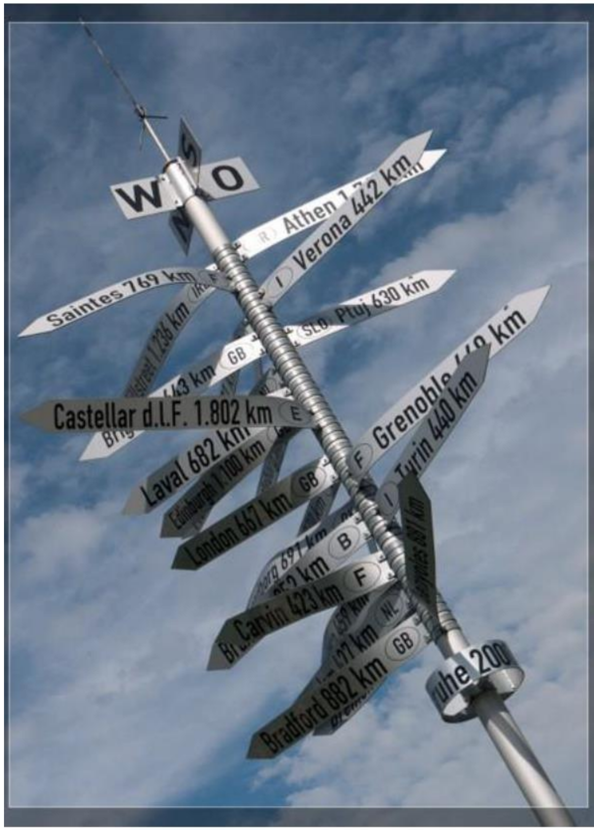
Agenda

ORGANIC COMPUTING

- Motivation
- Autonomy and self-organisation
- Quantification of self-organisation
- The survival cycle of an organic system
- Robustness
- Autonomy
- Conclusion and further readings



Motivation

Organic Computing systems…

- … are inspired by nature.
- … mimic architectural and behavioural characteristics, such as self-organisation, self-adaptation or decentralised control.
- … avoid a single-point-of-failure to achieve desirable properties, such as self-healing, self-protection, and self-optimisation.

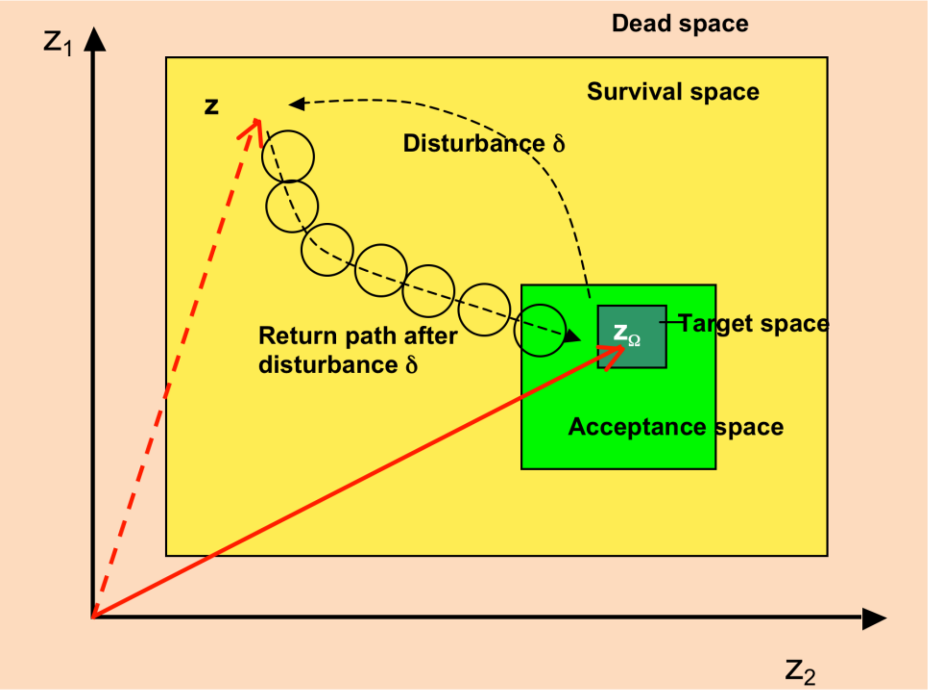
Goal:

- The ultimate goal is to use these concepts to make systems resistant against external or internal disturbances.
- OC systems do not per se achieve a higher performance than conventional systems but they return faster to a certain corridor of acceptable performance in the presence of disturbances.

We call this property: robustness!

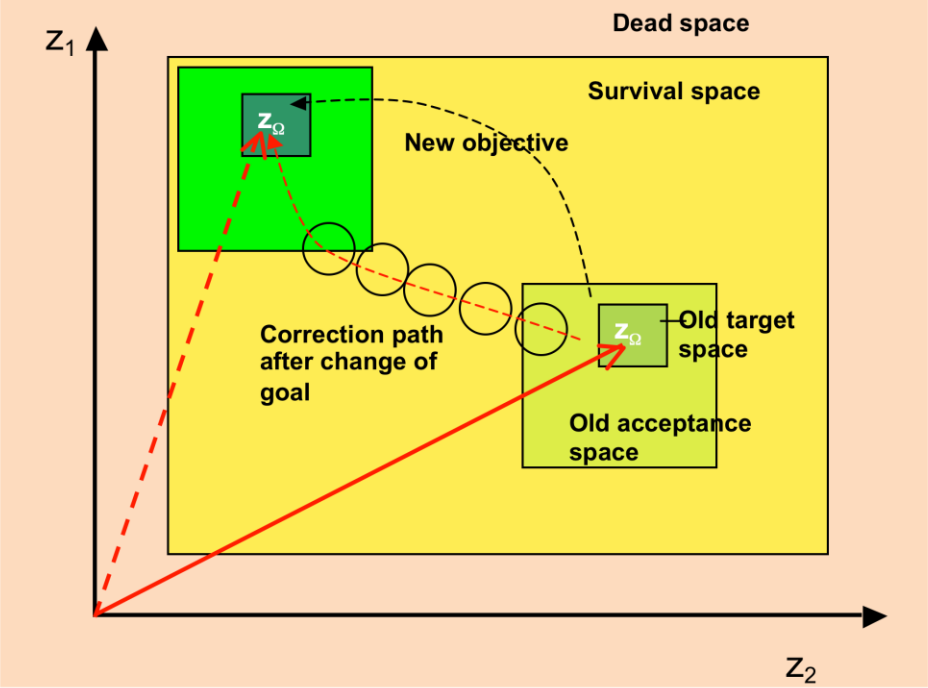
We distinguish two possible reasons for state changes of *S*:

1. The system state $\vec{z}(t)$ changes due to an internal change of the system (e.g. broken component) or a change of the environment (disturbance δ). If the system remains to be acceptable, this corresponds to the common understanding of a **robust** system.



Robustness vs. flexibility (2)

2. The state $\vec{z}(t)$ stays where it is but the evaluation and acceptance criteria change. This moves target and acceptance space – they have a new position within the *n*-dimensional state space. We call a system, which is able to cope with such changes in its behavioural specification, a **flexible** system.



Classes of robustness

- We call a system more robust if it has a large number of states that do not lead to a reduced performance or to undesired behaviour.
- Definition: Let D be a non-empty set of disturbances δ:

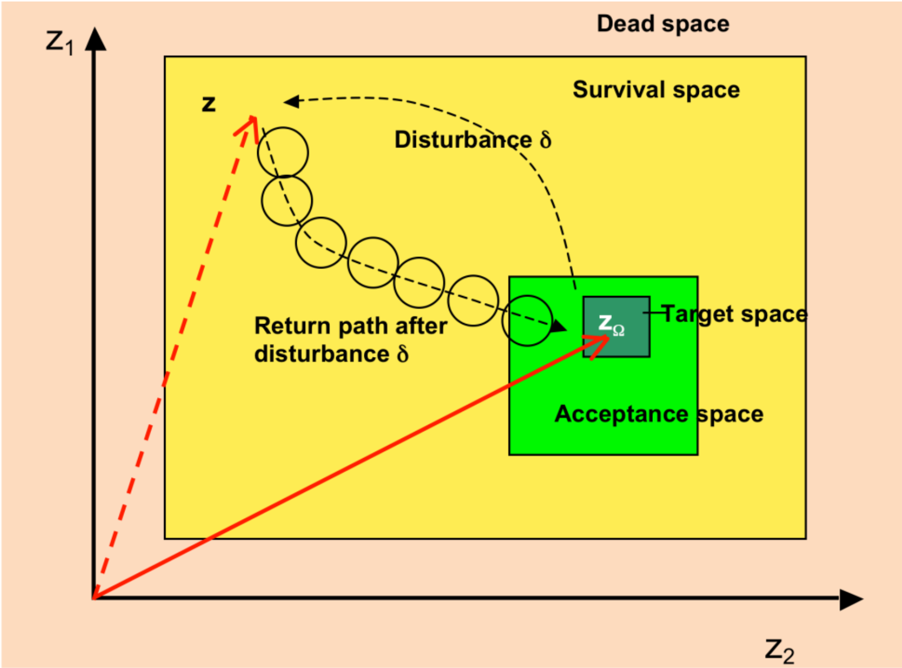
1. A system *S* is called **strongly robust** with respect to D, iff all the disturbances in $\delta \in D$ map the target space into itself.

2. A system *S* is called **robust** with respect to D, iff all the disturbances in $\delta \in D$ map the target space into the acceptance space.

3. A system *S* is called **weakly robust** with respect to D, iff all the disturbances in $\delta \in D$ map the target space into the survival space and the internal control mechanism CM is able to lead S back to at least an acceptable state.

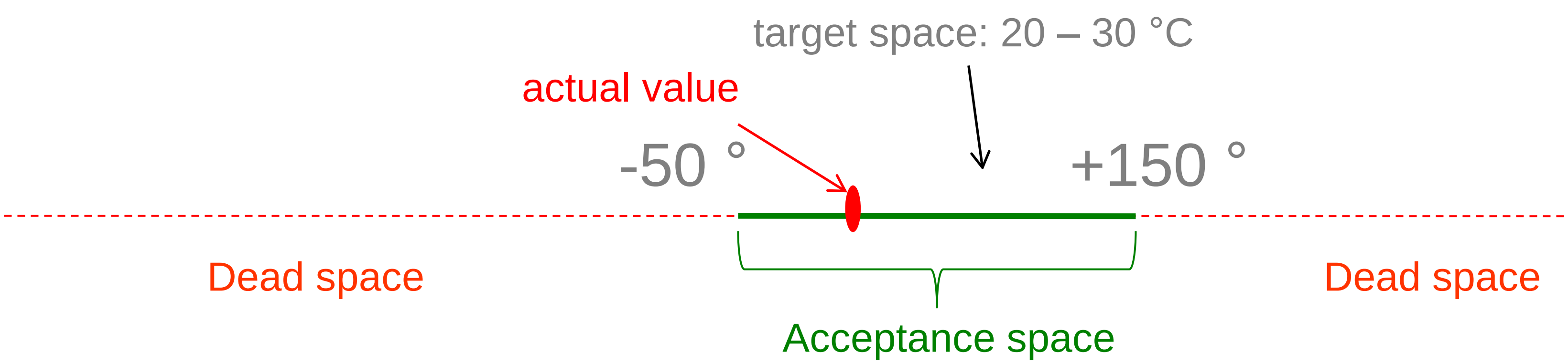
Classes of robustness (2)

- The (degree of) robustness of a system increases with the size of the set of disturbances *D* the system can handle.
- I.e. the system fulfils the requirements named by the three different robustness classes named before.



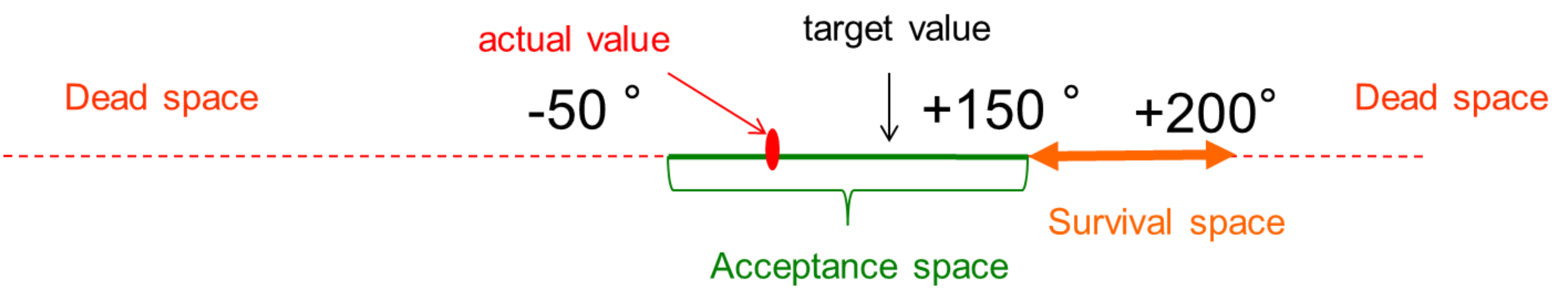
Example

- An integrated circuit IC_1 with automotive specification functions correctly in an environment with temperatures from -50°C to +150°C. It works best in the temperature range from 20 to 30 °C.
  – Within -50°C to +150 °C (acceptance space), there is no control action necessary.
  – It is strongly robust with respect to a change of environment conditions if it stays within 20 – 30 °C.
  – It is robust with respect to a change of environment conditions if it stays within -50 – +150 °C.



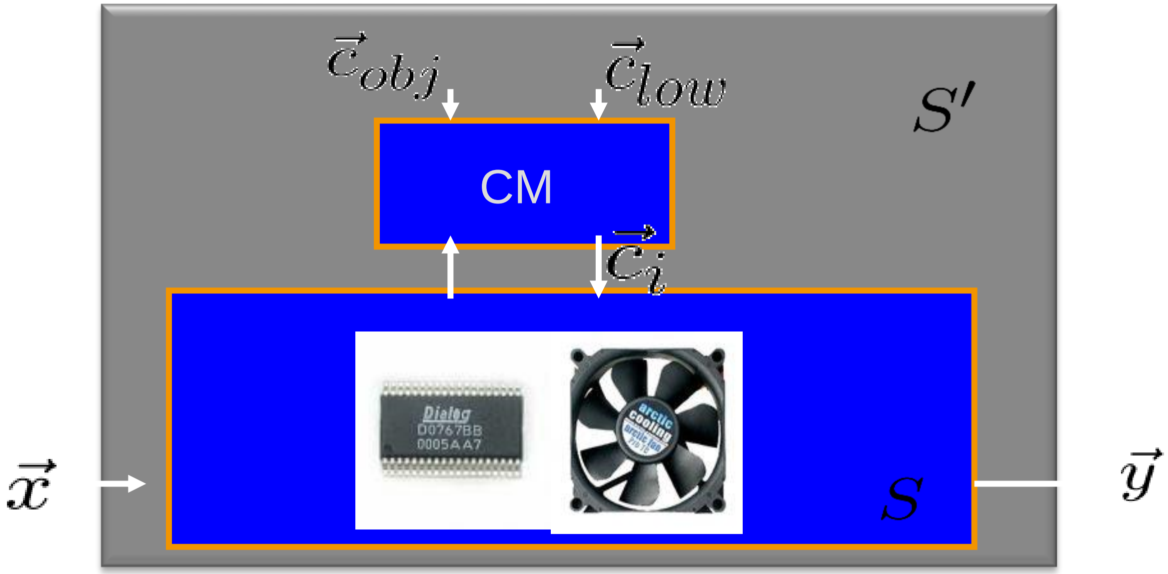
Example (2)

- An integrated circuit IC_2 functions correctly in an environment with temperatures from -50 °C to +150 °C.
  - Within this temperature range, there is no control action necessary (acceptance space).
- Now we add a cooling fan to IC_2 and a temperature sensor. The fan is able to cool the IC_2 if the temperature goes beyond +150 °C (but does not exceed 200 °C).



Example (3)

- The fan and the CM (which controls it) turn IC_2 into a weakly robust system with respect to a temperature range of -50 °C to + 200 °C.
  - S = (IC2 + sensor + fan) is an **adaptable** system.
  - S' = (IC2 + sensor + fan + CM) is an **adaptive** system.



Observation

- OC systems “under attack” show a characteristic behaviour (*attack* is a certain instance of the broader class of disturbances).
- I.e. a fast drop in utility after a disturbance (or an intentional attack).
- Followed by a somewhat slower recovery to the original performance (provided that there are suitable OC mechanisms).

Quantification of robustness

- Goal: Estimate the benefit of OC control.
- Approach: compare different CM designs in terms of their ability to provide resilience to external disturbances.
- Focus: use the area of the characteristic utility degradation over time.
  → Area captures (i) depth of the utility drop, (ii) duration of the recovery.
- Note: degradation of zero corresponds to an ideally robust system.

Quantification requirements

Generalised approach to quantify robustness:

a) works on externally measurable values,

b) does not need additional information sources (e.g. transactional data-bases),

c) distinguishes between system-inherent (or passive) and system-added (or active) robustness (to allow for an estimation of the effectiveness of the particular mechanism) and

d) provides a measure that allows for a comparison of different systems for the same problem instance.

We derive such a measure step-wise in the following…

Passive and active robustness

- System in undisturbed state shows a certain target performance.
- Rate a system by a utility measure **u**.
- System reacts to disturbance by deviating from its acceptable utility u_{acc} by Δu.
- Passively robust systems react to the disturbance by a deflection Δu = Δx.
→ Tower example: wind pressure -> Δx in horizontal distance.
- Active robustness mechanisms (e.g. an organic control mechanism) counteract the deviation and guide the system back to the undisturbed state with Δu = 0.
- OC systems typically have both!
→ We'll refer to passive/active as phase 1 and 2.

Robustness types (2)

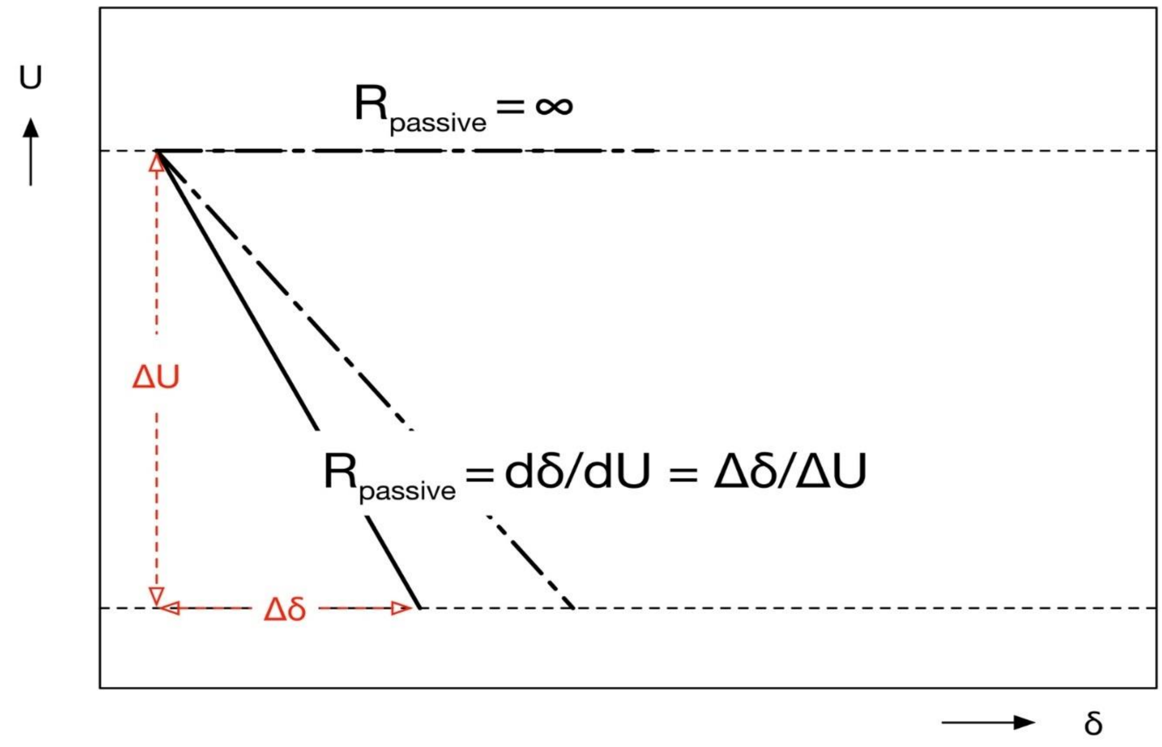
Quantification of robustness

- Phases I and II together constitute the deviation phase.
- Might be difficult to discriminate and may overlap.
  → Active recovery mechanism starts working already at t_δ.
- For quantification, take into account:
  - the strength of the disturbance δ
  - the drop of the system utility from the acceptable utility u (i.e. Δu)
  - the duration of the deviation (the recovery time t_{rec}).

Passive Robustness

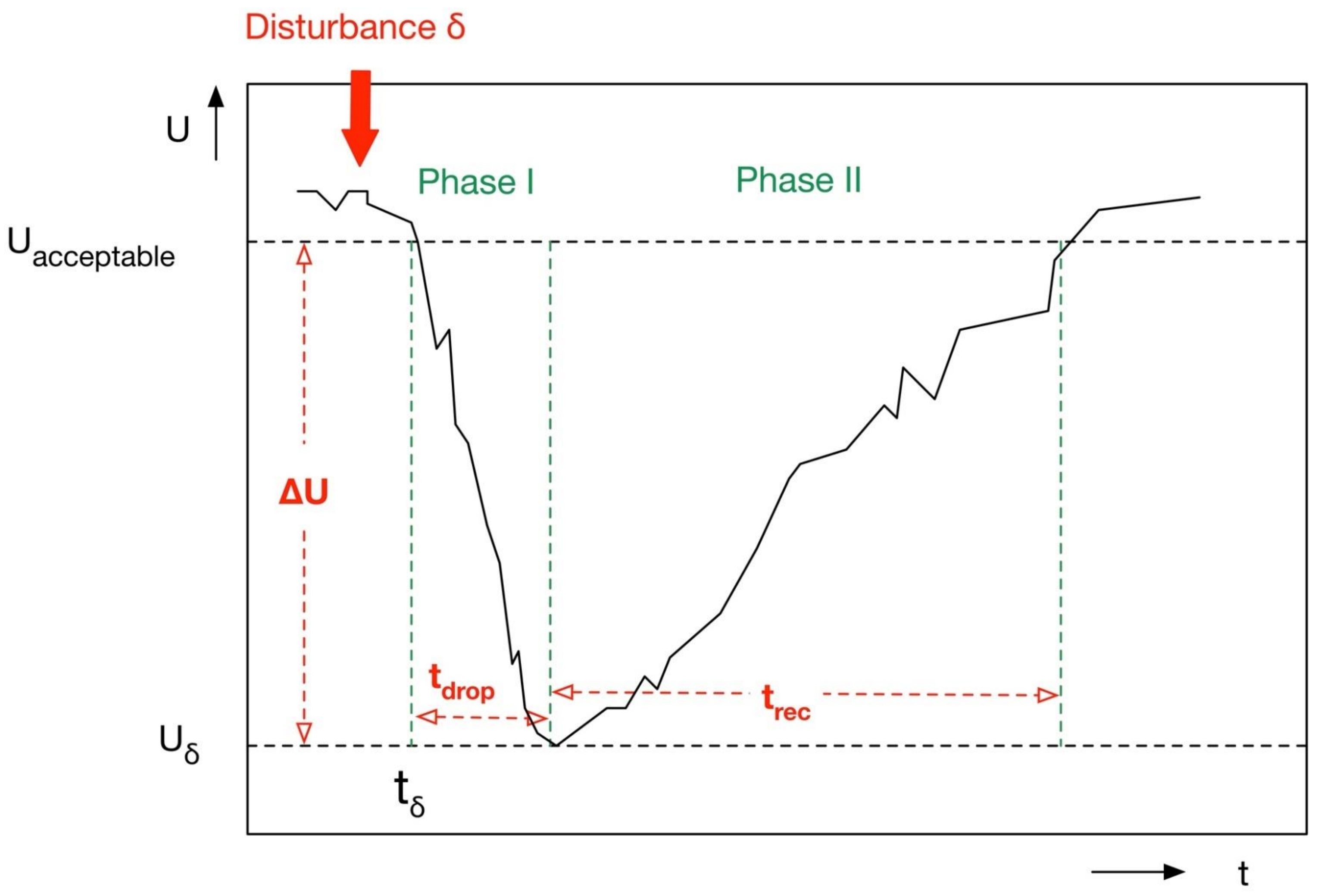
- Determined by the sensitivity of u against δ.
- Measure of the built-in stability of the system without an active CM.
- *Structural sensitivity* σ defined as the utility change caused by a disturbance δ (or the gradient of u(δ)): $\sigma = du(\delta)/d\delta$
- If δ has no effect on a system ($\Delta u = 0$) its sensitivity is $\sigma = 0$.
  - Example 1: A very stable concrete tower, which does not move ($\Delta u = 0$) under a storm of strength δ, is structurally infinitely stable, its sensitivity is $\sigma = 0$.
  - Example 2: A communication link with an error correcting code, which corrects errors up to 3 bits, is structurally insensitive to a disturbance of strength $\delta = 1$ bit.

Passive robustness (2)



- The sensitivity σ determines the utility drop ΔU caused by a disturbance δ.

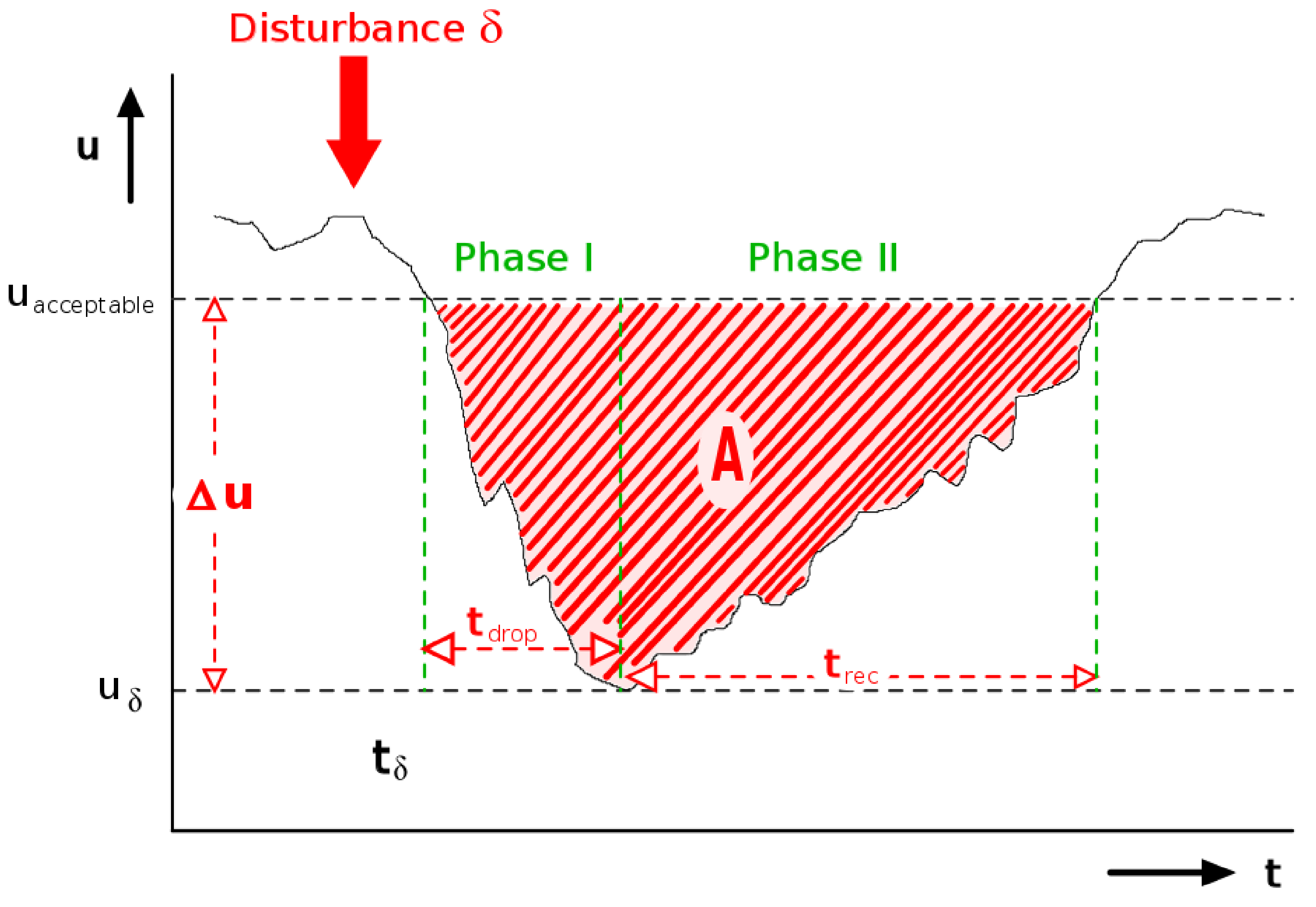
Robustness process



Active robustness

- Determined through the (averaged) recovery speed of the system.
- $s_{active} = du/dt$ or
- $s_{active} = \Delta u/t_{rec}$ (in case of a full recovery).
- With $t_{rec} = \Delta u/s_{active}$ and $\Delta u = \delta \cdot \sigma$ we get
- $t_{rec} = \delta \cdot \sigma \,/\, s_{active}$
- s_{active} is a property of the control mechanism (CM).
- Without a CM, the system stays at $u_{disturbed}$ at least as long as the disturbance remains.
- The recovery time t_{rec} depends on the initial utility drop Δu determined by the system's sensitivity against the disturbance as well as the active recovery speed.

Quantification of robustness (2)



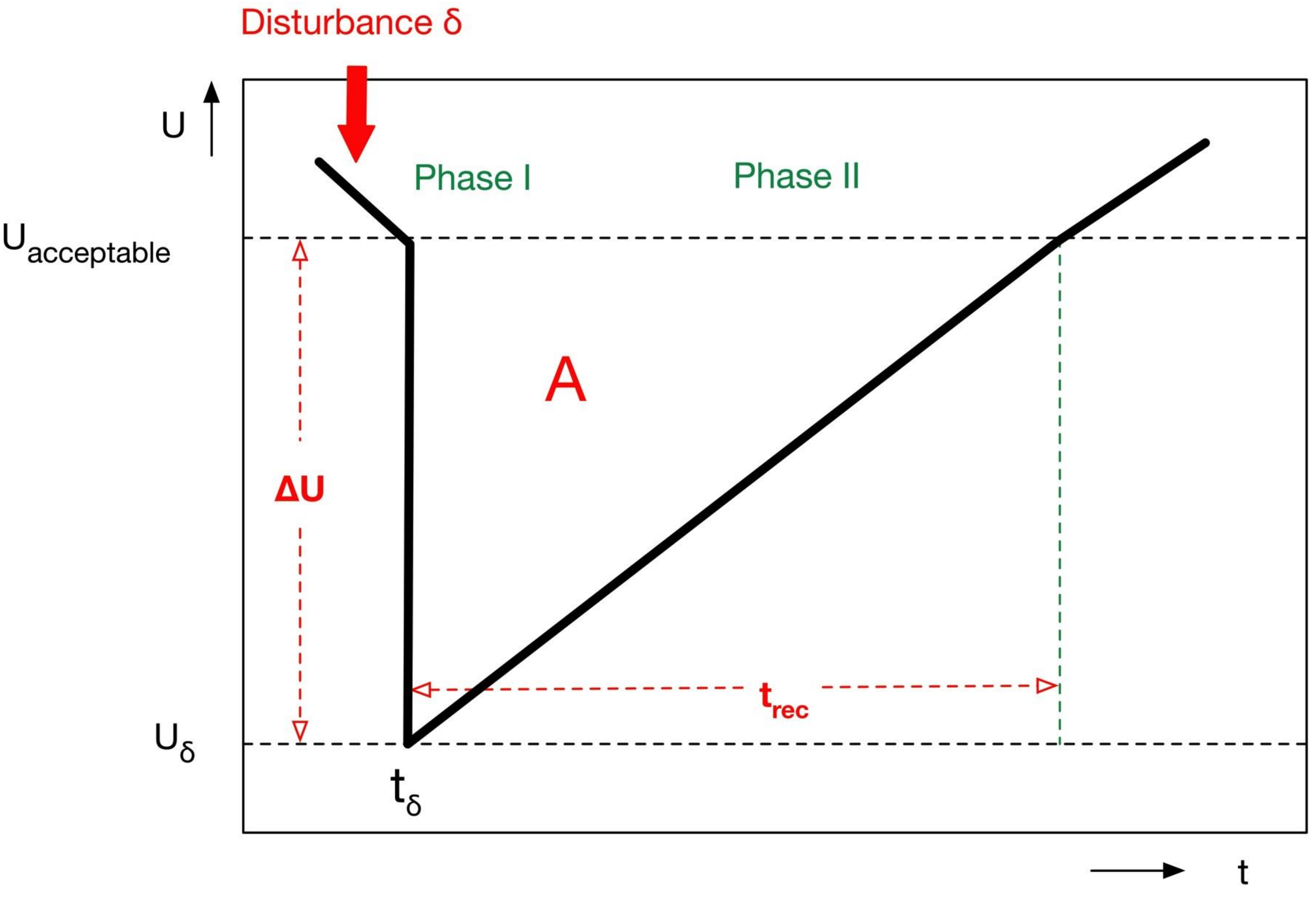
Effective utility degradation

- Robustness of a system under a given disturbance of strength δ is characterised by the triple (δ, Δu, t_{rec}) or (δ, σ, s_{active}).
- To gauge the total effect of the disturbance on the system: use the area A of the utility deviation from u_{acc} until full recovery to u_{acc} is reached.
- The area A between the accepted utility u_{acc} and the actual utility curve is defined as the utility degradation D_U (holds exactly only if $u_\delta = 0$, otherwise a correction is necessary).

$$D_u = \mathbf{\Delta} u \cdot (t_{drop} + t_{rec}) - \int_{t_\delta}^{t_\delta + t_{drop} + t_{rec}} u(t)\, dt$$

- To achieve a minimal degradation, we have to minimise t_{rec} and Δu.

Approximation of utility degradation

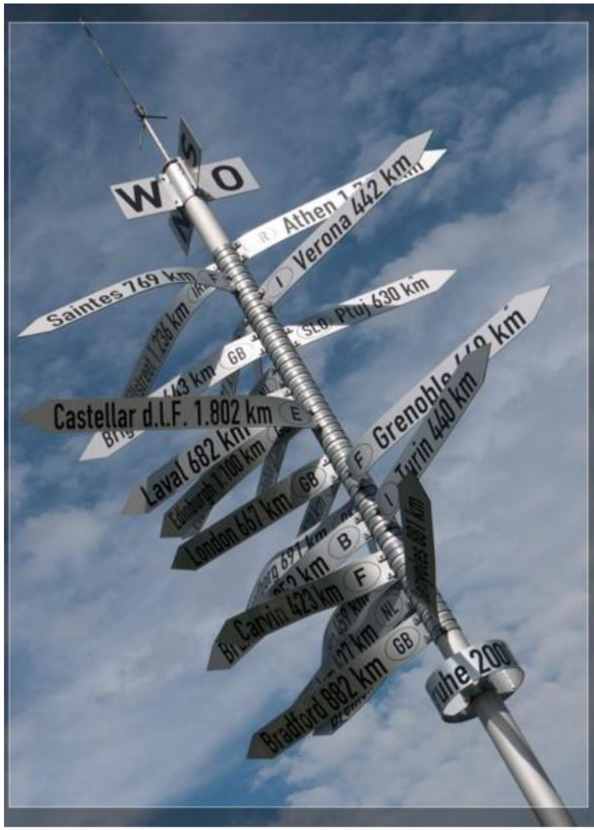


Approximation

- Goal: Simplification and better estimation
- Assume that the drop occurs very fast, hence we can set $t_{drop} = 0$.
- Assume for simplification a linear utility increase, which renders the utility degradation triangular.
- Then: $D_u \approx \Delta u \cdot t_{rec} / 2 = ½ \, \delta \, \sigma \cdot \delta \, \sigma / s_{active}$
- Effective utility degradation is $D_u = ½ \, \delta^2 \cdot \sigma^2 / s_{active}$
- Observation: Decrease of the sensitivity σ decreases D_u more effectively than an increase of the recovery speed s_{active}.
- Reason: σ influences Δu as well as t_{rec}.
- Formula also shows trade-off is possible between σ and s_{active} depending on the cost incurred for passive (σ) and active (s_{active}) robustness measures.

Agenda

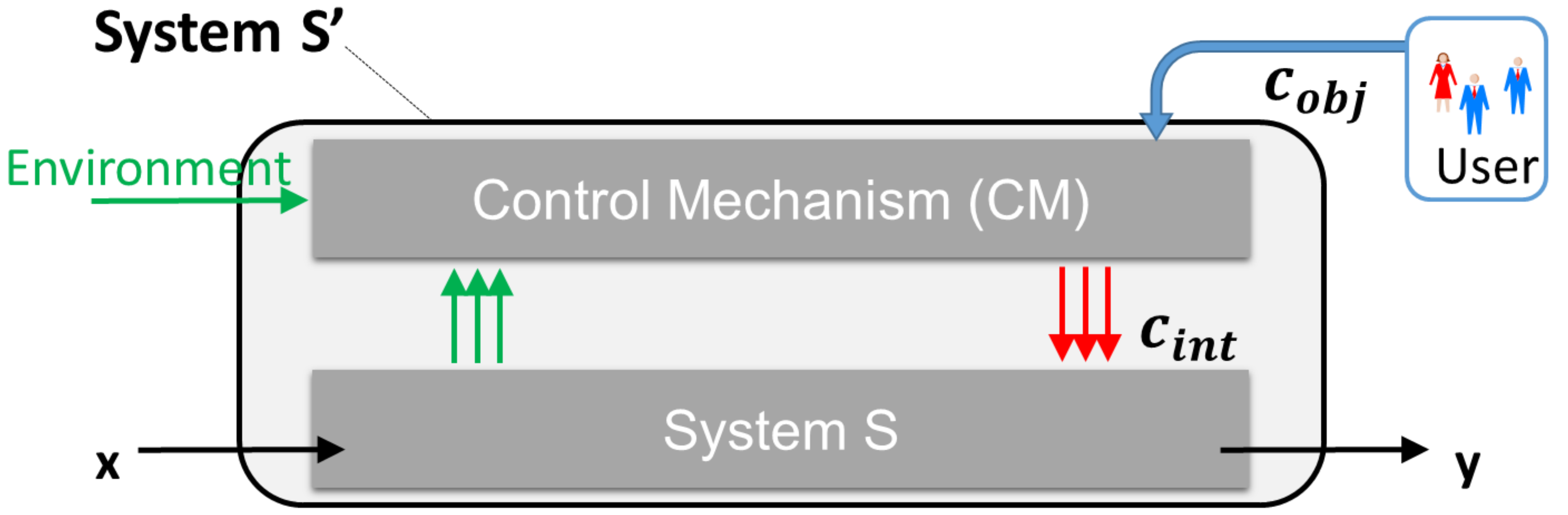
- Motivation
- Autonomy and self-organisation
- Quantification of self-organisation
- The survival cycle of an organic system
- Robustness
- Autonomy
- Conclusion and further readings



Motivation

OC systems

- Primary goal is the survival in a changing world.
- They must be robust and flexible.
  → stay in or to return to the acceptance space.
- Achieved by:
  - Adding control mechanism CM to the productive system S.
  - CM observes the state of S and that of the environment.
  - CM determines deviations of the state $\mathbf{z}$ of S from the acceptance space.
  - CM takes appropriate action to lead S back into the acceptance space.
  - Acceptance space is defined by the objective $\mathbf{c}_{\mathbf{obj}}$ (or goal) as provided by some external authority.


System S'
Environment
Control Mechanism (CM)
c_{obj}
User
c_{int}
System S
x
y

Autonomy

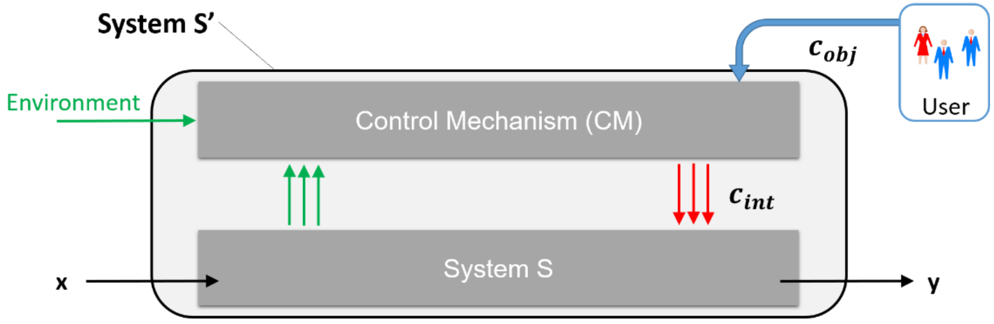
- Systems that survive in this sense are called *autonomous*.
- Term: "auto-nomy" (auto = self, nomos = law) is interpreted as obeying only some internal objectives of the system itself.
  → This is not what we want!
- System has to fulfil a certain purpose:
  - We want always to be able to control the system from the outside.
  - By prescribing goals and/or constraints the system must follow.
- But: system should act with as little external interference as possible.
- Goal: systems that keep the balance between
  - too much autonomy (makes them uncontrollable) and
  - too little autonomy (requires permanent corrective action from outside).
- Such systems are *semi-autonomous*.

In the following, we will develop a quantitative notion of the "degree of autonomy", which will allow us to capture the *semi*-autonomy more precisely.

Refining the control mechanism

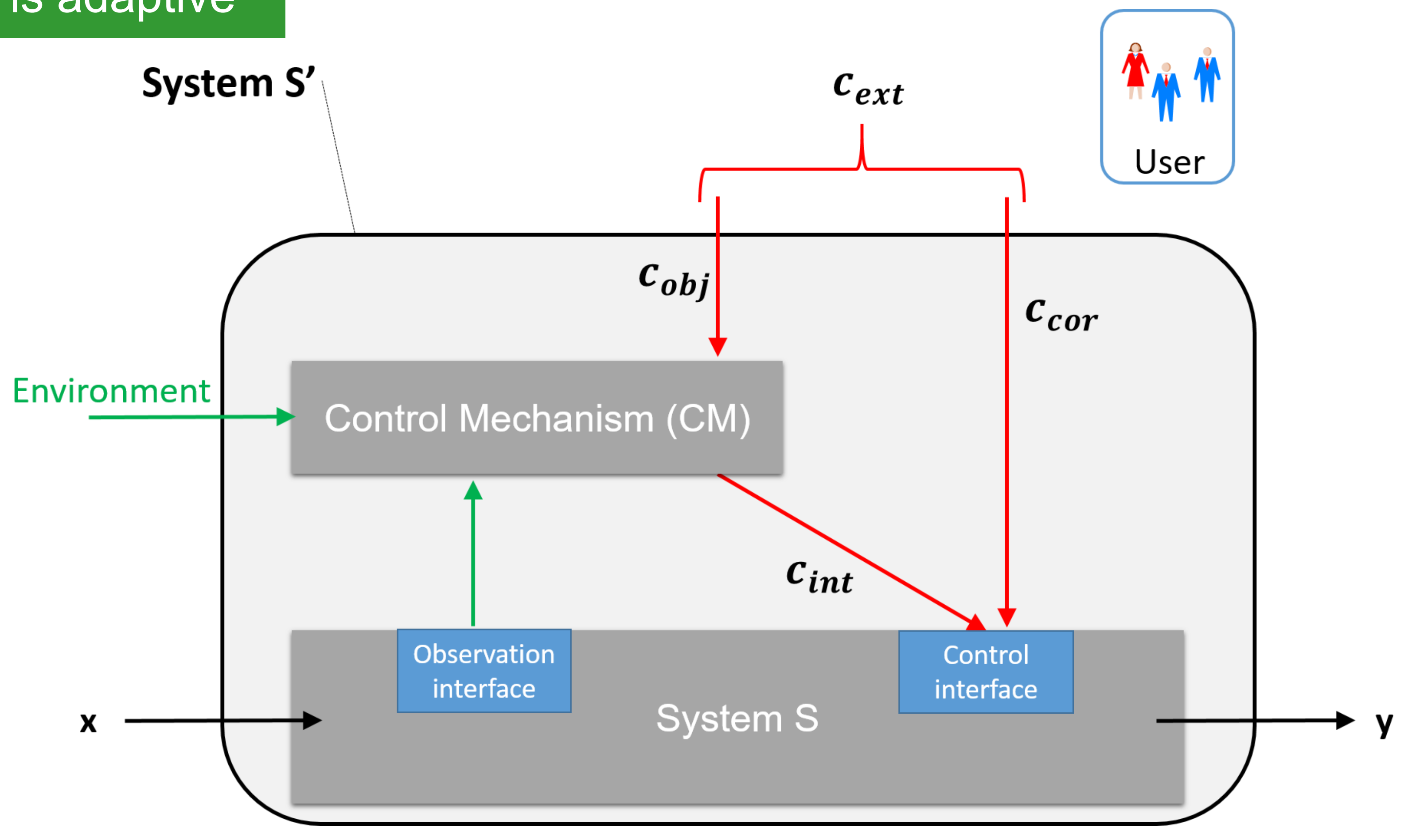
Architecture

- Refine the system architecture.
- Control mechanism
  - Has to observe and influence S.
  - Means: system must provide observation and control interfaces.
  - Observation interface defines certain internal parameters of S as visible for the CM (i.e. for the observer) for monitoring.
  - The control interface exposes certain parameters of S as modifiable by the CM (i.e.: by the controller).
  - A system S, which can be modified via a control interface, is *adaptable*.
  - Apparently, adaptability is a purely passive system property.
- If we add a CM with its active observation and control ability to an adaptable and observable system, we arrive at a system S' which can be called (semi-) autonomous.



Refining the control mechanism (2)

S is adaptable
S' is adaptive



Distinguish between control influences:

- c_{int} specifies control signals issued from within the system (i.e. from the control mechanism).
- c_{obj} specifies higher-level (and more abstract) goals issued by the user or other higher-layered systems / CMs.
  → These control signals influence *S* only indirectly.
- c_{ext} specifies control signals issued by external sources that influence *S* directly.

→ Obviously: A system S is not autonomous if the system S' adapts its behaviour only in response to control issues c_{ext}!

→ In general, external control signals are less frequently issued.

Variability

- Adaptable system S is influenced by control signals.
- Control parameters accessible for outside modification defines the possible configurations of S.
- A control vector c (comprising the parameters) applied to the control interface is a pointer selecting one possible configuration.
- The size of the configuration space is measured by the total size (in bits) of all control parameters.
- The size of the configuration space is called variability:

Variability V := #**c**

(#**c** denotes the number of bits used in **c**.)

ORGANIC COMPUTING

Examples for influencing S:

- Parameter modification can tune certain behaviours of S.
  → E.g. the timing or certain threshold values.
- Parameters can change the system structure of S.,
  → E.g. by adding or deleting edges in the communication graph of a distributed system.
- A system implemented as an FPGA (Field Programmable Gate Array) might be totally redefined by rewriting its control memory.
  → In this case, the configuration space is huge allowing all configurations acceptable by the FPGA.

Complexity reduction

ORGANIC COMPUTING

Effectiveness of an autonomous system

- $\mathbf{c}_{\mathbf{obj}}$ is (1) smaller and (2) less frequently applied than $\mathbf{c}_{\mathbf{int}}$.

  (if designed correctly).

- Quantification possible using the difference: count the number of bits necessary to express $\mathbf{c}_{\mathbf{obj}}$ and $\mathbf{c}_{\mathbf{int}}$, i.e. variabilities:

  $V_{obj} = \#\mathbf{c}_{\mathbf{obj}}$ and $V_{int} = \#\mathbf{c}_{\mathbf{int}}$.

- The difference of V_{int} and V_{obj} is the complexity reduction CR:

  $$CR = V_{int} – V_{obj}$$

- Positive value of CR: S has a larger configuration space than S';
  → CM achieves the desired complexity reduction.

- Assumption: coding of the parameters in $\mathbf{c}_{\mathbf{int}}$ and $\mathbf{c}_{\mathbf{obj}}$ is optimal in the sense that no unnecessary information is encoded.
  → Different variabilities are comparable.

Corrective control

- Perfectly designed OC system:
  - CM is able to translate higher-level control signals (expressed as objectives or goals) into lower-level internal control signals.
  - the prescribed objectives are met.
  - the system stays in or returns to the acceptance space.
- In contrast: CM may need frequent external corrections because it is not able to keep the system within the acceptance space.
- Corrections are applied in the form of additional external control signals $\mathbf{c}_{\mathbf{corr}}$.
- $\mathbf{c}_{\mathbf{corr}}$ defines an extension of the configuration space of S'.
- The total configuration space of S' is now a combination of $\mathbf{c}_{\mathbf{obj}}$ and $\mathbf{c}_{\mathbf{corr}}$.
- Combined configuration space is addressed by $\mathbf{c}_{\mathbf{ext}}$:

$$\mathbf{c}_{\mathbf{ext}} = (\mathbf{c}_{\mathbf{obj}} \,;\, \mathbf{c}_{\mathbf{corr}})$$

Corrective control (2)

If external corrections are applied:

- Variability of S' is then $V_{ext} = \#\mathbf{c}_{\mathbf{ext}}$.
- Complexity reduction CR is the $CR = V_{int} - V_{ext}$

For frequent use of external corrections:

- Many corrective actions necessary.
- V_{ext} might become even larger than V_{int} .
- Leads to a negative complexity reduction:
  → It is in this case more difficult to control S' than S!

Static degree of autonomy

Quantification of autonomy

- Use the complexity reduction CR
- Goal: define the static degree of autonomy α of a system S' as the complexity reduction CR relative to the internal variability V_{int}.

Static degree of autonomy

$$\alpha = (V_{int} - V_{ext})/V_{int} = CR/V_{int}$$

- with $0 \leq \alpha \leq 1$.

Implications:

- $\alpha = 0$ if there is no complexity reduction, i.e. $V_{int} = V_{ext}$.
- $\alpha = 1$ if $V_{ext} = 0$,
  → S' is a system, which cannot be controlled from the outside; it has a degree of autonomy of 100% (which is clearly undesirable).

Dynamic degree of autonomy

ORGANIC COMPUTING

From static to dynamic degree of autonomy

- Static degree of autonomy α is an indicator only of the *possible* control actions for S and S'.
- Does not express the actual control actions applied by CM or by the external authority to S.
- Example: Configuration space with a certain variability V might be used frequently to control or correct S or not at all.
- Goal: measure the control flow, which is actually applied via a control interface during a defined time period from t_1 to t_2.

Dynamic degree of autonomy (2)

Let $\#c(t_i)$ be the number of control bits applied at a discrete time t_i. Then we define the dynamic complexity reduction cr as

$$Dynamic\ complexity\ reduction\ cr = \sum_{t_1}^{t_2} [\#c_{\text{int}}(t_i) - \#c_{ext}(t_i)]$$

and the *dynamic* degree of autonomy β as

$$Dynamic\ \deg ree\ of\ autonomy\ \beta = \frac{cr}{\sum_{t_1}^{t_2}[\#c_{\text{int}}(t_i)]} = \frac{\sum_{t_1}^{t_2}[\#c_{\text{int}}(t_i) - \#c_{ext}(t_i)]}{\sum_{t_1}^{t_2}[\#c_{\text{int}}(t_i)]}$$

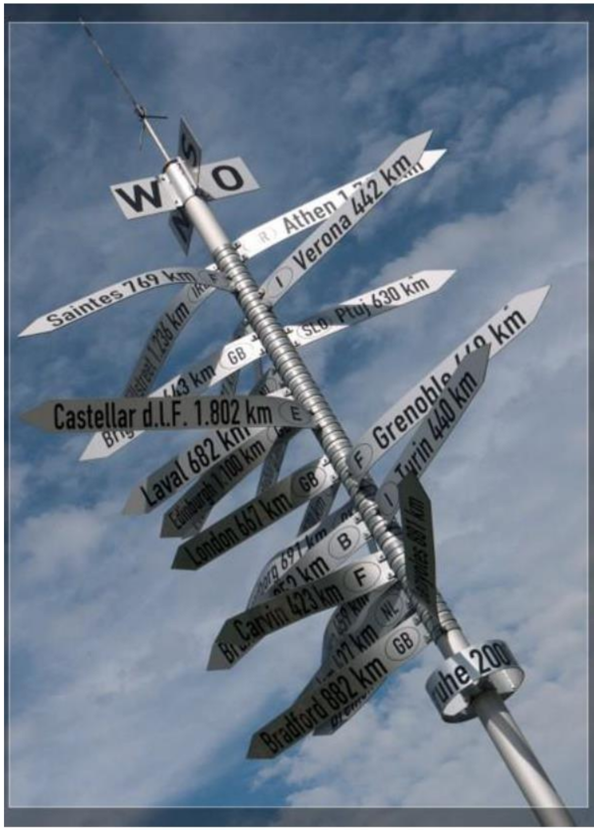
with $0 \leq \beta \leq 1$.

Implications:

As above for α, an autonomy degree of β = 0 means that internal and external control are equal, hence all the control originates from the external authority instead of CM. And β = 1 means that there exists no external control.

Agenda

- Motivation
- Autonomy and self-organisation
- Quantification of self-organisation
- The survival cycle of an organic system
- Robustness
- Autonomy
- Conclusion and further readings



Conclusion

This chapter:

- Introduced the necessary terminology for OC systems.
- Illustrated the runtime process of an OC system as survival cycle.
- Explained how major aspects of these systems can be quantified.
- Highlighted that the overall goals of organic control mechanisms are:
  - Achieve robustness
  - Reduce complexity

By now, students should be able to:

- Define what the terms self-organisation, autonomy, adaptability, utility, robustness, disturbance, and variability mean.
- Explain the behaviour of an organic system according to a state space model.
- Describe the OC survival cycle.
- Quantify robustness, self-organisation, and autonomy.

References

- Schmeck, Hartmut; Müller-Schloer, Christian; Cakar, Emre; Mnif, Moez; Richter, Urban: Adaptivity and Self-organisation in Organic Computing Systems, (Reprint), in „Organic Computing: A Paradigm Shift for Complex Systems“, Ed. Müller-Schloer, Schmeck Ungerer, Birkhäuser 2011, ISBN 978-3034-801-294 http://www.springerlink.com/content/t32485387608687w/
- Kantert, Jan; Tomforde, Sven; Müller-Schloer, Christian: Measuring Self-Organisation in Distributed Systems by External Observation. In Proceedings of the 28th GI/ITG International Conference on Architecture of Computing Systems -- ARCS Workshops, held 24 - 27 March 2015 in Porto, Portugal, Workshop on Self-Optimisation in Organic and Autonomic Computing Systems (SAOS15), ISBN 978-3-8007-3657-7, pp. 1 - 8
- Mühl, Gero, Werner, Matthias, Jäger, Michael, Herrmann, Klaus, and Parzyjegla, Helge: „On the definitions of self-managing and self-organising systems”. In Proceedings of the KiVS Workshop 2007: Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS’07). T. Braun, G. Carle, and B. Stiller Eds., VDE Verlag, 291–301

Questions …?