Robot operációs rendszerek és fejlesztői ökoszisztémák

Bevezetés

Pepó Tamás, Gincsainé Szádeczky-Kardoss Emese

2025. szeptember 8.

KUKA

MŰEGYETEM 1782

iit

Tartalom

Általános tudnivalók

A tárgy oktatói

GINCSAINÉ SZÁDECZKY-KARDOSS EMESE

BME-IIT, docens

szadeczky.emese@vik.bme.hu

IB.329

DR. KISS BÁLINT

BME-IIT, tanszékvezető

bkiss@iit.bme.hu

IB.316

PEPÓ TAMÁS

KUKA Hungária Kft, team manager

Tamas.Pepo@kuka.com

KOMLÓSI ISTVÁN

KUKA Hungária Kft, vezető fejlesztő

Istvan.Komlosi@kuka.com

SVASTITS ÁRON

KUKA Hungária Kft, senior fejlesztő

Aron.Svastits@kuka.com

PÁSZTOR KRISTÓF MÁTYÁS

KUKA Hungária Kft, gyakornok

Kristofmatyas.Pasztor@kuka.com

Oktatás módja

- Heti egy előadás (demonstrációkkal egybekötve)
  - Hétfő 12:15-14:00
  - IB.410

- Tananyag
  - Előadás fóliák és kiegészítő anyagok: https://edu.vik.bme.hu/
  - Az előadás fóliák mellett szereplő kiegészítő anyagokat nem kérjük számon a ZH-n.
  - További ismeretek: https://www.ros.org/

- Tantárgyi adatlap: https://portal.vik.bme.hu/kepzes/targyak/VIIIAV55/

Számonkérések – Projekt feladat

- 3 fős csoportok
- Feladat kiadás: szeptember 22, hétfő
- Elektronikus beadási határidő: november 30, vasárnap
- Projekt szóbeli bemutatása: december 1, hétfői órán
- Projekt feladat késedelmes elektronikus beadása: december 16, kedd
- Projekt feladat késedelmes szóbeli bemutatása: december 17-19 között, oktatókkal egyeztetett időpontban
- Laborlátogatás, megoldás tesztelése valós roboton: december 8, óra időpontjában
- Projektfeladat eredménye 50%-ban beszámít az érdemjegybe.

Számonkérések - ZH

Zárthelyi dolgozat:

- November 24.
- Előadás idejében (12:15-14:00), de az IL.406-ban
- Moodle teszt

Pót ZH:

- December 15, 12:15
- Pót-pót ZH: **NINCS!**

ZH érdemjegye 50%-ban beszámít a végső érdemjegybe.

Előismeretek

Robotikai alapok

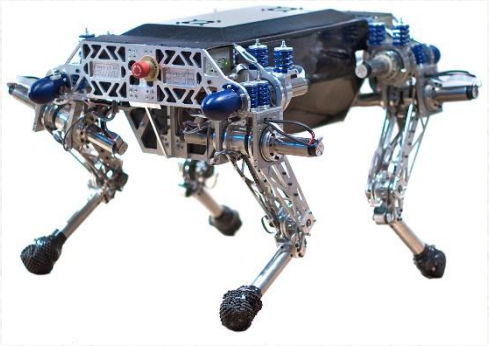
Robotok

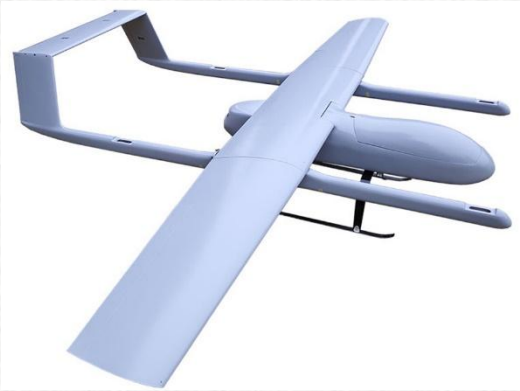
Mit nevezünk *robot*nak?

- Irányított mechanizmus
- Előírható pályán mozog
- Mozgás közben (a pálya mentén) előírható feladatokat lát el
- Lehet helyhez kötött és/vagy helyváltoztatásra képes

Robotok típusai

Mobilis robotok

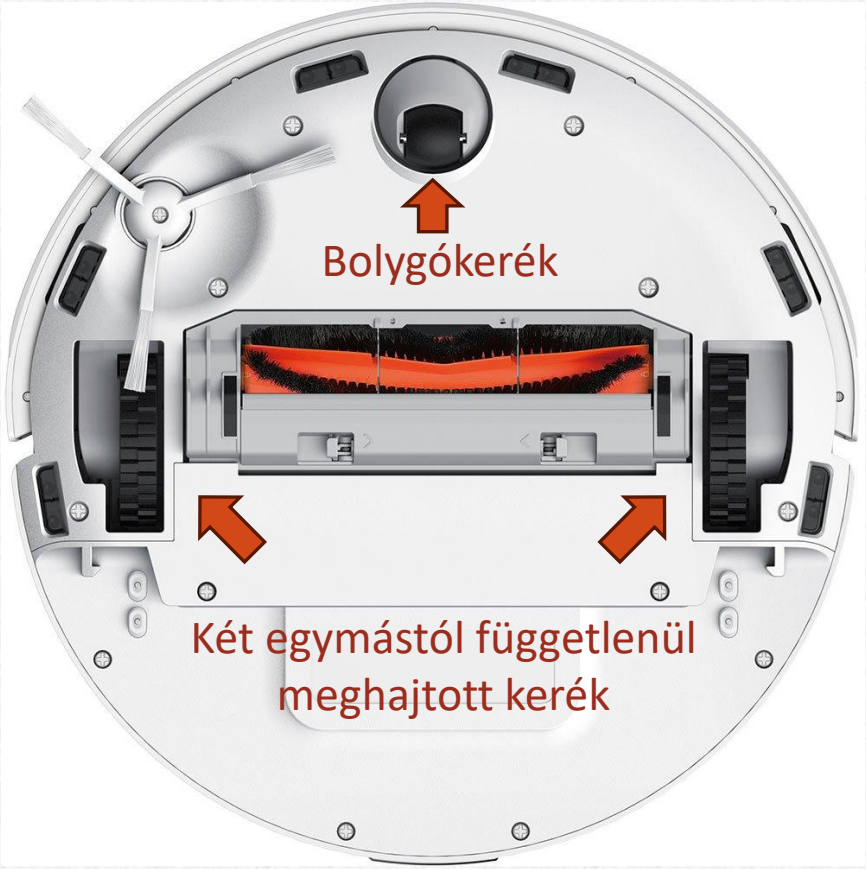




- Helyváltoztatásra képesek
- Többféle csoportosítás
  - Hol mozog (pl. földi, vízi, légi)
  - Mi mozgatja (pl. kerekek, propellerek, lábak)
  - Hogyan mozog (távirányításos vagy önműködő)

Tipikus kerék-meghajtások

Differenciális meghajtás

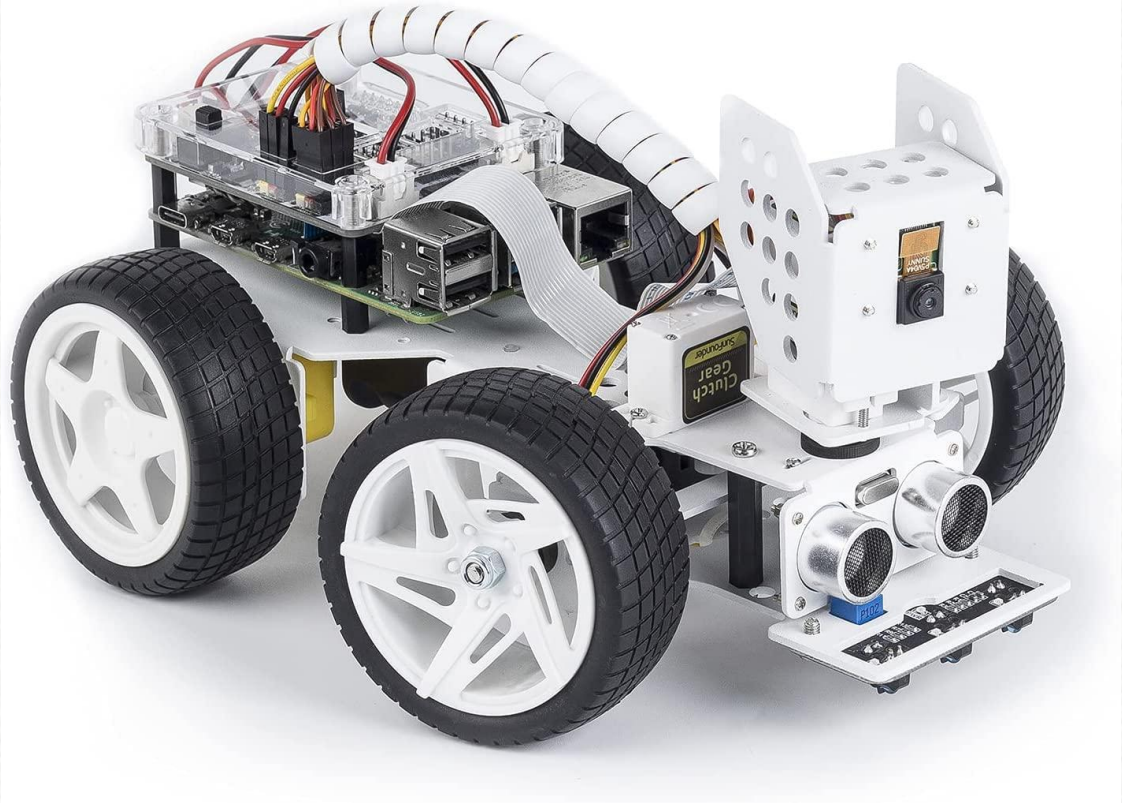


Tipikus kerék-meghajtások

Omnidirekcionális meghajtás

Tipikus kerék-meghajtások

Autószerű robotok

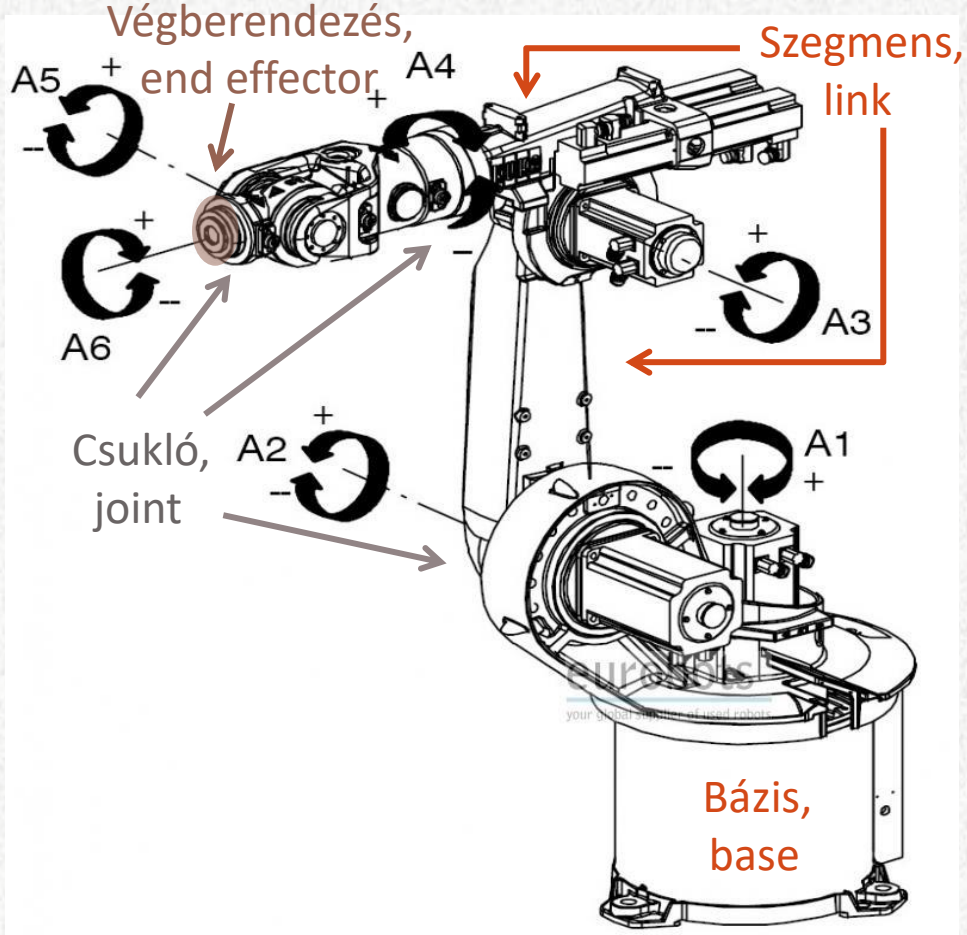


Ipari robotkarok – tipikus felépítés

KUKA KR15

Felépítés:

- Fix *bázis* (base)
- *Szegmensek* (link) merev testekből
- *Csuklók* (joint) biztosítják a szegmensek egymáshoz képesti elmozdulását
- *Végberendezés* (end effector)
  - Elvégzendő feladatnak megfelelő eszköz (pl. megfogó, szerszám)



Ipari robotkarok – alapfogalmak

- Csuklók típusai:
  - ***T**ranszlációs csukló* (prismatic joint) – lineáris elmozdulást biztosít a szegmensek között
  - ***R**otációs csukló* (revolute joint) – a szegmensek egymáshoz képesti elfordulását valósítja meg
  - Egy vagy több tengely menti/körüli elmozdulás
- *Csuklóképlet* (joint formula): a csuklók típusát és sorrendjét adja meg. Például: RRTR
- *Szabadságfok* (*DoF*, degree of freedom): a csuklótengelyek száma
- *Csukóváltozó* (joint variable): i. csuklótengely körüli/menti mozgás mértékét adja meg (q_i)
  - T csuklónál elmozdulás (d_i)
  - R csuklónál elfordulás (θ_i)
  - Az összes csuklóra vektorba rendezve (*konfiguráció*): $\mathbf{q} = (q_1, \dots, q_m)^T$

Ipari robotkarok – pár típus

Nyílt láncú, elágazás nélküli robotok

Zárt láncú robot

Delta

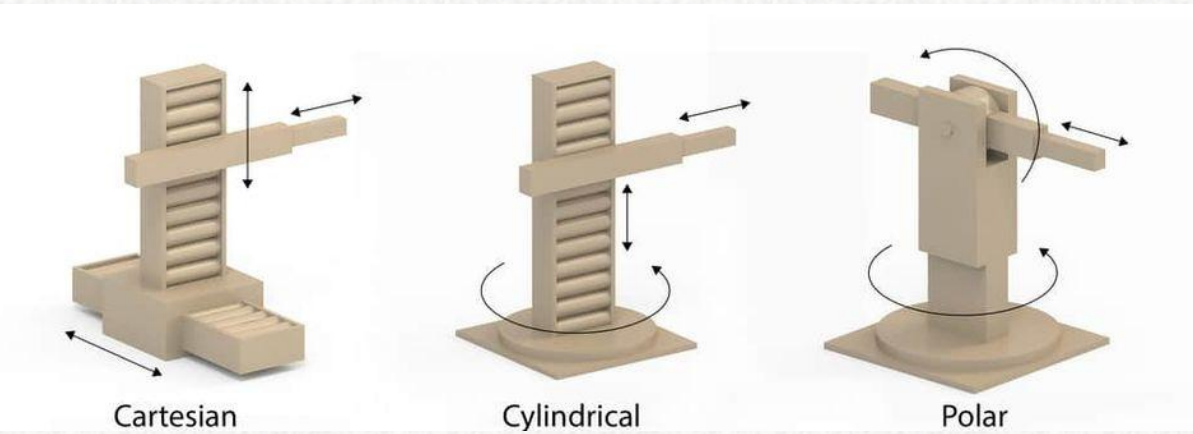
Ipari robotkarok – cobot

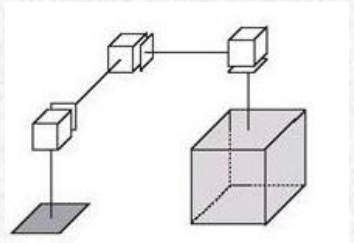
- *Kollaboratív robotok*:
  - ember közelében (robotcella nélkül) dolgoznak vagy
  - emberrel közös térben működnek vagy
  - emberrel együtt dolgoznak
- Biztonsági megoldások
  - Kialakítás
  - Korlátozott sebesség és nyomaték
  - Szoftveres megoldások
- Robotkar mozgatása emberi erővel

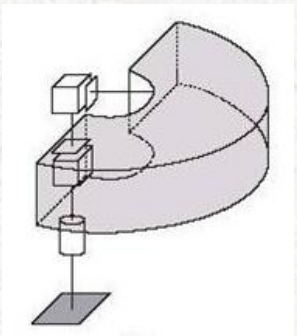
Ipari robotkarok – munkatér

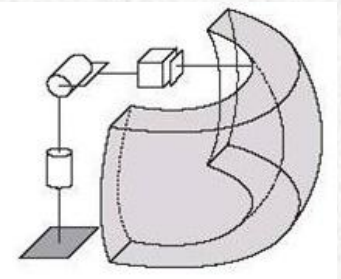
A *munkatér* (workspace) a robot végberendezése által elért tér.

SCARA





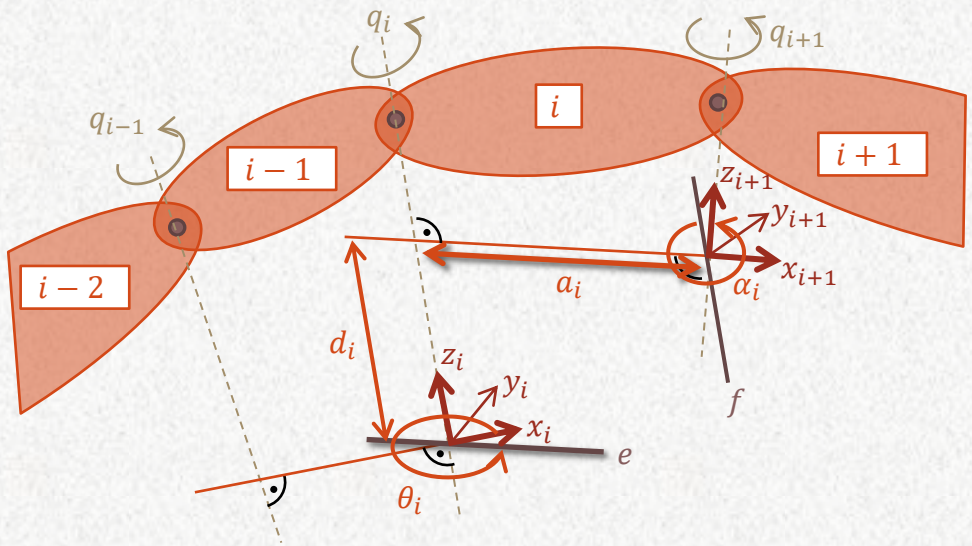




Ipari robotkarok – további elemek

- Érzékelők:
  - Belső érzékelők – csuklóváltozókat és deriváltjaikat mérik
  - Külső érzékelők – közvetlen pozíció/orientáció mérés, erő/nyomaték mérés, taktilis érzékelés
- Beavatkozók:
  - Motor + áttétel
- Szabályozó + teljesítmény elektronika
- Betanító pult

Ipari robotkarok – geometriai modell



Denavit-Hartenberg (DH) alak:

- megadja a csuklótengelyekhez rendelt koordináta-rendszerek egymáshoz képesti helyzetét

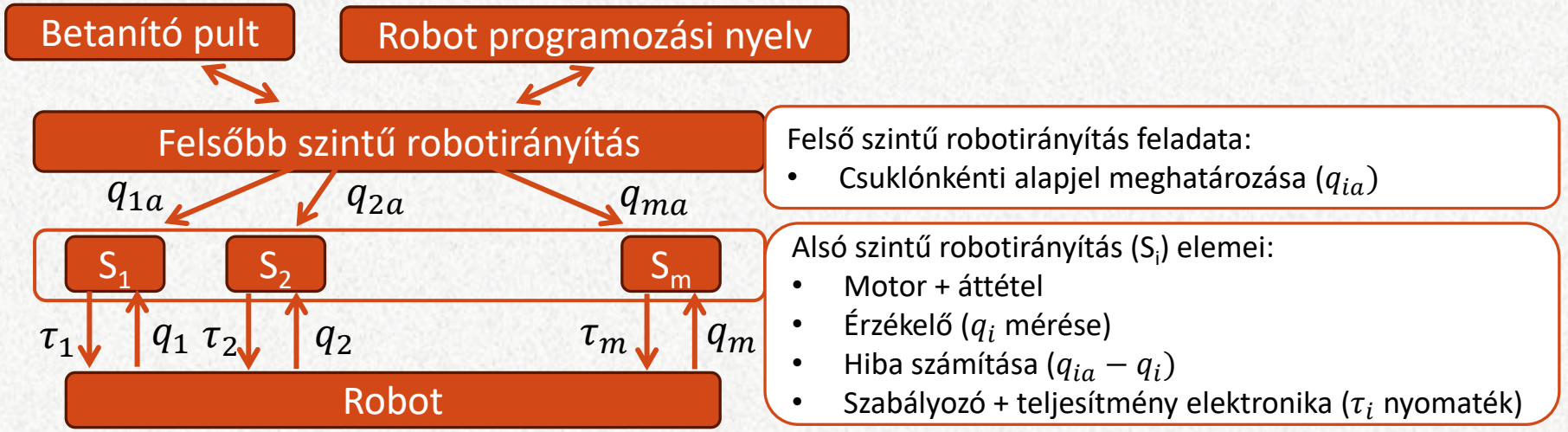
Csuklónként négy lépés:

- z_i körül forgatás θ_i szöggel (x_i-t e-be)
- z_i mentén eltolás d_i távolsággal (e-t x_{i+1}-be)
- x_{i+1} mentén eltolás a_i távolsággal (z_i-t f-be)
- x_{i+1} körüli forgatás α_i szöggel (f-et z_{i+1}-be)

Csuklónként négy DH parameter: $\theta_i, d_i, a_i, \alpha_i$

Ipari robotkarok irányítása

- Előírhatjuk:
  - Csukló nyomatékokat vagy erőket
  - Csuklóváltozók értékét
    - Ha tudjuk, melyik csukló hogy áll, és mik a DH paraméterek, meg tudjuk határozni a végberendezés pozícióját (direkt geometriai feladat)
  - Végberendezés pozícióját
    - Ebből vissza kell(ene) számítani a csuklóváltozó értékeket (inverz geometriai feladat)



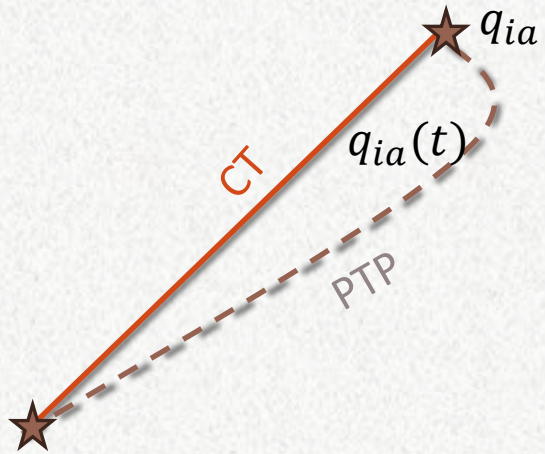
Felső szintű robotirányítás feladata:
- Csuklónkénti alapjel meghatározása (q_{ia})

Alsó szintű robotirányítás (S_i) elemei:
- Motor + áttétel
- Érzékelő (q_i mérése)
- Hiba számítása ($q_{ia} - q_i$)
- Szabályozó + teljesítmény elektronika (τ_i nyomaték)

Ipari robotkarok irányítása

Kétféle irányítási lehetőség

- *Pont-pont irányítás* (PTP, point-to-point control)
  - Nem definiáljuk a teljes referencia mozgást ($q_{ia}(t)$)
  - Csak a pálya végpontja adott (q_{ia})
  - $q_{ia}(t)$ trajektóriát szabályozás adja (tranziensekkel)
  - Ütközésveszély a környezettel

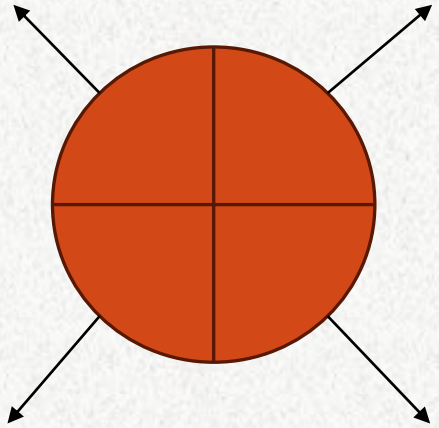


- *Folytonos pályairányítás* (CT, continuous path control)
  - Teljes $q_{ia}(t)$ időfüggvény definiált
  - Pálya megadható csuklóváltozókban vagy munkatérben (pl. interpoláció használatával)
  - Kicsi az ütközésveszély

SoA rendszerek

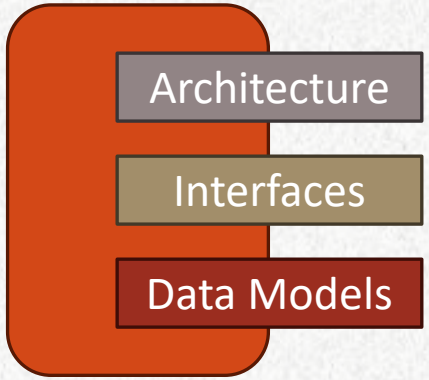

Lufthansa

SoA – Service-oriented Architecture



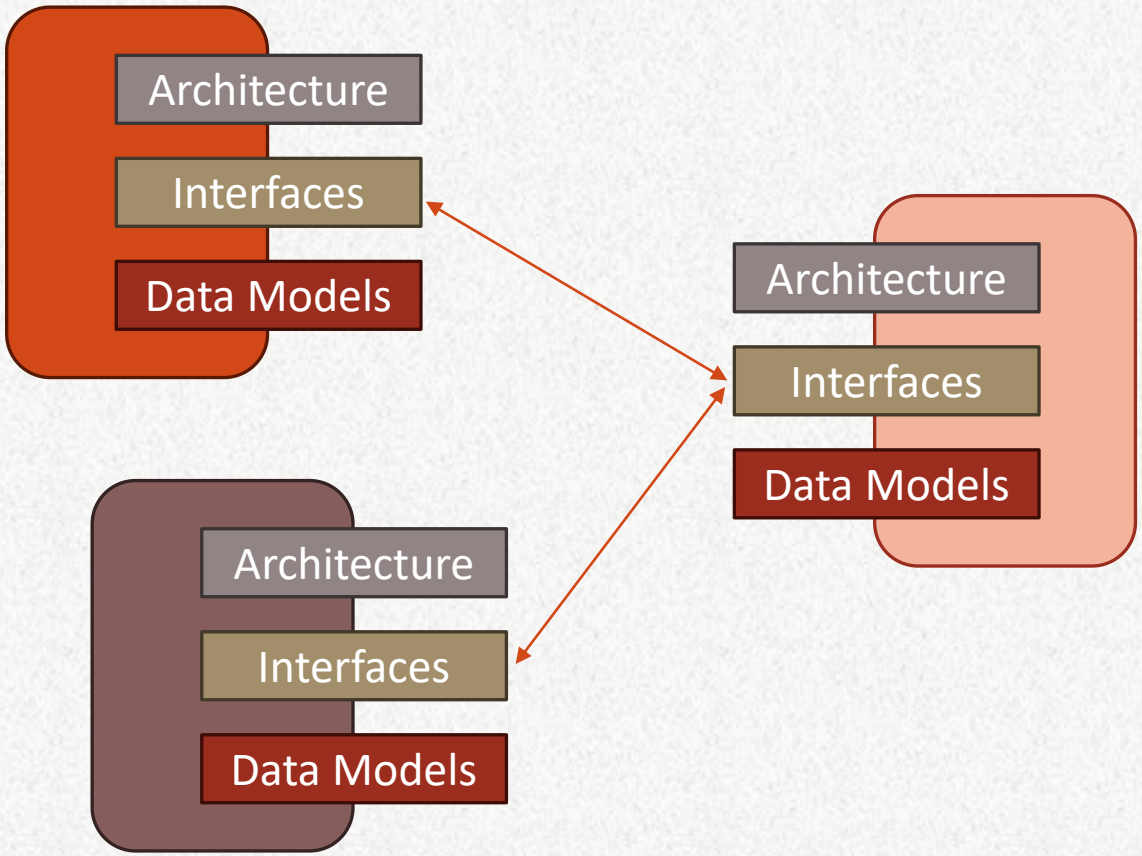
- "Service-oriented Architecture"
- Services shall be…
  - Reusable
  - Interoperable
- System integration pattern
  - Faster time to market
  - Efficient maintenance
  - Greater adaptability
- Services reusable on the scope of the enterprise
  - Representing a business capability

Describing a service



- Services have an internal **architecture**
  - *Not relevant for consumers of the service*
- Services share their behavioral description (**Interfaces**)
- Services share their **data models** on which they work

Essence of SoA



- Standardized interface...
  - Technology
  - Description
- Common data meta-models
  - Conversion of data possible
- Communication across
  - Platforms
  - Languages

Principles of SoA

According to **Amazon**, the basic principles of SoA are:

- Interoperability
  - **Any client system can run** the service, interact with it
  - Functionalities and capabilities are well defined
- Loose coupling
  - **Stateless** services
  - Little to no dependency to external resources
- Abstraction *(see Describing a service slide!)*
  - Consumers don't need the knowledge of implementation details -> **black box**
- Granularity
  - One discrete **business function** per service

Communication in SoA

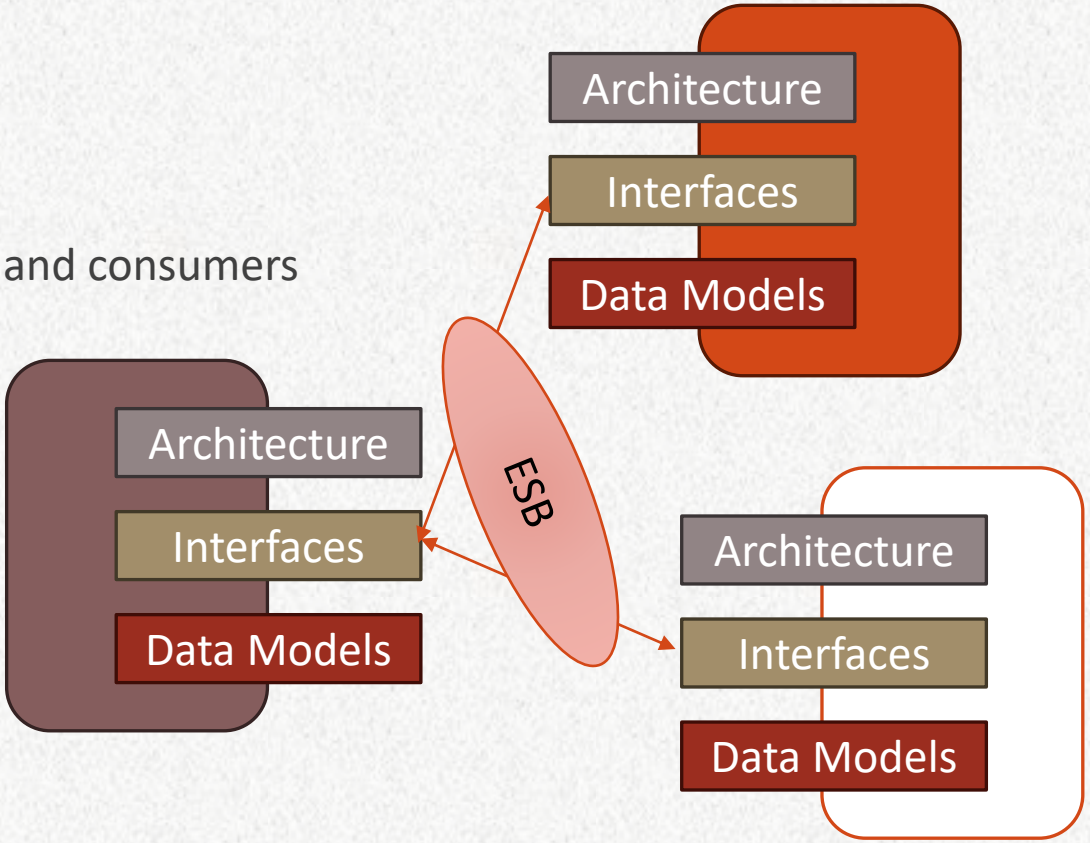
Usually runs through an ESB

- "Enterprise Service Bus"

The ESB...

- Establishes communication between service providers and consumers
- Provides communication patterns, capabilities
- Transforms data
- Based on standardized communication technologies

However, the ESB is a central component, thus, is against the concept of decentralization



Microservices

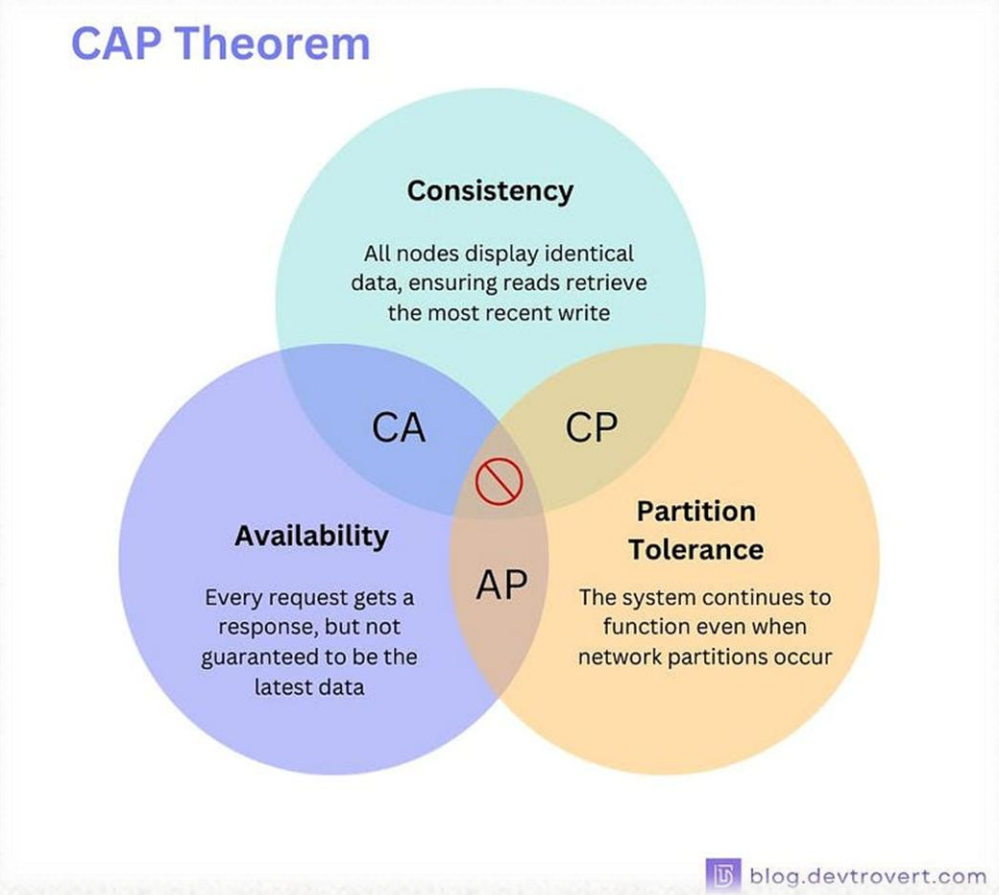
Microservices is an evolution of SoA

- Made up of very small and independent software components
- According to Martin Fowler: microservices are the natural consequence of applying the single responsibility principle at the architectural level

Key differences to basic SoA

- Focuses on cloud-compatibility
- Fine grained, very small services ("micro")
- Connected through their APIs usually directly
  - Removes the need for an ESB
- Favor data duplication to data sharing
- High performance
- Higher agility

CAP Theorem



- Consistency
- Availability
- Partition Tolerance

Robotic systems on SoA principles

SHALL WE USE BASIC SOA, OR MICROSERVICES?
WHAT ARE THE CORNERSTONE TOPICS?

Structure of a traditional robot system

Robot

Controller + Robot OS

Control Point
„SmartPad”

Engineering Suite

Structure of a modern robot system

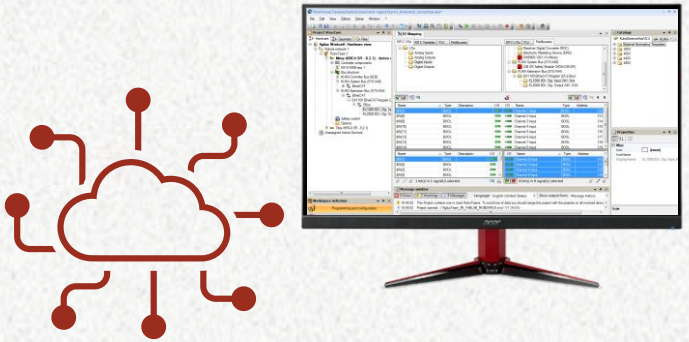
Robots

Controller +
Distributed Robot OS

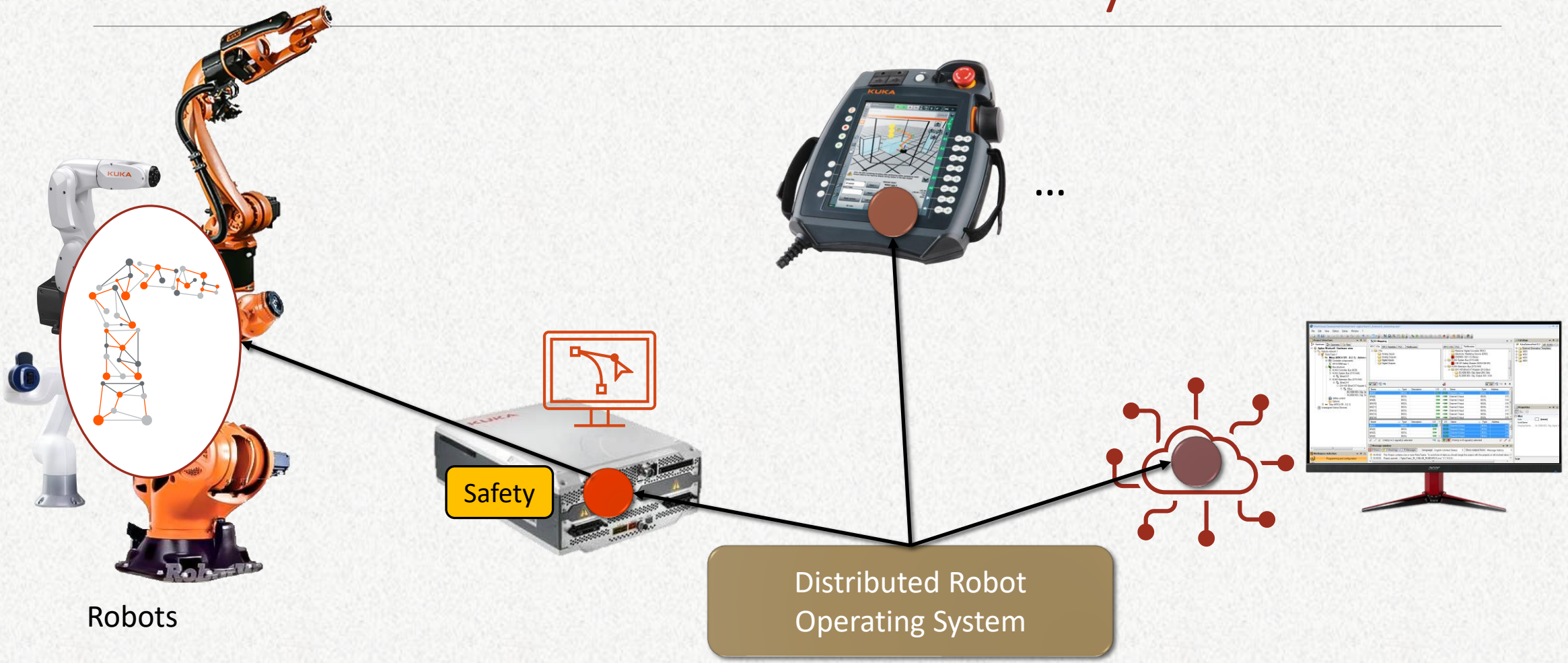
Control Points

...

Engineering Suite + Control Points



Structure of a modern robot system



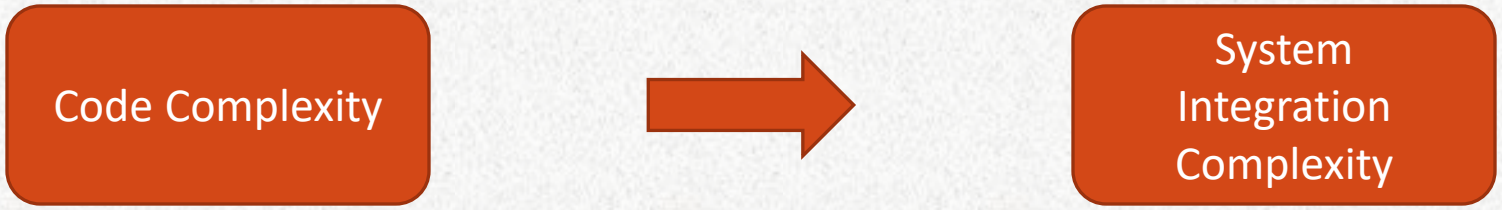
Problems in a modern robot system

- Services are dependent on the machine data & description of the robot
- Services are dependent on the environment data & fieldbus configurations
- Various system topologies based on estimated resource bottlenecks
- Dynamically changing system topologies
- Fail-safe approach to achieve maximum time between breakdowns (single-point-of-failure)
- Complex variant management
- Complex interfacing to external ecosystems
- Provision of real-time capabilities among distributed services

Complexity

- We have torn apart a system to reduce complexity…
  - … and now, we are speaking of complex problems that are hard to solve!

- Both *SoA* and *Microservices* are moving complexity from code to system integration.



- But what is commonly forgotten, is that **not only the system integration, but also system testing, variant handling, deployment, and in general, management gets more complex!**

Tackling the complexity monster

- Testing on multiple levels are inevitable
  - As services are deployable on their own, systems are to be released more frequently
  - Testing on system level can’t cope with the exponential effort
- Keeping track of equivalences, system variants
- Semantic Versioning, explicit dependency graphs (provided contracts to consumed contracts)
- Continuous Integration (Why not Continuous Deployment?)

An Ideal Robot OS

- Scalable in the resources & capabilities dimension
- Scalable in the machine portfolio dimension
- Offers standardized capabilities & interfaces
- Can be certified for safety standards
- Easily testable
- Cloud & simulation support

Does ROS fulfill these?

Is ROS 2 performing better?