraints for the **rolling ball**

- **Known bounds** and **heuristics** can be introduced to accelerate the optimization.
Adding constraints

- The introduction of
  \[
  \text{[Only one player can dribble the ball]} \iff [b_{dribble, Rob_1}(k) + b_{dribble, Rob_2}(k) \leq 1]
  \]
  reduces the computing time to a third!
- Dropped conditions in former steps must be replaced by \textbf{weaker (state-specific) constraints} on the control and state variables.
- Introducing a \textbf{defender} and constraints for the \textbf{rolling ball}
- \textbf{Known bounds} and \textbf{heuristics} can be introduced to accelerate the optimization.
- The solution of the relaxed optimization problem should be as near as possible to the optimum of the real problem!
Introduction

Optimizing a mixed-integer linear program (MILP)

Modelling the cooperative system

Results and potentialities for joint projects
  Results
  Future work
(Adjusted) hierarchical automaton

1. Considering the original automaton
(Adjusted) hierarchical automaton

1. Considering the original automaton
2. Dropping the outer state 

"soccer"
(Adjusted) hierarchical automaton

1. Considering the original automaton
2. Dropping the outer state “soccer”
3. Dropping the flow conditions
(Adjusted) hierarchical automaton

1. Considering the original automaton
2. Dropping the outer state “soccer”
3. Dropping the flow conditions
4. ”stand”  ”grab_ball”
(Adjusted) hierarchical automaton

1. Considering the original automaton
2. Dropping the outer state “soccer”
3. Dropping the flow conditions
4. ”stand” ↨ ”grab_ball”
5. Introducing states for the ball
(Adjusted) hierarchical automaton

1. Considering the original automaton
2. Dropping the outer state "soccer"
3. Dropping the flow conditions
4. "stand" $\leadsto$ "grab_ball"
5. Introducing states for the ball
6. Connecting the states of the ball to the automaton
The matrix characterizing the linear constraints

structure:

used automata:
The matrix characterizing the linear constraints structure:

dimensions:

- # conditions (simple automaton)
- # conditions (hierarchical automaton)
- # variables (hierarchical automaton)
- # variables (simple automaton)
Visualization

view from above:  switches in the automaton:
Computing time

- 100 randomly created instances
- comparison between the simple-automata modelling an the hierarchical automata.
- optimization without any initial guess
- the computing time mainly depends on the quality of the defender and on the initial positions of ball, robots and defender
Computing time

- 100 randomly created instances
- comparison between the simple-automata modelling and the hierarchical automata.
- optimization without any initial guess
- the computing time mainly depends on the quality of the defender and on the initial positions of ball, robots and defender

![Graph showing mean computing time vs. time steps](image)
Computing time

- 100 randomly created instances
- comparison between the simple-automata modelling and the hierarchical automata.
- optimization without any initial guess
- the computing time mainly depends on the quality of the defender and on the initial positions of ball, robots and defender
Computing time

- 100 randomly created instances
- comparison between the simple-automata modelling an the hierarchical automata.
- optimization without any initial guess
- the computing time mainly depends on the quality of the defender and on the initial positions of ball, robots and defender
Future and joint work

▶ testing different MIQP-models
Future and joint work

- testing different MIQP-models
- embed the optimization in a model predictive control framework
Future and joint work

- testing different MIQP-models
- embed the optimization in a model predictive control framework
- creation of additional constraints by using bounds and heuristics resulting from model checking
Future and joint work

- testing different **MIQP-models**
- embed the optimization in a model predictive control framework
- creation of additional constraints by using bounds and heuristics resulting from model checking
- best **worst-case** optimization
Future and joint work

- testing different MIQP-models
- embed the optimization in a modell predictive control framework
- creation of additional constraints by using bounds and heuristics resulting from model checking
- best worst-case optimization
- consider synchronisation in the context of optimization
Thank you for your attention!