Co-Design of Autonomous Systems: From Hardware Selection to Control Synthesis

European Control Conference 2021

G. Zardini, A. Censi, and E. Frazzoli

Institute for Dynamic Systems and Control (IDSC)
ETH Zürich

gzardini@ethz.ch - http://gioele.science
The pain of engineering complex systems

An autonomous robot = actuation, sensing, computation, control, energetics, software, behavior, coordination, localization, interaction, mapping, learning, liability, communication, planning, social acceptance, control, regulations

So many components (hardware, software, ...), so many choices to make! Nobody can understand the whole thing!

We forget why we made some choices, and we are afraid to make changes...
These “computer” thingies are not helping us that much for design...

“My dear, it’s simple: you lack a proper theory of co-design!”

anthropomorphization of 21st century engineering malaise
Co-design of autonomous systems: from hardware selection to control synthesis

An autonomous robot = hardware + software + behavior + coordination

sensing + actuation + localization + planning + social acceptance

computation + control + interaction + mapping + acceptance

perception + energetics + learning + liability + regulations

sensing + coordination + computation + control + perception + energetics + communication + learning + liability + regulations

**Takeways** of this talk:

- Using co-design, it is easy to *embed* the synthesis of *controllers* into the co-design problem of the whole *autonomous robot*

- Very *intuitive* modeling approach (no “acrobatics” needed)

- Rich *modeling capabilities*: analytic models, catalogues, simulations

- Compositionality and modularity allow *interdisciplinary collaboration*

- Co-design produces *actionable information* for designers to *reason* about their problems
Across fields, design or synthesis problems are defined with 3 spaces:

- **implementation space**: the options we can choose from;
- **functionality space**: what we need to provide/achieve;
- **requirements/costs space**: the resources we need to have available;
An abstract view of design problems

- Across fields, design or synthesis problems are defined with 3 spaces:
  - **implementation space**: the options we can choose from;
  - **functionality space**: what we need to provide/achieve;
  - **requirements/costs space**: the resources we need to have available;

**Partially ordered sets**

\[
\langle F, \leq_F \rangle \quad \text{to maximize} \quad \text{choices} \quad \langle R, \leq_R \rangle \quad \text{to minimize}
\]
Partial orders allow to model various trade-offs

**Definition.** A poset is a tuple \( \langle P, \leq_p \rangle \), where \( P \) is a set and \( \leq_p \) is a partial order, defined as a reflexive, transitive, and antisymmetric relation.

- All **totally ordered sets** are particular cases of **partially ordered sets**:
  \[ \langle \mathbb{R}_{\geq 0}, \leq \rangle \quad \langle \mathbb{N}, \leq \rangle \]
Partial orders allow to model various trade-offs

Definition. A poset is a tuple \( \langle P, \leq_p \rangle \), where \( P \) is a set and \( \leq_p \) is a partial order, defined as a reflexive, transitive, and antisymmetric relation.

- All **totally ordered sets** are particular cases of **partially ordered sets**:
  \[
  \langle \mathbb{R}_{\geq 0}, \leq \rangle \quad \langle \mathbb{N}, \leq \rangle
  \]

- In this work, among others, we consider the poset of **positive semi-definite matrices**

  Definition. A symmetric matrix \( M \in \mathbb{R}^{n \times n} \) is **positive semi-definite** if \( x^T M x \geq 0 \) for all non-zero \( x \in \mathbb{R}^n \). We call the set of all such matrices \( \mathcal{P}^n \).

- We can define a **partial order** as \( A \leq B \iff (B - A) \in \mathcal{P}^n \), \( A, B \in \mathcal{P}^n \)

- Symmetric matrices have **real** eigenvalues

- Can be interpreted as **axes lengths** of **ellipsoids**

- Order is given by **ellipsoids inclusion**

\[
C = \begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}
\]

\[
A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}
\]
**Definition** (Design problem with implementation). A *design problem with implementation* (DPI) is a tuple

$$< F, R, I, prov, req >,$$

where:
- $F$ is a poset, called *functionality space*;
- $R$ is a poset, called *requirements space*;
- $I$ is a set, called *implementation space*;
- the map $prov : I \rightarrow F$ maps an implementation to the functionality it provides;
- the map $req : I \rightarrow R$ maps an implementation to the resources it requires.
We use this graphical notation:

- **functionality:** *green continuous wires* on the left
- **requirements:** *dashed red wires* on the right.

### Graphical Notation Components

- **Battery**
  - Capacity [J]
  - Max current [A]
  - Mass [g]
  - Cost [USD]

### Implementations

- Amazon search for batteries:
  - AA Batteries
  - AAA Batteries
  - 9V Batteries
  - D Batteries
  - C Batteries
Engineering is constructive

- For the purpose of design, we need to know how something is done, not just that it is possible to do something: engineering is constructive.

- We need to know what are the implementation(s), if any, that relate functionality and costs.

- For the algorithmic solution of co-design problem, it is useful to consider a direct feasibility relation from functionality to costs.

- Monotone map: Lower functionalities does not require more resources, higher resources do not provide less functionalities.
The composition of any two DPs returns a DP (closure)

Very practical tool to decompose large problems into subproblems
Two basic design queries are:

- **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
- **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

**Given the functionality** to be provided, what are the **minimal resources** required?

**Given the resources** that are available, what is the **maximal functionality** that can be provided?
Two basic design queries are:

- **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
- **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

The two problems are **dual**

From the solutions, one can retrieve the **implementations** (design choices)
Two basic design queries are:

- **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
- **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

**Given the functionality** to be provided, what are the **minimal resources** required?

We are looking for:

- A map from functionality to upper sets of feasible resources: \( h : F \rightarrow \mathcal{U}R \)
- A map from functionality to antichains of minimal resources: \( h : F \rightarrow \mathcal{A}R \)
This is the semantics of \textbf{FixFunMinReq} as a family of optimization problems.

- **chosen by user:** \( \overline{f} \) \( \overline{r} \) to minimize

- **variables:** \( r_k \in \langle R_k, \leq_{R_k} \rangle \) \( f_k \in \langle F_k, \leq_{F_k} \rangle \)

- **constraints:** for each node: \( f_k \) \( d_k : F_k \rightarrow R_k \) \( r_k \)

- **for each edge:** \( f_j \) \( r_i \) \( r_i \leq f_j \)

- **objective:** \( \text{Min} \overline{r} \)

\( d_k(f_k^*, r_k) = \top \)
Solving DP queries

- Suppose we are given the function $h_k : F_k \rightarrow \mathcal{R}_k$ for all nodes in the co-design graph.

- Can we find the map $h : F \rightarrow \mathcal{R}$ for the entire diagram?

- **Recursive approach:** We just need to work out the composition formulas for all operations we have defined.

- The set of **minimal** feasible resources can be obtained as the **least fixed point** of a monotone function in the space of anti-chains.
Use case: Co-design of an autonomous drone

Actuation
- speed [m/s]
- lift [N]
- control effort

Feature Extraction
- observe at δ [Hz]
- precision
- tracking error

Mission Planning
- number of missions
- mission time [s]

Vision Sensor
- power [W]
- resolution [px/sterad]
- acquisition frequency [Hz]
- impl. feature at δ [Hz]

Implement Feature
- power [W]
- mass [g]
- cost [CHF]

Implement Control
- power [W]
- mass [g]
- cost [CHF]
- control effort

Battery
- energy stored [J]
- mass [g]
- cost [CHF]

Computing
- total computation [op/s]
- total power [W]

LQG Control
- system noise W
- control effort
- precision
- tracking error

Cost and Mass
- total cost [CHF]
- total mass [g]
Infinite-horizon LQG control in one slide

- Let’s consider the **continuous time, stochastic** dynamics
  
  \[
  \begin{align*}
  dx_t &= Ax_t dt + Bu_t dt + Edw_t \\
  dy_t &= Cx_t dt + Gdv_t,
  \end{align*}
  \]

  where \( A, B, C, D, E, G \) are of adequate dimensions, \( v_t \) and \( w_t \) Brownian processes, and \( W = EE^*, V = GG^* \) noise covariances.

- We consider the classic **infinite-horizon LQG problem**, finding a control law minimizing the cost
  
  \[
  J = \lim_{T \to \infty} \frac{1}{T} E \left\{ \int_0^T ((x_t^T Q x_t) + (u_t^T R u_t)) dt \right\}
  \]

  where \( Q \) is a positive semi-definite matrix and \( R \) is a positive definite matrix.

- **Well-known lemma**: the optimal control law for the problem is
  
  \[
  u_t^* = -Kx_t = -R^{-1}B^*\hat{S}x_t
  \]

  where \( \hat{x}_t \) is the unbiased minimum-variance estimate of \( x_t \), and \( \hat{S} \) solves the Riccati equation \( SA + A^*S - SBR^{-1}B^*S + Q = 0 \).

- We can obtain the **optimal cost**
  
  \[
  J^* = \text{Tr}(\hat{S}\hat{\Sigma}C^*V^{-1}C\hat{\Sigma} + \hat{\Sigma}Q) = \text{Tr}(\hat{\Sigma}\hat{S}BR^{-1}B^*\hat{S} + \hat{\Sigma}W),
  \]

  where \( \hat{\Sigma} \) solves the Riccati equation \( A\Sigma + \Sigma A^* - \Sigma C^*V^{-1}C\Sigma + W = 0 \).
LQG control as a co-design problem

Let’s consider the **performance** metrics

\[
P_{\text{track}} = \lim_{t \to \infty} \mathbb{E}\{x_t^\top Q x_t\} \quad P_{\text{effort}} = \lim_{t \to \infty} \mathbb{E}\{u_t^\top R u_t\}
\]
LQG control as a co-design problem

- Let's consider the **performance** metrics

  \[
  P_{\text{track}} = \lim_{t \to \infty} \mathbb{E}\{x_t^\top Q x_t\} \quad P_{\text{effort}} = \lim_{t \to \infty} \mathbb{E}\{u_t^\top R u_t\}
  \]

- **Theorem**: We can write the LQG problem as a design problem of the form:

![Diagram showing LQG control and associated metrics](image-url)
LQG control as a co-design problem

- Let’s consider the **performance** metrics

\[
P_{\text{track}} = \lim_{t \to \infty} \mathbb{E}\{x_t^\top Q x_t\} \quad P_{\text{effort}} = \lim_{t \to \infty} \mathbb{E}\{u_t^\top R u_t\}
\]

- **Theorem**: We can write the LQG problem as a design problem of the form:

- **Proof procedure** in four steps:

  - Show that one can *rewrite* the performance metrics as

    \[
    \lim_{t \to \infty} \mathbb{E}\{x_t^\top Q_0 x_t\} = \text{Tr}(Q_0 (\Sigma + F)) \quad \lim_{t \to \infty} \mathbb{E}\{u_t^\top R_0 u_t\} = \text{Tr}(SB^*R^{-1}R_0 R^{-1}BSF),
    \]

    where \( F \) solves the Lyapunov equation \((A - BK) F + F (A - BK)^* + LVL^* = 0 \), and \( L = \Sigma C^*V^{-1} \).

  - Show **monotonicity** of tracking error and control effort performances with respect to \( Q \) and \( R \).

  - Show \( \langle V, W \rangle \leq \langle V', W' \rangle \Rightarrow \Sigma(V, W) \leq \Sigma(V', W') \).

  - Show **monotonicity** of tracking and effort with respect to \( V \) and \( W \).
Theorem: For the LQG problem with observation and computation delays we can write the design problem:

Proof sketch:
- Substitution principle: *If in the case a certain nuisance is “lower”, the controller could simulate a “higher” nuisance*
LQG control with delays and the discrete version

- **Theorem:** For the LQG problem with observation and computation delays we can write the design problem:

- **Proof sketch:**
  - Substitution principle: *If in the case a certain nuisance is “lower”, the controller could simulate a “higher” nuisance*

- Analogous statements can be proven for the discrete-time case

- **Theorem:** One can write a design problem of the form:
Use case: Co-design of an autonomous drone

- **Actuation**
  - lift [N]
  - control effort

- **Feature Extraction**
  - precision
  - tracking error

- **Mission Planning**
  - mission time [s]

- **Vision Sensor**
  - resolution [px/sterad]

- **Implement Feature**
  - impl. feature at δ [Hz]

- **Implement Control**
  - impl. control at δ [Hz]

- **LQG Control**
  - power [W]
  - mass [g]

- **Battery**
  - energy stored [J]

- **Computing**
  - total computation [op/s]

- **Total**
  - total power [W]
  - total cost [CHF]
  - total mass [g]
Use case: Co-design of an autonomous drone

- **Actuation**
  - Speed [m/s]
  - Lift [N]

- **Feature Extraction**
  - Precision
  - Tracking error

- **Mission Planning**
  - Number of missions
  - Mission time [s]

- **Vision Sensor**
  - Observation at δ [Hz]
  - Resolution [px/sterad]

- **Implement Feature**
  - Impl. feature at δ [Hz]

- **Implement Control**
  - Impl. control at δ [Hz]

- **Battery**
  - Power [W]
  - Mass [g]
  - Energy stored [J]

- **Computing**
  - Total computation [op/s]
  - Cost [CHF]
  - Mass [g]

- **LQG Control**
  - Control effort
  - System noise [W]

- **Catalogue of algorithms**
  - Catalogue of computers
  - Catalogue of batteries

- **Catalogue of sensors**
Co-design is very intuitive!

- The theory comes with a **formal language** and a **solver (MCDP)**

- Very intuitive to use:

```plaintext
mcdp {
  provides computation [op/s]
  requires cost [CHF]
  requires mass [g]
  requires power [W]
}

choose(
  SedanS: (load Car_SedanS),
  SedanM: (load Car_SedanM),
  SedanL: (load Car_SedanL),
  SUVS: (load Car_SuvS),
  SUV: (load Car_SuvM),
  Minivan: (load Car_Minivan),
  Shuttle: (load Car_Shuttle),
  Hybrid: (load Car_Hybrid),
  BEV: (load Car_BEV)
)
```

Choose query type:
- **Fixed the functionality, minimize the resources.**
- **Fixed the resources, maximize the functionality.**
- Given an implementation, evaluate functionality/resources. [UI not implemented]
- Given min functionality and max resources, determine if there is a feasible implementation. [UI not implemented]
- Given min functionality and max resources, find a feasible implementation. [UI not implemented]
- "Solve for X": find the minimal component that makes the co-design problem feasible. [UI not implemented]
Fix functionalities, 
Minimize resources

Details of autonomy, 
both hardware and software

Monotonicity
Takeaways

- Using co-design, it is easy to embed the synthesis of controllers into the co-design problem of the whole autonomous robot.
- We have shown how to embed (variations of) LQG control problems into the co-design problem of an autonomous robot.

- Very intuitive modeling approach (no acrobatics like common in optimization theory).
  *The interpreter allows one to easily model problems of interest*.

- Rich modeling capabilities:
  - *Simulation*: Algorithms’ performances
  - *Catalogues*: Sensors, vehicles, computers, algorithms, ...
  - *Analytical*: LQG closed-form solutions, discomfort models, ...

- Compositionality and modularity allow interdisciplinarity
  *We did all of it, but technically this could have been possible with different teams*.

- Co-design comes with a formal language and an optimizer
  *After easily modeling the problem, you can directly solve queries of your choice*.

- Co-design produces actionable information for designers to reason about their problems.
  *We have shown actionable information for municipalities, as well as for AV developers.*
Outlook and references

- Showcase **compositionality** by including the co-design of the **robot** in the co-design of **fleets of robots** (fleet control)

- Generalize this modeling approach to other **control structures** (nonlinear, receding horizon, ...)

- Exploit the framework to synthesize **energy** and **computation-aware** control strategies

References:

- This paper: *Co-Design of Autonomous Systems: From Hardware Selection to Control Synthesis* ([https://bit.ly/3ixXa5g](https://bit.ly/3ixXa5g))

- Related work:

- This is a **new** topic, we are making an effort in **evangelization**:
  - We are writing a **book**, teaching **classes**, both at ETH and internationally, and organizing **workshops**

  *[https://applied-compositional-thinking.engineering](https://applied-compositional-thinking.engineering)*  
  *[https://idsc.ethz.ch/research-frazzoli/workshops/compositional-robotics](https://idsc.ethz.ch/research-frazzoli/workshops/compositional-robotics)*  
  *[http://gioele.science](http://gioele.science)*