MEIM | MASTER IN ENTREPRENEURSHIP INNOVATION MANAGEMENT IN COLLABORATION WITH MIT SLOAN

IN COLLABORATION WITH MIT MANAGEMENT SLOAN SCHOOL | UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

MASTER MEIM 2023

Robotics. Design of control systems

Prof. Fabio Ruggiero, Ph.D.

www.fabioruggiero.name

fabio.rugiero@unina.it

A cura del Prof. Fabio Ruggiero

Professore Associato di Robotica e Controlli Automatici presso l'Università degli Studi di Napoli Federico II

What is automation?

- Automation means using computer software, machines or other technology to carry out a task which would otherwise be done by a human worker
- The term was introduced by Ford Motor in the 1946 during the mass production era
- From mass production to mass customization

The role of automation

↑ product quality
↑ flexibility
↓ production time
↓ production waste
↓ product costs
↑ compliance to terms and regulation
↓ environmental impact
↓ energy consumption
↑ competitiveness

Automation principles: centrifugal governor

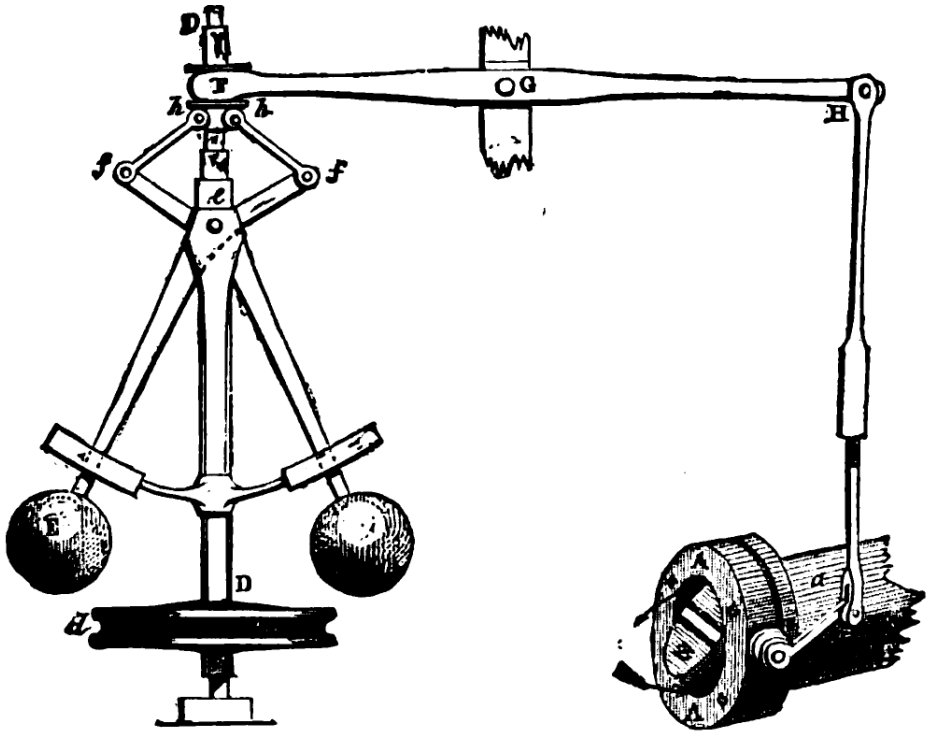
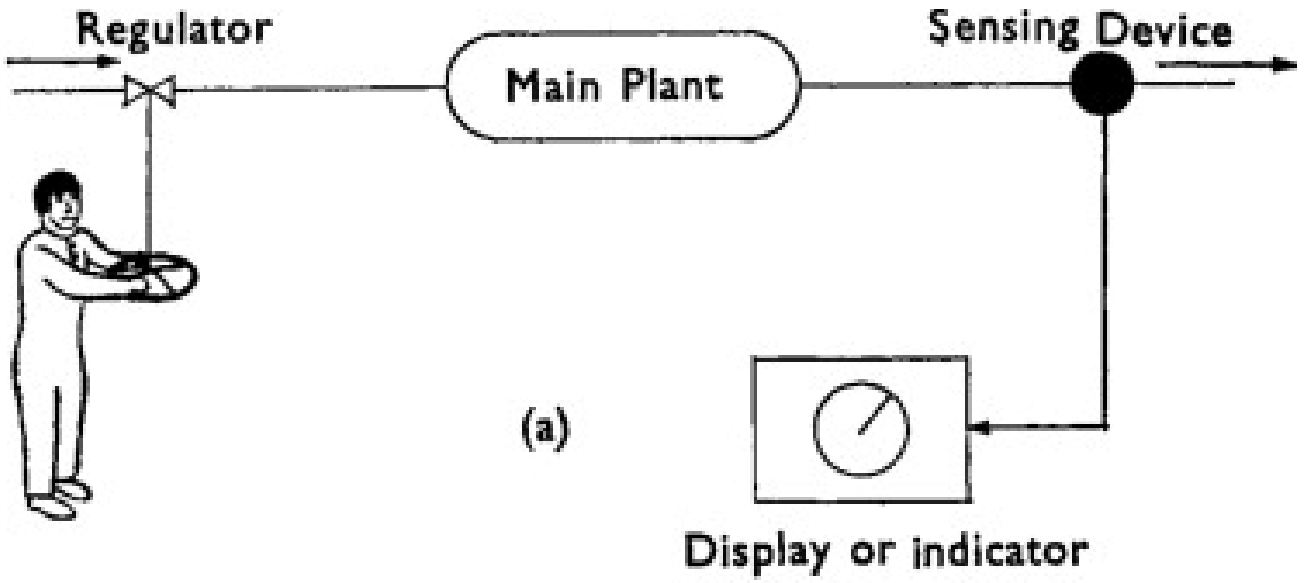


FIG. 4.—*Governor and Throttle-Valve.*

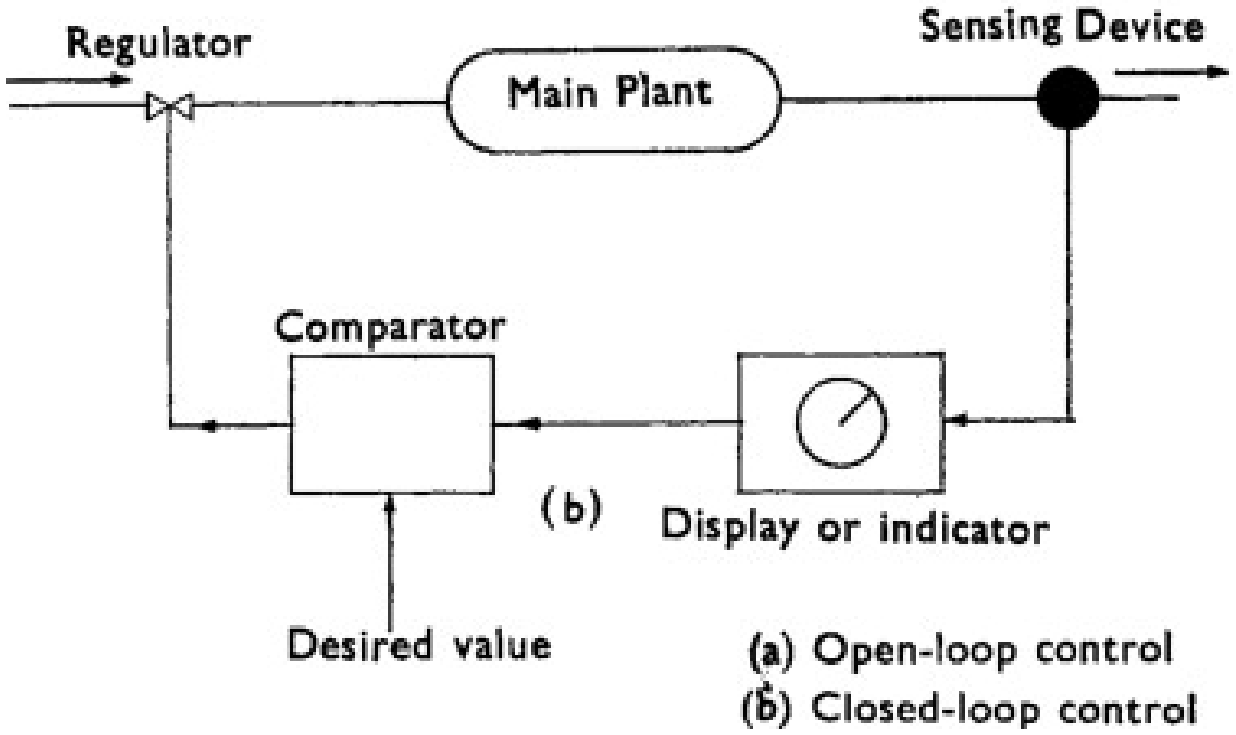
https://en.wikipedia.org/wiki/Centrifugal_governor

Automation principles: manual vs automatic control



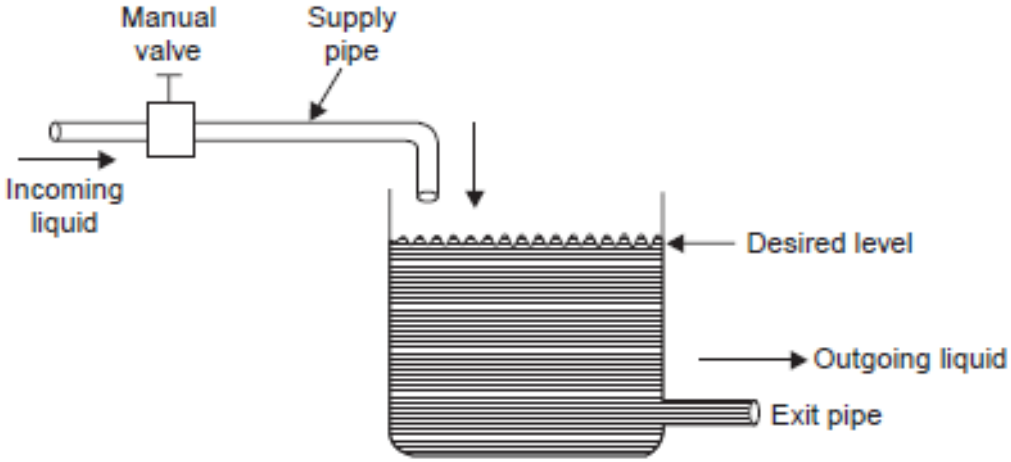
D. Gray, in Centralized and Automatic Controls in Ships, 1966

Automation principles: manual vs automatic control



D. Gray, in Centralized and Automatic Controls in Ships, 1966

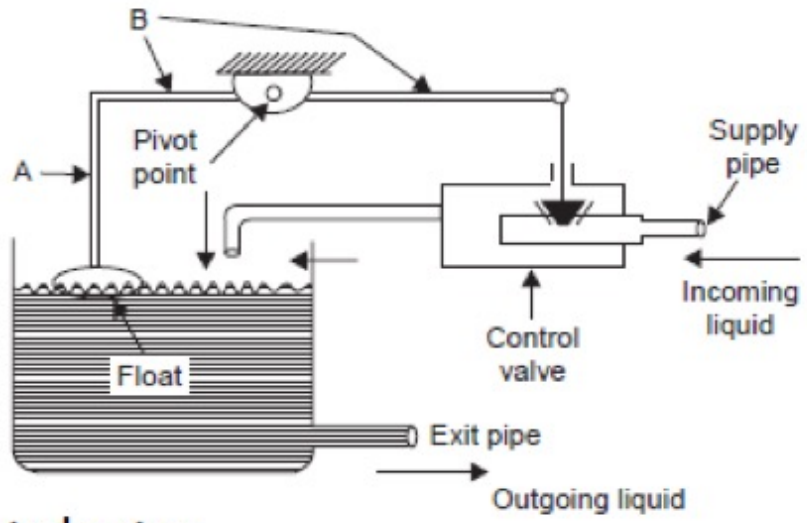
Automation principles: open loop vs closed loop



Open loop control system

http://www.electronicsengineering.nbcafe.in/what-is-an-automatic-control-systems/

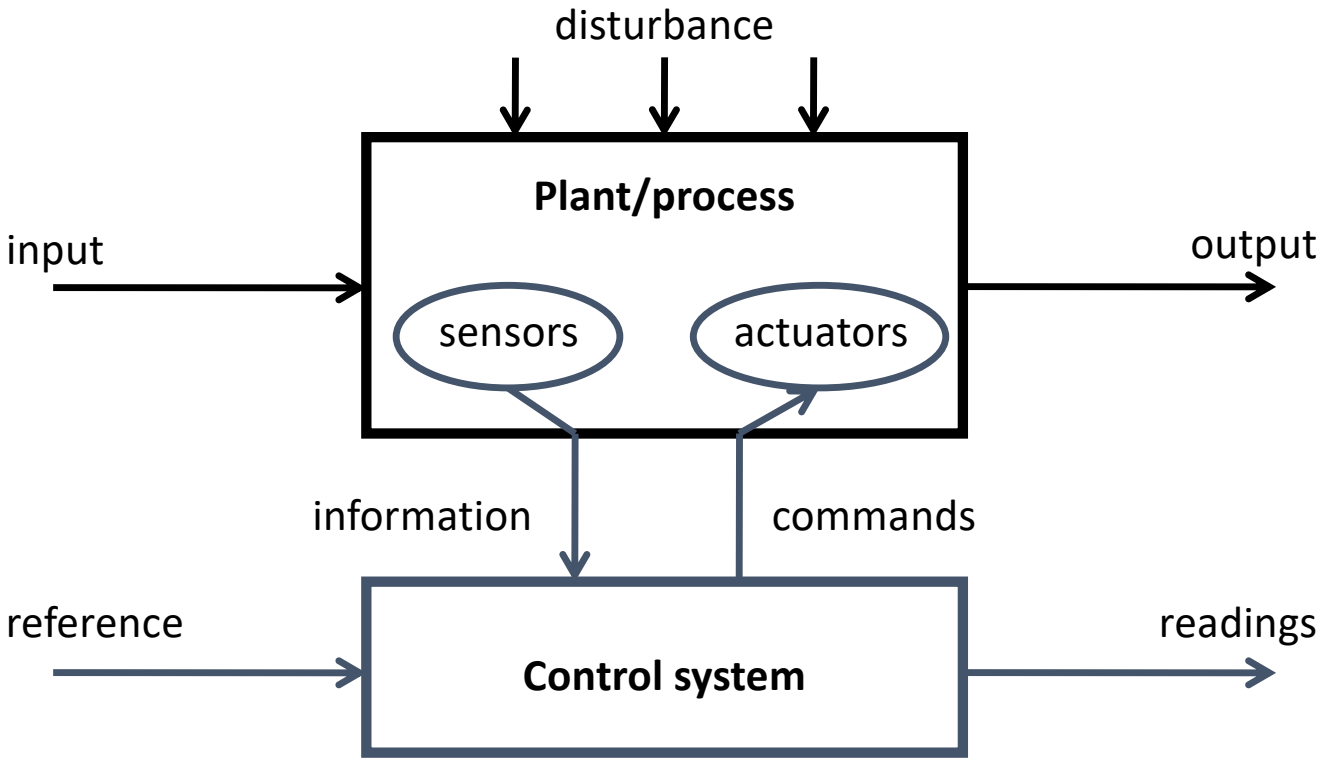
Automation principles: open loop vs closed loop



Closed loop control system

http://www.electronicsengineering.nbcafe.in/what-is-an-automatic-control-systems/

Automation principles: control systems



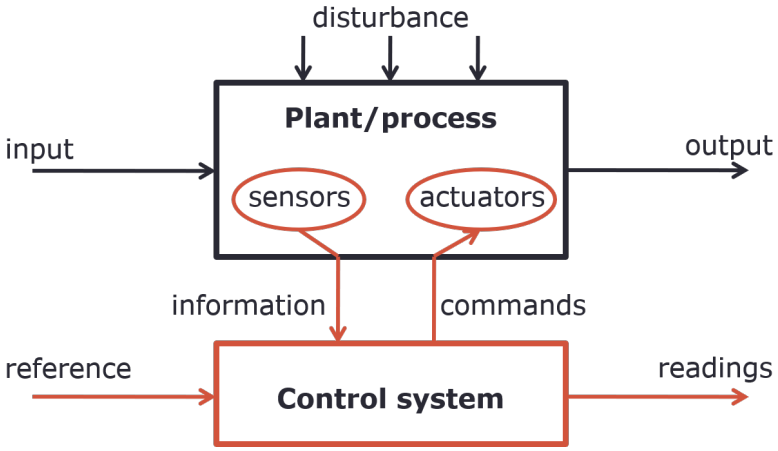
Automation principles: control systems

Essential components of a control system

- Plant/process
- Sensors
- Actuators
- Control

Additional components of a control system

- Governor unit
- Human-machine interface
- Disturbances
- ...



What is automation?

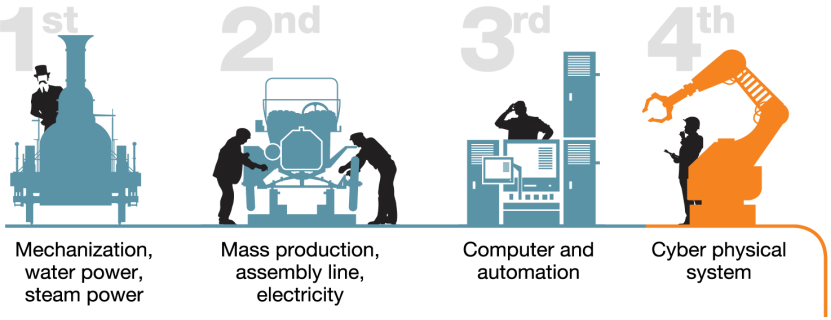
https://www.youtube.com/watch?v=9SIrEXDivrQ

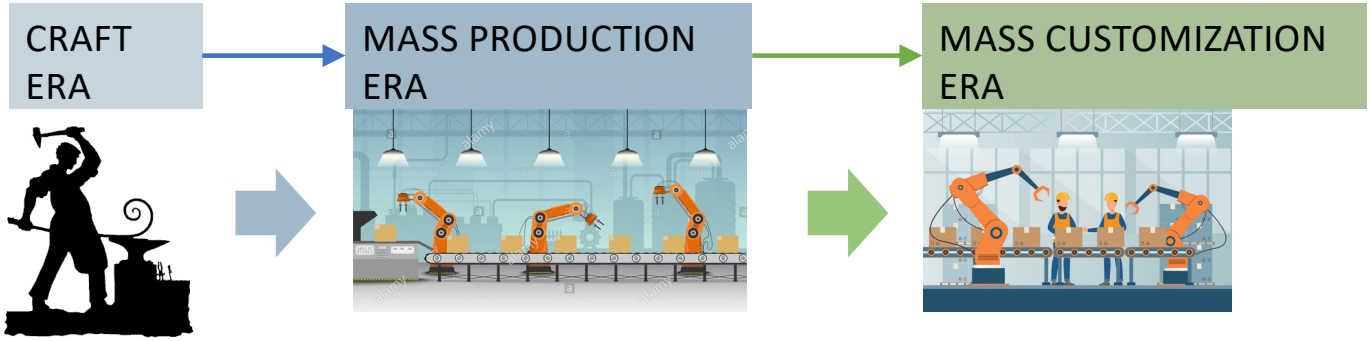
What is robotics?

- *The Chamber's Dictionary* (2003): Robotics is the branch of technology dealing with the design, construction and use of robots
- Robots have appeared extensively in the artistic field of science fiction writing (e.g. *I, Robot* - based on Isaac Asimov's stories)
- The actual name robot arose from its use by the playwright Karel Capek in the play *Rossum's Universal Robots* (1920)

Robotics for industry 4.0

In the fourth industrial revolution, digital analytics enables a new level of operational productivity.



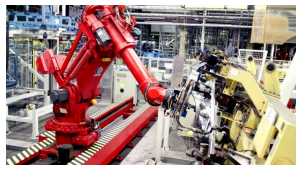


MEIM | MASTER IN ENTREPRENEURSHIP INNOVATION MANAGEMENT IN COLLABORATION WITH MIT SLOAN

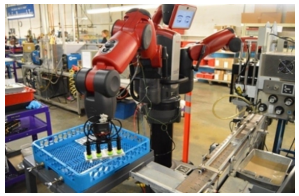
UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

Robotics for industry 4.0

LEVEL OF AUTONOMY

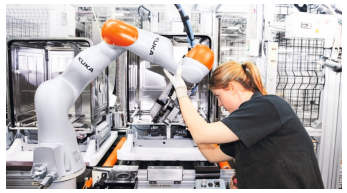


automatic machines executing programmed tasks with high levels of accuracy, speed and repeatability in a perfectly known environment



machines equipped with sensors (vision, distance, force,…) for perceiving the external environment and with decision-making capabilities

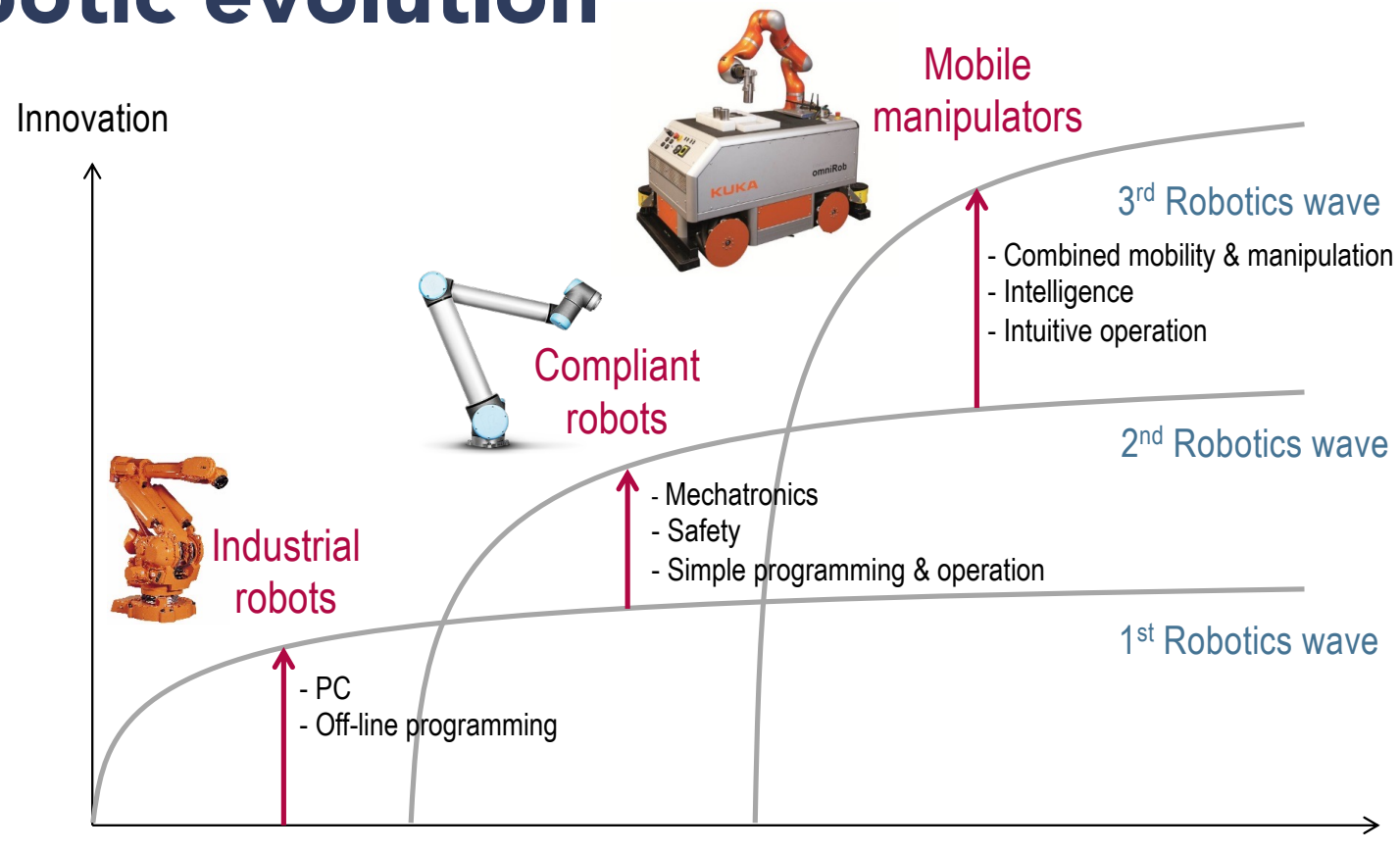
machines confined in fenced areas, designed to handle the dull, dirty and dangerous tasks in place of human workers



lightweight machines able to work safely in the same workspace or even physically collaborate with humans

SAFETY AND COLLABORATION CAPABILITY

Industrial robotic evolution

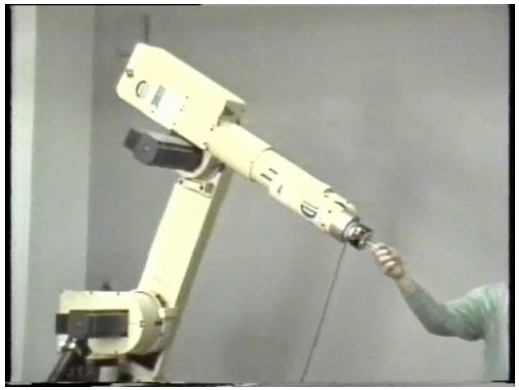


From motion control to interaction control

motion control

compliance control

impedance control

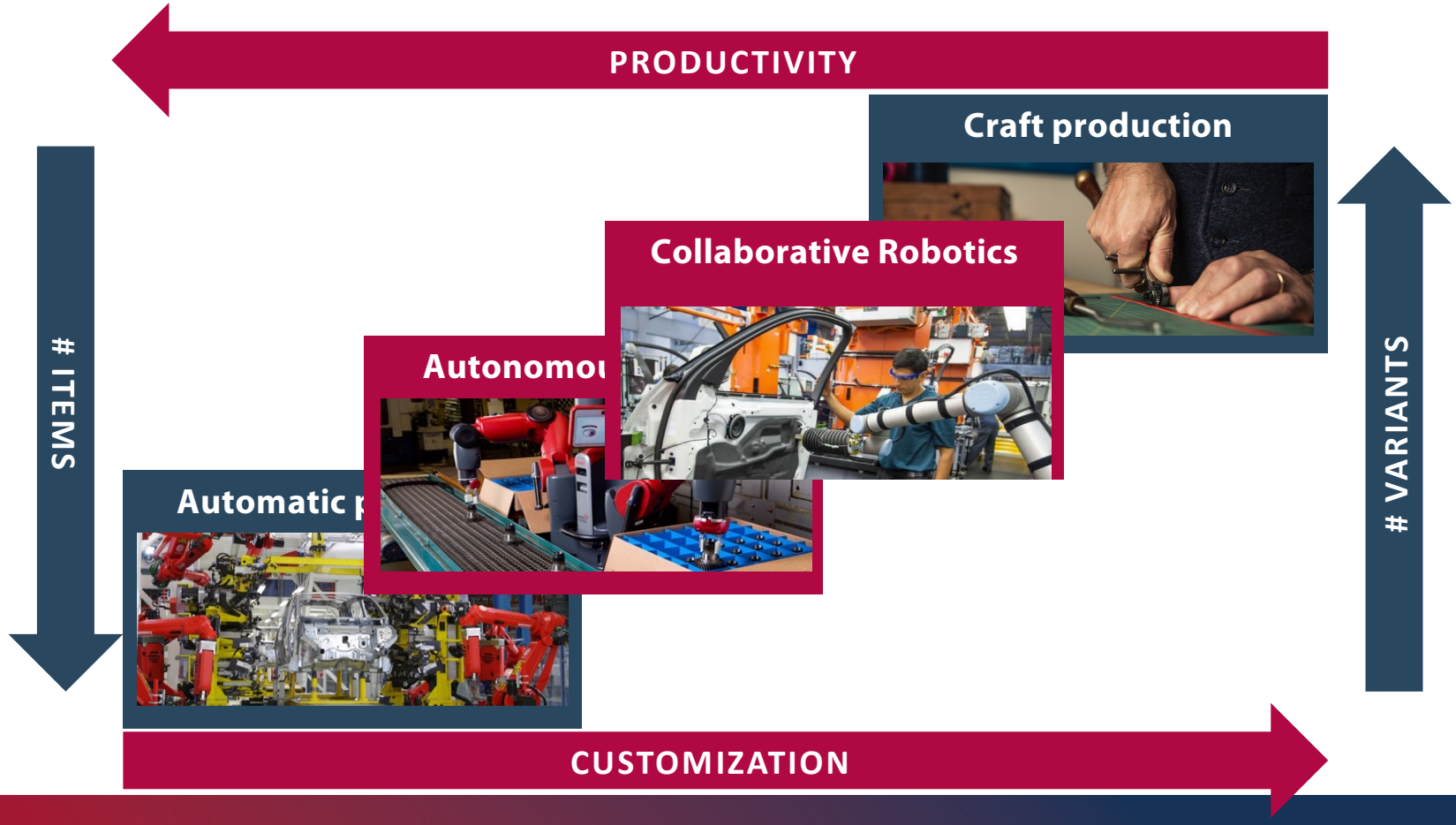


force control

MEIM | MASTER IN ENTREPRENEURSHIP INNOVATION MANAGEMENT IN COLLABORATION WITH MIT SLOAN

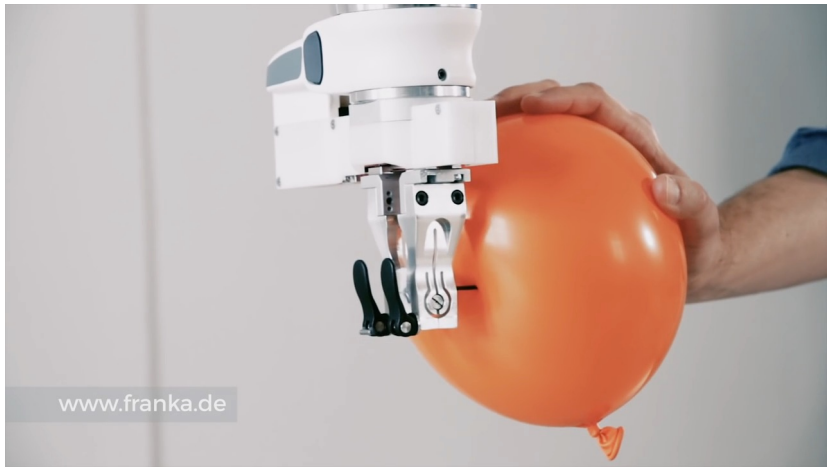
UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

Autonomous and collaborative robotics



COllaborative roBOTS

- can be used safely in a space shared with humans
- special mechanical characteristics, extroceptive sensors, advanced control system
- intuitive programming and communication interface
- fast setup, commissioning, and reconfiguration
- low costs (<20k) and suitable for Small and Medium Enterprises

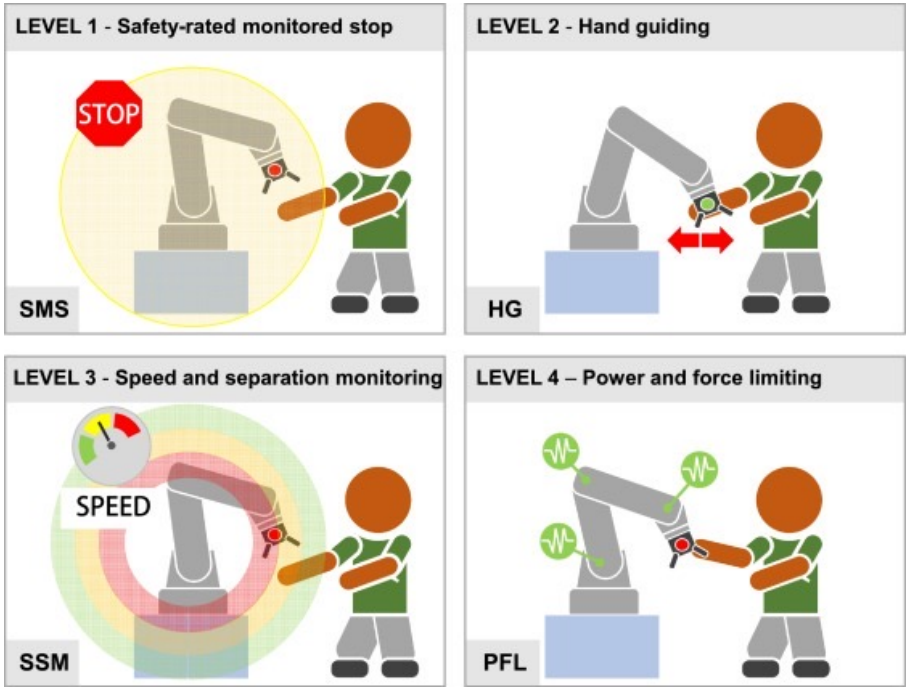


Safety standards

ISO 10298-1/2 -> safety requirements and guidelines for robots in industrial environments

ISO/TS 15066 -> technical specifications for collaborative robot system safety

4 levels of collaborative operations



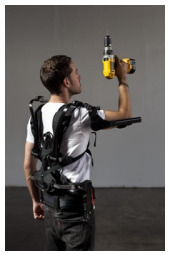
Cobots: hw and sw

- Mechanics
  - lightweight
  - redundant/double arms
  - soft covers, no edges
  - elastic joints
- Sensors
  - joint torques
  - force/torque at the end effector
  - 3D vision
  - sensitive skin
- Control
  - compliant , collision detection, collision avoidance
  - coexistence/cooperation/collaboration

Robots and Humans working as one

- Wearable robots: Exoskeletons
  - composed by a frame fitted with (motorised) muscles supporting parts of the human body
  - they allow to multiply the strength of its user's or to redistribute the weight
  - enable workers to carry out a variety of industrial tasks
  - protect workers from the heavy physical workload, repetitive movements and non-ergonomic postures


arm support



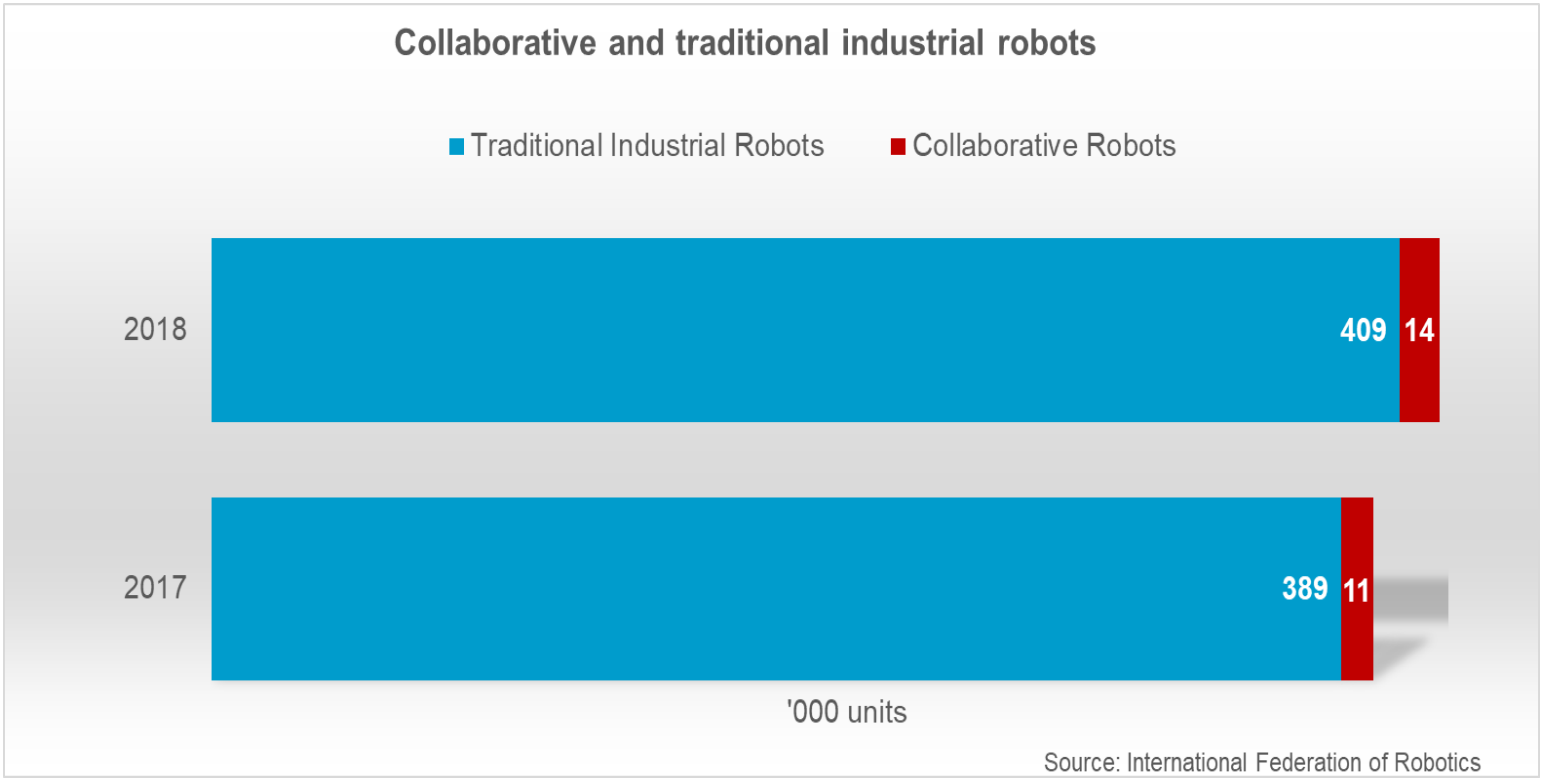

back support


legs support


tool holding

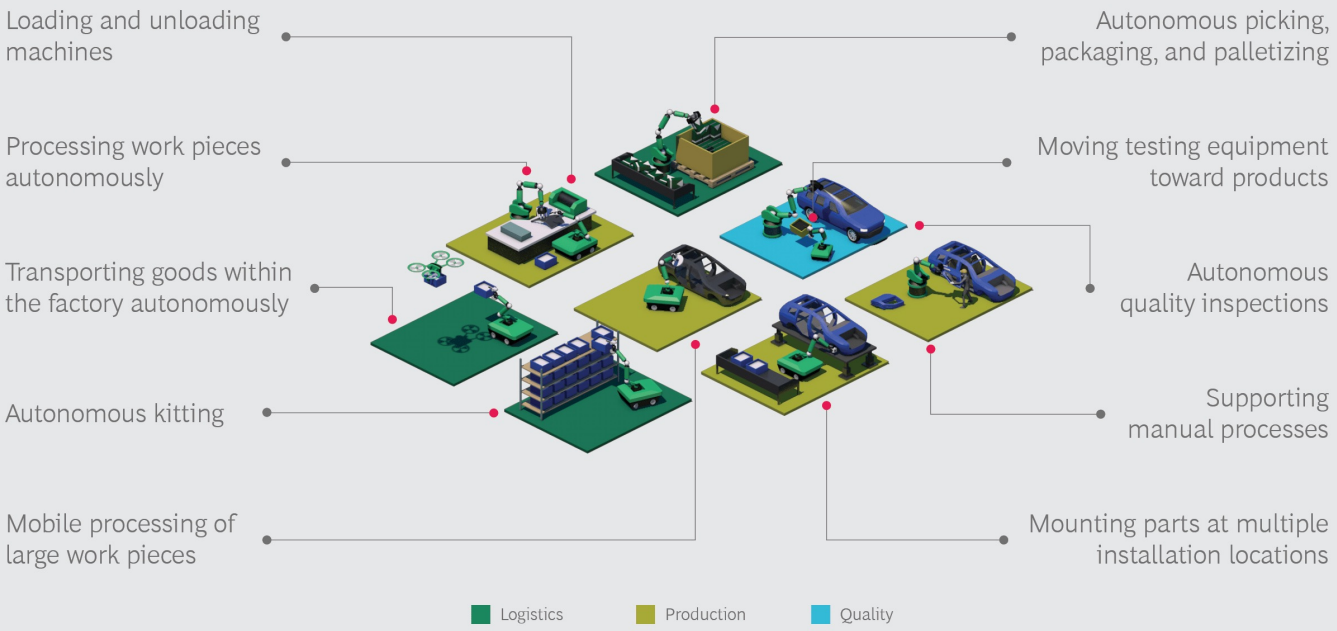
https://exoskeletonreport.com

Cobots are still a niche



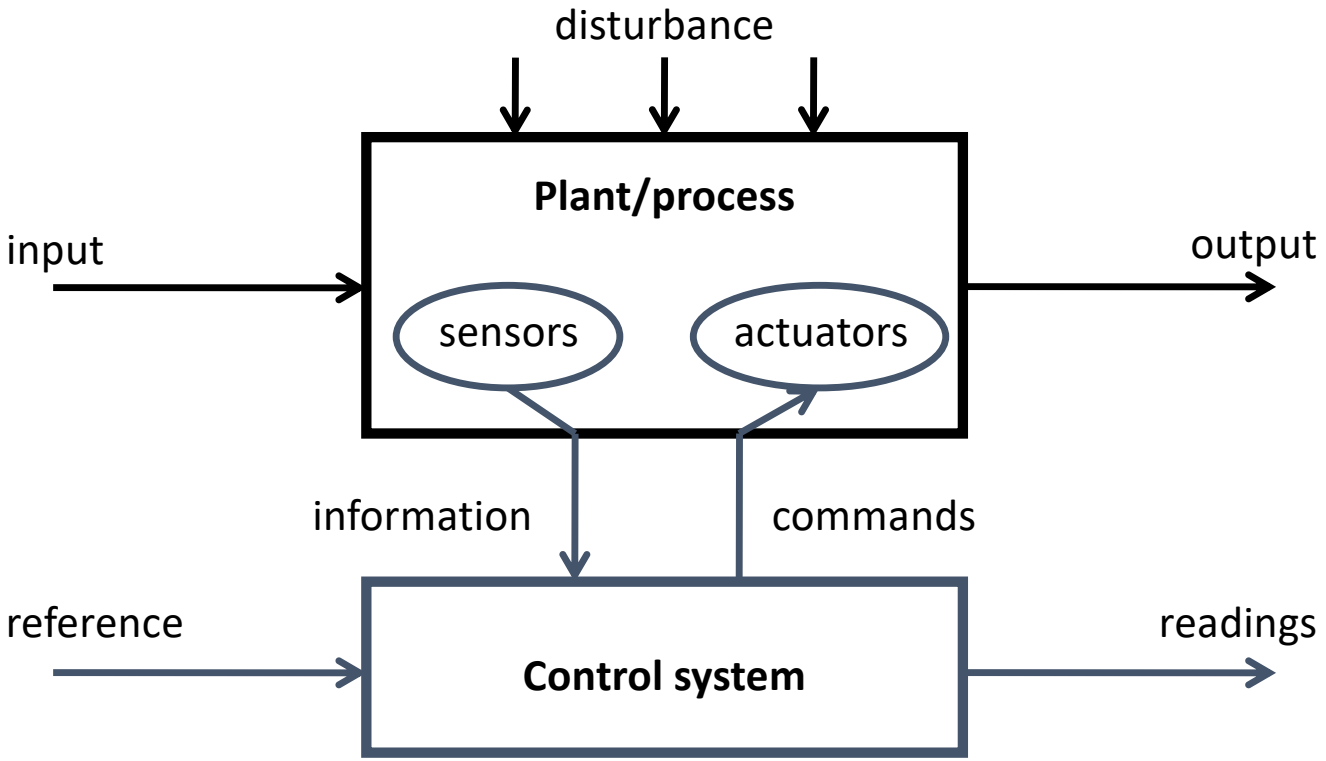
Applications in the Factory of the Future

EXHIBIT 3 | Advanced Robotics Has Many Applications in the Factory of the Future

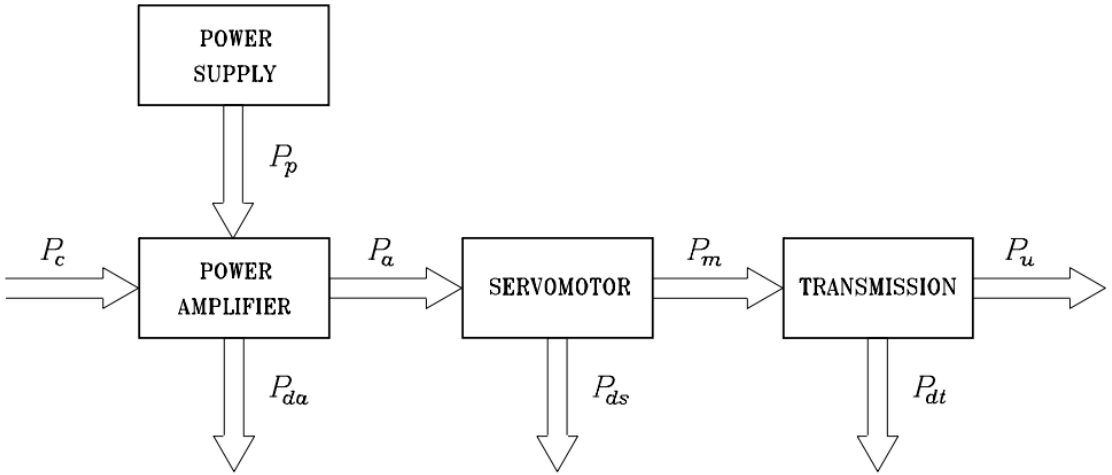


Sources: BCG Global Advanced Robotics Survey, January–February 2019; BCG analysis.

Have a look again

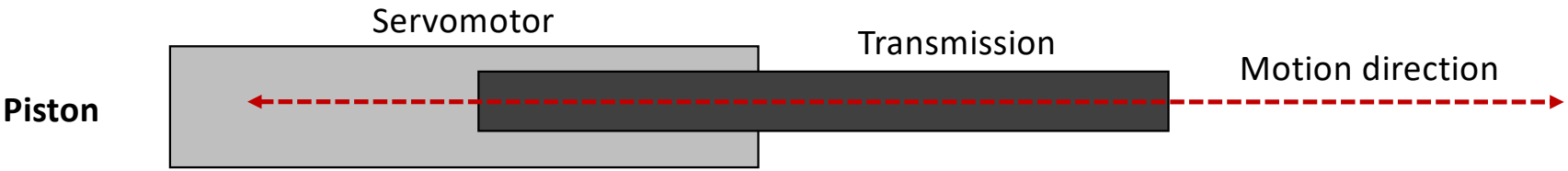


Actuators: General scheme

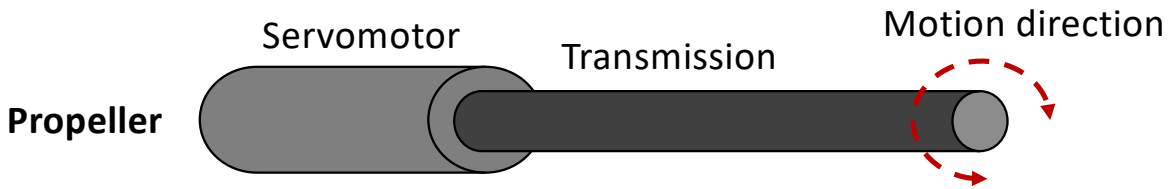


Actuators: Acutation systems

- Linear



- Rotational



Actuators: Servomotors

- Air Motors
  - Pneumatic energy served by a compressor, transformed into mechanical energy by means of pistons or air turbines
- Hydraulic Motors
  - Hydraulic energy stored in a storage tank by means of appropriate pumps, converted into mechanical energy
- Electric Motors
  - Electricity from the distribution network

- Motors for industrial robots
  - Low resistance to rotation and high power-to-weight ratio
  - High possibility of overload and development of impulsive torques
  - Ability to develop accelerations
  - High speed variation range
  - High positioning accuracy (at least 1/1000 of a turn)
  - Ensure continuous rotation even at low speeds

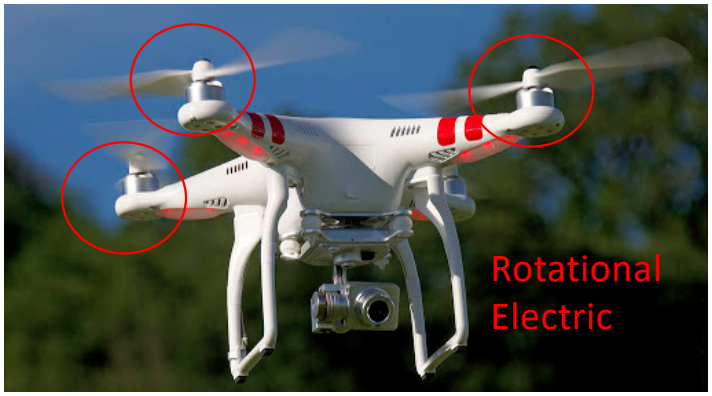
Actuators: Electric servomotors

- Pros
  - Widespread availability of the power source
  - Low cost and wide range of products
  - Good power conversion efficiency
  - Easy maintenance
  - Absence of pollution in the working environment
- Cons
  - Overheating problems, especially in static conditions due to the effect of gravity on the manipulator (can be remedied with parking brakes)
  - Need for special protection in flammable environments (ATEX)

Actuators: Hydraulic servomotors

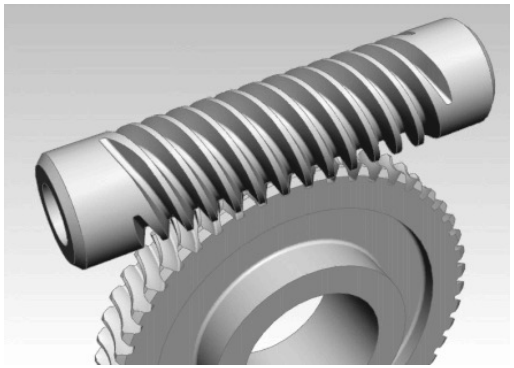
- Pros
  - No overheating problems in static conditions
  - They're self-lubricating
  - They are inherently stable in hazardous environments
  - Excellent power-to-weight ratio
- Cons
  - Requires a hydraulic power station
  - High cost, reduced product range (less competition), difficulty of miniaturization
  - Low power conversion efficiency
  - Need for periodic maintenance
  - Pollution of the working environment with oil leak

Actuators: Examples



Actuators: Transmission organs

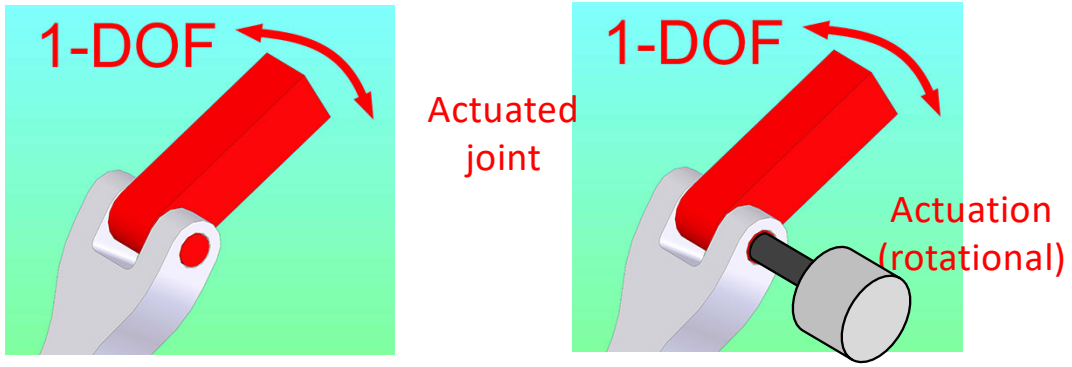
- Joints' motion
  - Low speed
  - High pairs
- Gear wheels
  - Varies the axis of rotation and/or translates the point of application
- Screw-nut
  - Convert the rotational motion into one of translation
  - Ball screws
- Toothed belts
  - Allocate the motor away from the axis of the implemented coupling
  - High speed but low forces (can deform)
- Chains
  - Allocate the motor away from the coupling implemented
  - Low speeds (suffer from vibrations)
- Direct coupling
  - Eliminate elasticity and clearance
  - Require more sophisticated control algorithms

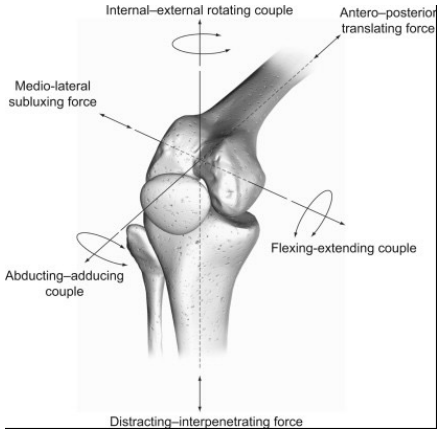


Actuators: Joints

- The joint is a device that join two extremities of two links such that the motion can be transferred from a link to the other
  - They extend the transmission concept
  - They can be *actuated* or *unactuated*

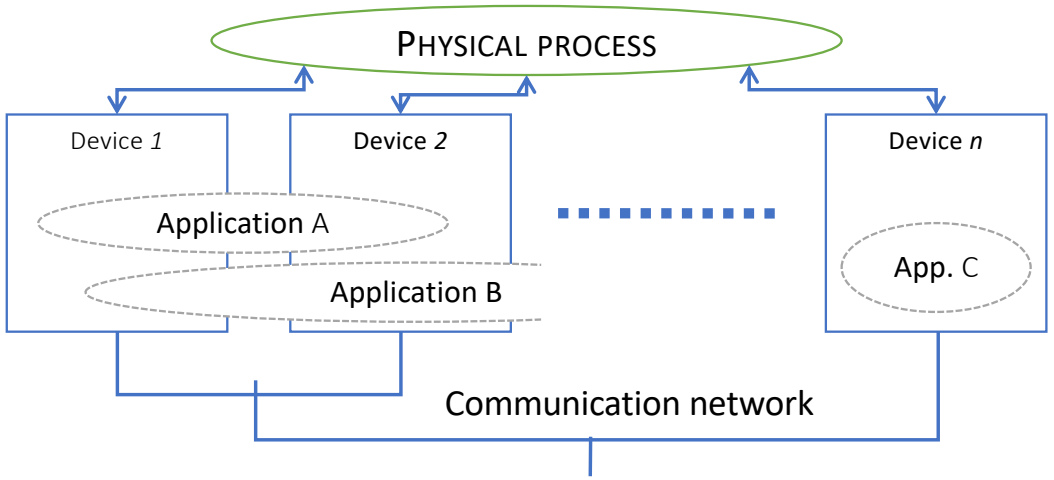
- The system *Degrees-Of-Freedom* – DOFs are the number of independent variables necessary to univocally determine the system position in the space





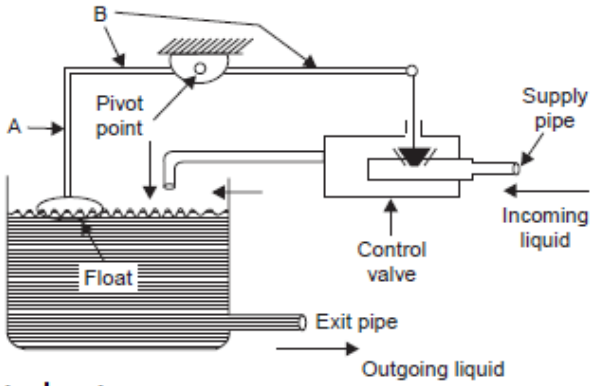
Control devices: Industrial process control system

- A definition is included in a proposed international standard for distributed automation
- *Definition*: Set of interconnected and communicating devices through one or more communication networks
- Communication networks can be organized according to hierarchical relationships



Control devices: Introduction

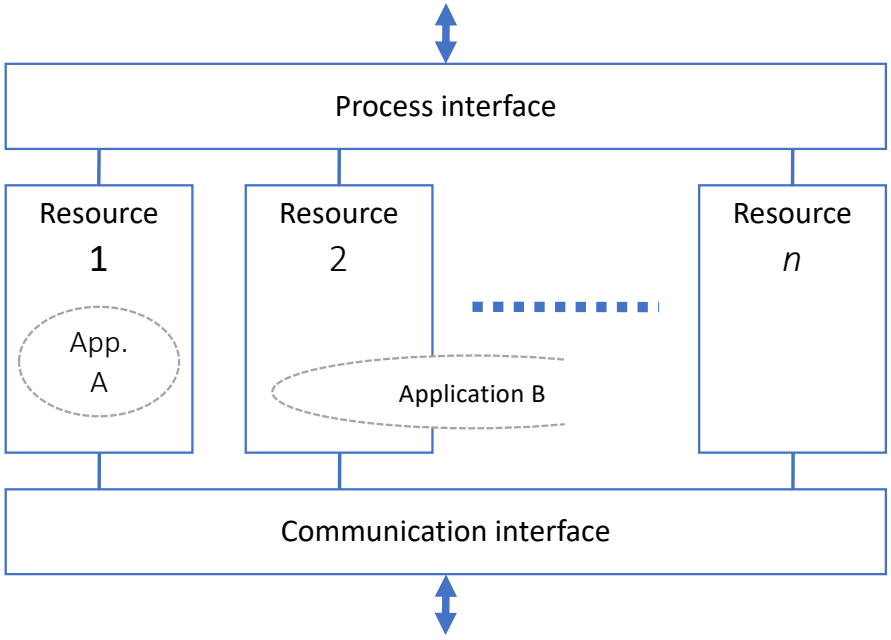
- The *functions* performed by the control system are modeled as applications
- An *application* can reside in a single device or be distributed among several devices
- *Example:* closing a control loop, where the acquisition of the measurements is carried out by one device, the control algorithm by another and the sending of the commands for control by another



Closed loop control system

Control devices: Device

- *Definition:* An independent physical entity capable of performing one or more applications, or parts of applications
- A device is limited by its interfaces, i.e., the hardware and software components that allow it to communicate with the outside world
- One device must contain at least:
  - one resource
  - an interface to the process (process interface) and/or to the communication network (communication interface)

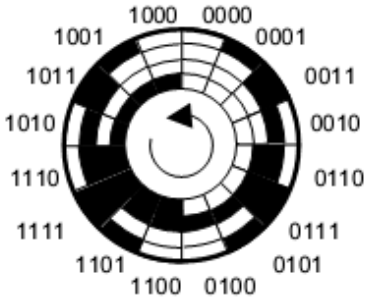


Control devices: Resource

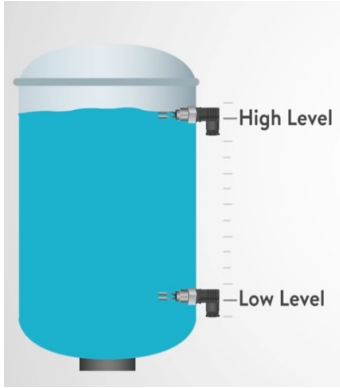
- *Definition:* logical subdivision of the software structure (possibly hardware) of a device that has independent control over its operations
- A resource can be created, configured, parameterized, started and deleted without affecting other resources
- This definition includes multitasking processing on systems with one or more processors
- Functions performed by a resource:
  - Accept data and/or events from the process and/or communication network
  - Process data received
  - Return data and/or events to the process and/or communication network
- Note: The functions performed by a resource must be performed as specified by the application using it
- In an asset must be present:
- One or more local applications, or parts of distributed applications, that process internal data and events
  - Functions that connect data and events from and to the process and/or communication network with the internal ones
  - An activity planning function (e.g. cyclic) for coordination between these functionalities

Control devices: Data and Event

- Data
  - Data are representations of facts or concepts expressed in a formalised manner suitable for communication, interpretation or processing by resources
  - *Example*: binary code corresponding to a measurement
- Events
  - The events represent the occurrence of particular conditions
  - *Examples*: reaching a certain fluid level



Gray code



Control devices: Application

- Specifies the operations that must be performed on the data as a result of the events
- Can be distributed among many resources in the same or different devices
- Specifies the causal relationships with which a resource determines responses to internal, process or communication events
- *Examples:* execution of operations, operation planning, modification of variables, generation of additional events, interactions with process and communication interfaces

Control devices: Process interface

- It relates the resources contained in the physical device to the physical process, communicating with the sensors and actuators
- It consists of a set of hardware devices and the software for their management (drivers)
- *Examples*: analog input/output cards, digital input/output cards
- The information exchanged with the physical process is presented to resources as process data and/or events associated with the process

Control devices: Communication interface

- It links resources to those belonging to other devices for the exchange of information through a communication network
- It consists of a set of hardware devices and the software for their management (drivers)
- *Examples*: network card, modem card
- Allows you to present information (data and events) to the resource and provide additional services (programming support, system configuration, diagnostics)

Control devices: Types of control and measurement systems

- Connected directly to machines to be controlled
- Requirements: real-time and multitasking operating systems, highly developed process interfaces, robust construction
- *Example*: PLC (Programmable Logic Controller)
- Man Machine Interface
- Requirements: good graphics skills, skills developed for communication with other devices, do not need a process interface: (example: personal computer)
- Control of complex machines
- Example: industrial robot manipulators
- Database management

Control devices: Basic requirements for a control device

- Ability to respond to stimuli coming from outside in the form of events or data coming from sensors
- Ability to act externally by modifying the behavior of the physical process they control

Definition: A control device is an information processing system intended for the direct control of physical processes

Control devices: Functionalities of a control device

- Closed-loop control of physical variables
- Adjusting and servoing
- Calculation of reference values for these variables
- Logical/sequential control
- Alarm and fault management
- Operator Interface
- Communication with other devices
- Treatment of input/output signals of different nature (even in large numbers)

Control devices: Methods of carrying out tasks

- *Periodically*, at assigned time intervals
- *Cyclically*, as soon as the execution is over, you have to start all over again
- *Only once*, when special events occur

- Not all features are always present
- The functionality actually implemented depends on the complexity of the device, i.e., the application for which it is intended
- *Example*: Microwave oven vs. robot manipulator controller

Control devices: Real time

- Interaction with the physical world must take place in real time
- Definition: A real-time processing system must respond in a certain way and within a fixed time frame to unforeseeable external events
- Delays in the execution of a process are considered malfunctions in real-time systems
- *Examples*: all direct control processes
- The real time requirement, from a hardware point of view, requires:
  - Use of one or more processors with adequate processing speed duration of the execution of tasks within limits
  - Known instruction execution time (at least maximum values)
  - Fast, reliable and deterministic access to memory and input/output devices
  - Guarantee of a certain reference timing
  - Presence of self-diagnosis functions
  - Presence of structural redundancies possibility to continue to operate even in the presence of malfunctions
- Control applications can be:
  - Simple → absence of an operating system, everything is entrusted to the program written by the user
  - Complex → presence of an operating system (albeit simple) that deals with the planning of the execution of processes (scheduling) and the management of communication between processes

Control devices: Real-time operating systems

- Presence of a prioritisation mechanism→ processes with an assigned priority cannot be interrupted by processes with a lower priority
  - *Multitasking pre-emptive* → possibility to interrupt any process to switch resources to a process that needs them the most
  - Absence of *deadlock* situations between processes → presence of a priority acquisition mechanism
  - In the event that a high priority process is blocked because it is waiting for a resource acquired by a low priority process, while yet another process is running, the low priority process must acquire the priority of the higher priority process that it is blocking itself
  - Mechanism of *synchronization* and communication between processes → predictable exchange of data at certain times
  - Known times (maximum values) needed by the system to suspend a process, launch another process and make a system call
  - Management of malfunctioning events in a non-abrupt manner
  - *Example*: Preventing a division by zero from necessarily interrupting the execution of a process
- Scalability: possibility to choose the operating system features necessary for the application to avoid unnecessary loading of the control device
- Input/output signal management
- Analog and digital signals
- Device equipped with special hardware interfaces and drivers to drive them
- Same considerations for communication interfaces

Control devices: Important requirements

- Device capable of operating in hostile environments (dirt, vibration, electromagnetic interference)
  - Solidity and robustness of construction
  - Shielding from electromagnetic fields

Control devices: Physical architecture

- Very varied, from a single chip, containing everything necessary to realize a control, to a bus system, where several specialized modules reside

- Control device architectures for generic applications
- Monolithic: single chip or single integrated circuit
- Based on bus combinations of modules with different functionalities interconnected via a common bus
- Based on personal computers

Control devices: Monolithic controllers

- Incorporates everything you need to realize control functions in a single board or integrated circuit
- With reference to a generic application, a monolithic controller must have:
- A processing unit that executes user and operating system programs
- A non-volatile memory that contains the programs
- Monolithic controllers have no mass memory
- A (volatile) memory for storing variables
- A clock → timing signals necessary for control
- Interface devices for the acquisition and generation of digital and analog signals
- Samplers, D/A and A/D converters, etc.
- An interrupt management system

Control devices: Microcontrollers

- *System-on-chip*: all components integrating a single chip
- Specific circuit solution → instructions dedicated to bit handling, input and output management, fast and efficient interrupt handling
- Widespread presence in a large number of devices
- *Examples*: household appliances, computers and peripherals, automobiles, etc.
- Suitable for control applications where the processing power required is not high and there are a small number of inputs and outputs to manage

Control devices: Microcontrollers

- Available in different sizes and varieties
- Most popular 8-bit microcontrollers
- There are still 4-bit controllers
- Market for 16 and 32-bit controllers on the rise
- Also characterized according to
  - Integration technology
  - Type of architecture
  - Memory type and size
  - Consumption
  - Number and type of interrupt management
  - Number and type of inputs and outputs

Control devices: Microcontrollers

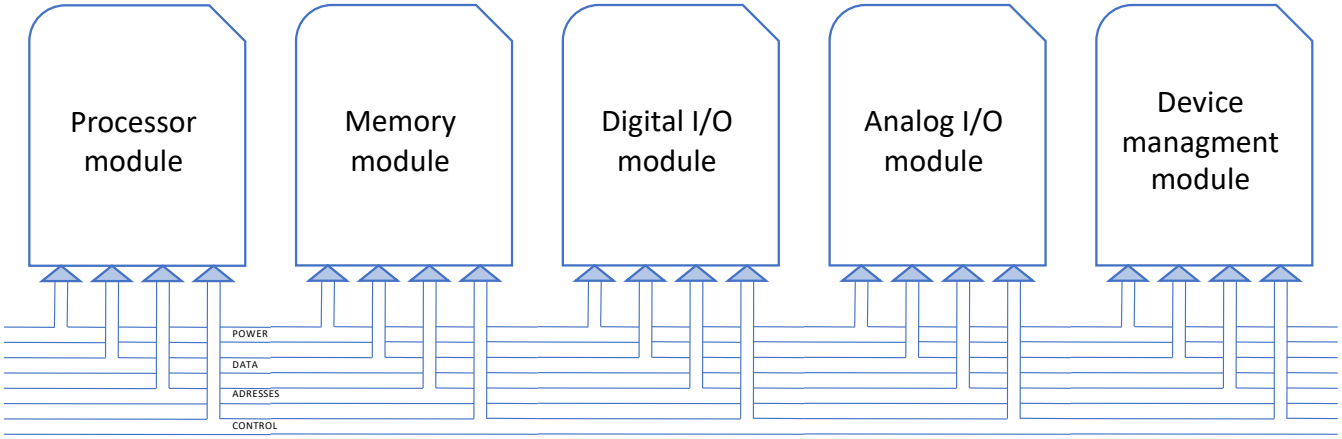
- Operating System
  - Normally not foreseen
  - User must take charge, by writing the program, of the management of the resources
  - Where necessary, scalable operating systems are used, from which only the process execution planner (scheduler) and the inter-process communication manager can be used.
- Development system
  - Based on personal computer
  - Programming in assembly languages or compilers for high-level languages like C (cross-compiler)
  - Availability of emulators to run programs, also step by step, which can be connected directly to the process with a probe

Control devices: Microcontrollers

- Operating System
  - Normally not foreseen
  - User must take charge, by writing the program, of the management of the resources
  - Where necessary, scalable operating systems are used, from which only the process execution planner (scheduler) and the inter-process communication manager can be used.
- Development system
  - Based on personal computer
  - Programming in assembly languages or compilers for high-level languages like C (cross-compiler)
  - Availability of emulators to run programs, also step by step, which can be connected directly to the process with a probe

Control devices: Controllers with bus architecture

- High processing capacity
- Possibility of processing a large number of inputs and outputs
- Ability to communicate via computer networks
- Possibility to realize very sophisticated operator interfaces



Control devices: What is a bus?

- A *bus* is defined as the combination of three concepts:
  - A set of electric lines, grouped by functions that connect various boards or modules together
  - A protocol through which modules can use power lines to communicate with each other
  - A set of electrical and mechanical characteristics of the connectors used to connect the modules together
  - Boards connected to the bus in parallel → the speed of propagation of electrical signals, compared to the frequency with which they vary, limits the bus length to a few decimeters

Control devices: What is a bus?

- Typical connections made by a bus are:
  - The address lines: each card will decode its own address
  - Data lines (in some cases multiplexed with address lines)
  - The control lines: handshake, clock, interrupt, bus arbitrage
  - The supply lines
  - The lines available to the user

- DMA (Direct Memory Access)
  - Access to memory for transferring blocks of data without using the processor

- Bus choice in multiprocessor systems
  - Always the first card, rotating or priority card

- Standard bus → availability of modules from different manufacturers that can be connected together to create the desired architecture

Control devices: What is a bus?

- The *features* of a bus are the following:
  - The electrical and mechanical data
  - Signal levels, physical conductor arrangement, connectors
  - The data transmission speed
  - Addressable space
  - Data length
  - The number of interrupt lines and how they are handled
  - Synchronous bus (need a clock) or asynchronous bus (no clock, more flexible need synchronization signals)
  - The number of master units on the bus that are permitted
  - Processor type constraints
  - Auto-configuration of the bus when inserting new modules

Control devices: General application controllers with bus architecture

- Bus architectures are modular: possibility to select the best hardware modules for the application to be realized

- Typical modules:
  - Processor module (CISC, RISC, DSP)
  - Analog input and output modules
  - Digital input and output modules
  - High speed counter modules
  - Solid State Mass Storage Modules
  - Modules for computer network communication
  - Modules for controlling peripheral devices (screens, keyboards, etc.)

- Rack: contain the modules mechanically and connect them electrically with a printed circuit with connectors

Control devices: General application controllers with bus architecture

- Operating System
  - More complex system -> you need an operating system to manage processing processes and system resources

- Development system
  - The use of high-level programming languages, such as C
  - There are development systems that allow programming directly into one of five automation-specific programming languages, such as Ladder Diagram and SFC

Control devices: Personal computer-based general controllers

- Using "general purpose" personal computers as control devices
- Advantages over control devices:
  - Lower cost
  - Less professionalism needed to use them
  - Wide variety of hardware suppliers
  - Choice of software suppliers
  - Simplified maintenance
  - Very extensive basic functionality (e.g. graphical user interface, mass storage, network interfaces, etc.).

Control devices: Personal computer-based general controllers

- Disadvantages compared to control devices:
  - Limited process interface (few cards can be installed)
  - Not robust : not suitable for a use in hostile environments, such as those in which the processes to be controlled are located

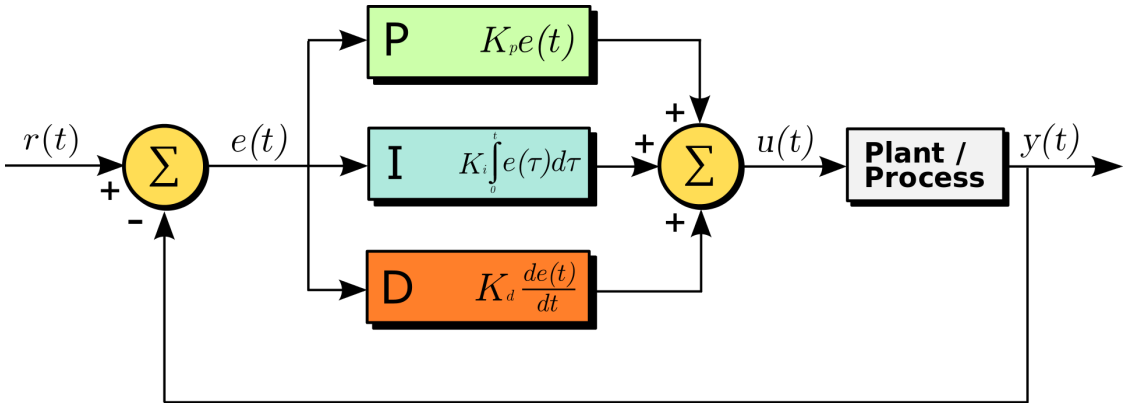
- Disadvantages that can be overcome with the use of computer networks
  - The computer can be placed away from the process
  - A large number of remote I/O (Input/Output) cards can be connected.
  - penalization of input detection and output update times

- Real-time operating system
  - Necessary, given the complexity of the hardware
  - There are native real-time systems, or special extensions, based on UNIX and Windows

- Development system
  - Programming with high-level languages
  - There are specialized controller emulators on PC platform (example: Soft-PLC, Soft-CNC)

Control devices: PID regulators

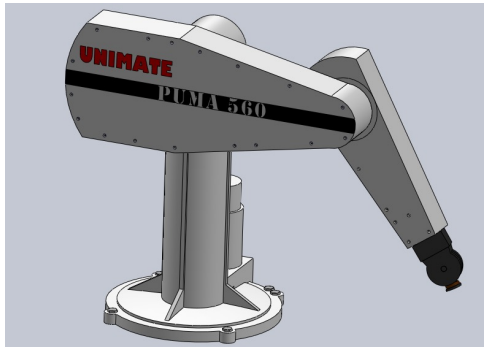
- PID regulator → proportional-integral-derivative
- They are the most widely used in various areas to maintain physical variables at constant values (regulation)
- Equation of operation in continuous time: $u(t) = K_P e(t) + K_I \int_0^t e(\tau)\mathrm{d}\tau + K_D \frac{\mathrm{d}e(t)}{\mathrm{d}t}$



Control devices: PID regulators

- Where *u* is the control signal, *e* is the difference (error) between the reference value and the actual value of the variable to be checked
  - *First term :* proportional, defined by K_P *gain*
  - *Second term :* integral, defined by K_I gain
  - *Third term* : *derivative*, defined by K_D gain

  - *Example*: Robotic arm
    - An electric motor may lift or lower the arm, depending on forward or reverse power applied, but power cannot be a simple function of position because of the inertial mass of the arm, forces due to gravity, external forces on the arm such as a load to lift or work to be done on an external object



Control devices: PID regulators

- Example: Robotic arm
  - *Proportional control*: the motor current is set in proportion to the existing error. However, this method fails if, for instance, the arm has to lift different weights
  - *Integral control*: increases action in relation not only to the error but also the time for which it has persisted. So, if applied force is not enough to bring the error to zero, this force will be increased as time passes
  - A *derivative term* does not consider the error (meaning it cannot bring it to zero), but the rate of change of error, trying to bring this rate to zero. It aims at flattening the error trajectory into a horizontal line, damping the force applied, and so reduces overshoot (error on the other side because too great applied force)

Control devices: PID regulators

- PID control is used when the realization of the plant differs from the ideal one represented by the ideal equation

- Only the controller configuration parameters (K_P, K_D, K_I) must be assigned (tuning)

In some applications, self-determination of the parameters of the algorithm is foreseen (self-tuning)

- Dedicated controllers available in monolithic form

Connect directly to sensors and actuators on the process

- Communication
  - In some cases it is possible to communicate over a computer network with other devices
  - Communication of controller status
  - Receiving a new reference

Control devices: Program Logic Counter-PLC

- It can be considered a control device for generic applications, but is characterized by a high degree of specialization -> logic/sequential control
- It is the most widespread control device in industry, but not only

Modular device with bus architecture

- Possibility to treat up to thousands of input/output points of various nature (analog/digital, voltage/current, etc.)
- Equipped with proprietary real-time multi-tasking operating systems
- They are devices of very robust construction suitable for applications in industrial environments (vibration, dirt, electromagnetic disturbances)

Control devices: Program Logic Counter-PLC

Historical notes

- The technical objective has always been the automatic execution of machining processes
- Before the discovery of electricity -> mechanical controllers (e.g. watt speed governor)
- The advent of electronics (transistors, integrated circuits) solved many of the problems of electromechanical systems
- One essential feature was still missing: flexibility
- Possibility to be quickly adapted to perform new functions
- Advent of the electronic calculator
- The characteristic of being a programmable system also revolutionized the industrial automation sector.

Control devices: Program Logic Counter-PLC

Historical notes

- In 1968 the first programmable logic controller was born
- General Motors specified the desired features for a new generation of controllers for their systems:
- Ease of programming and reprogramming, even directly at the place of operation
- Easy maintenance -> modular devices
- Robustness for operation in industrial environments (electromagnetic interference, dust, vibration)
- Compactness (small size compared to previous systems)
- Reduced costs
- Secondary specifications were also required, including:
- Possibility to expand the memory
- Communication with data logging devices
- Possibility to accept medium alternating voltage signals
- The result was the development of the first generation of PLCs

Control devices: Program Logic Counter-PLC

Historical notes

- Allen Bradley introduced the first microprocessor-based PLC (it was 8080) in the mid-1970s
- Since then, the evolution of PLCs has continued parallel to that of classical computer science
- The current high-class PLCs are based on multiprocessor systems
- They have the possibility to connect to computer networks
- They have the ability to perform very complex functions
- Modern PLCs have the characteristics of conventional computers but are suitable for their main use, industrial process control

Control devices: Program Logic Counter-PLC

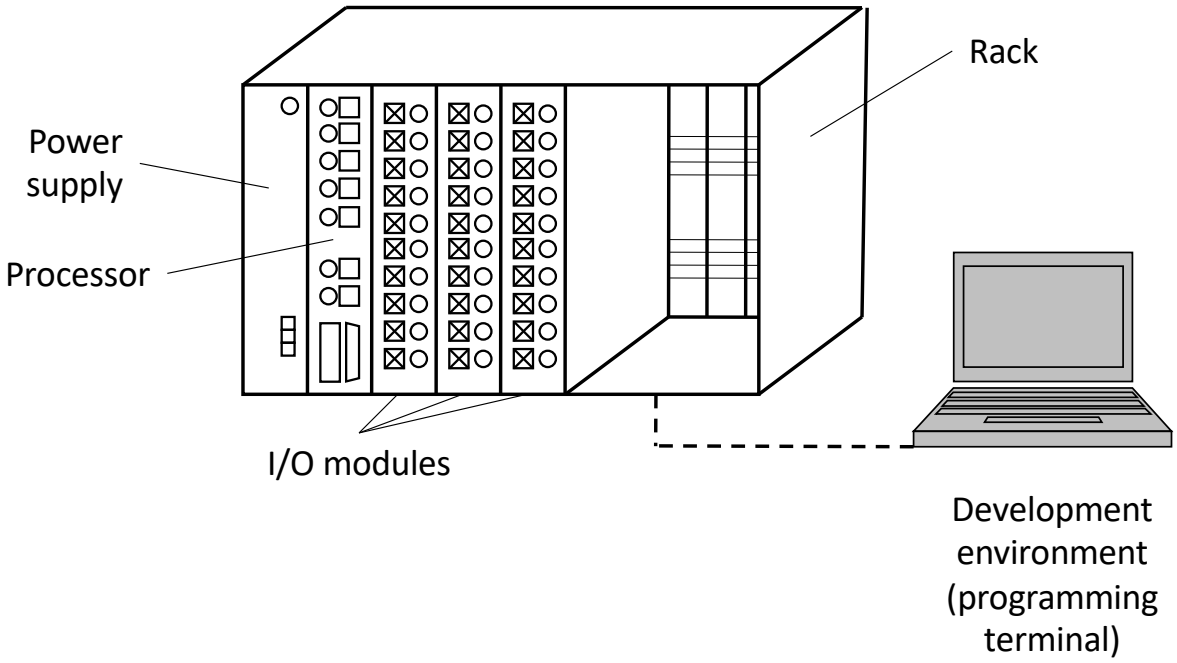
- Standard 61131-1 CEI (International Electronics Committee)

- PLC: electronic system with digital operation, intended for use in industrial environments, using a programmable memory for the internal storage of user-oriented instructions for the implementation of specific functions, such as logic, sequencing, timing, counting and arithmetic calculations, and for controlling, by means of both digital and analogue inputs and outputs, various types of machines and processes

- PLC system: configuration made by the user, consisting of a programmable logic controller (PLC) and associated peripherals, necessary for the automation of the planned system

Control devices: Program Logic Counter-PLC

- Minimum configuration of a PLC

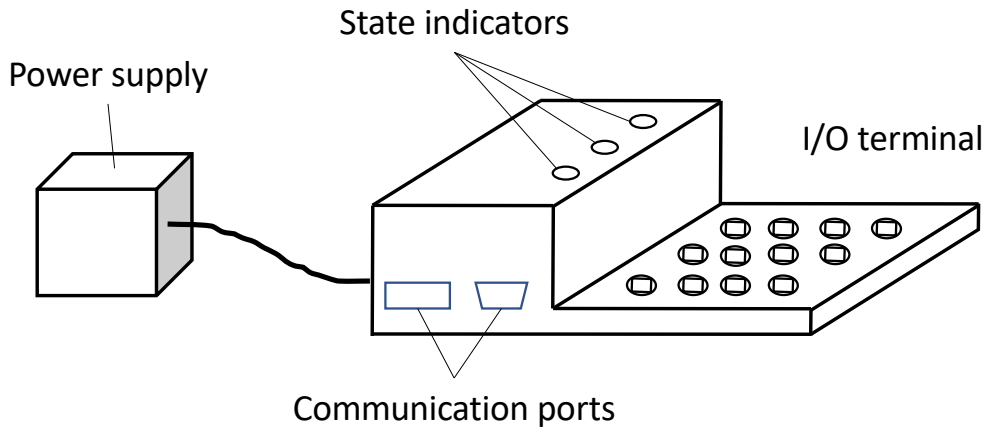


Control devices: Program Logic Counter-PLC

- Rack: contains and encloses all other modules, ensuring their mechanical connection and electrical connection
  - Form of a parallelepiped, open on one side to allow the insertion of modules that are electrically connected to each other thanks to the presence on the opposite side of a printed circuit board with connectors
- Processor module: is the real PLC and is essentially made up of a microprocessor board with an architecture similar to that of conventional computers
  - Controls and supervises all operations performed within the system
  - Executes the instructions in the memory
- Input/output modules (I/O): are boards that allow the interface between the microelectronics of the PLC and the outside world
  - Provide signal conditioning and isolation
- Power supply module: is a card that powers all the other modules in the cabinet
  - Connected to the power grid, it provides one or more stabilized voltages
- Programming terminal: It is typically a personal computer and is used for PLC programming
  - It is connected to the PLC via network or serial connection only during programming

Control devices: Program Logic Counter-PLC

- There are small PLCs of non-modular type
- They are monolithic controllers that do not include the rack
- They enclose all or some of the elements described above in a single tab



Control devices: Program Logic Counter-PLC

Control devices: Program Logic Counter-PLC

- Processor module
  - Encloses a card with one or more processors that run the operating system and user programs, and a memory where these programs are stored
  - Microprocessors are used as the central processing unit
    - Some are specially designed to allow direct interpretation of single bit instructions, which are very common in logic/sequential control
  - Some realizations involve three microprocessors:
    - One specialized to operate on the single bit
    - One for arithmetic/logical instructions
    - One dedicated to communications with input/output modules and external devices
  - The most common mode of operation is cyclical
    - Updating the specially reserved memory area with the values coming from the physical inputs
    - Running the user program(s)
    - Execution of system management programmes
    - Diagnostics
    - Writing on the physical outputs of their values stored in the specially reserved memory area

Control devices: Program Logic Counter-PLC

- Processor module

  - The described cycle is also called "massive copy cycle of inputs and outputs"
  - Optimizes communication with input/output modules
  - Ensures that the stored values of the inputs remain unchanged during program execution consistency
  - The reading of the inputs and the writing of the outputs is completely managed by the operating system, which is not the user's responsibility
  - The system remains "blind" until the next reading
  - Delay in detection of input change at most equal to cycle time
  - If a digital input changes twice during a scan cycle, the change is not and will never be detected again
  - Unspecified cycle time depends on the length and nature of the PLC program that is not suitable for closing control loops

Control devices: Program Logic Counter-PLC

- Scan time (T_s)

  - Describes the processing speed of the processor module
  - *Definition*: time elapsing between two successive activations of the same portion of the application program in cyclic operation mode
  - The time required to run the operation cycle depends on the number of inputs and outputs to be updated and the size and complexity of the user program
  - It is indicated by the manufacturer as an average value for programs of medium complexity
  - Varies from a few units to a few tens of milliseconds per Kiloword (1024 words) of program depending on PLC class

Control devices: Program Logic Counter-PLC

- Response time (T_r)
  - *Definition*: maximum time interval between the detection of a certain event and the execution of the scheduled response action
  - Also takes into account delays introduced by input/output modules
  - Relationship between scan time and response time, referring to the worst-case scenario:

$$T_r = 2T_s + T_{i/o}$$

  - where T_r is the response time, T_s is the scan time, $T_{i/o}$ is the delay introduced by the input/output modules

Control devices: Program Logic Counter-PLC

- Operating System
  - Set of supervision programs permanently stored in the PLC, dedicated to:
  - Control of activities
  - Processing user programs
  - Communication
  - Other functions (e.g. watchdog timer - supervisor timing and serves to detect any infinite cycles or stall situations - parity checks on memory and communication lines, power supply voltage control, buffer battery status check, etc.)

Note: the control functions can activate status indicators and start user programmable emergency routines

Control devices: Program Logic Counter-PLC

- Operating modes of a PLC
  - Activated through a hardware key:
    - *Execution*
    - *Validate*
    - *Programming*

  - Execution mode
    - User programs are executed
    - Inputs and outputs are updated
  - Validation mode
    - User programs are executed
    - Output update disabled
  - Programming mode
    - Allows you to load the developed program into memory

Control devices: Program Logic Counter-PLC

- Methods of the programs' execution
  - Even today there are PLCs in which programs are *interpreted*
  - Each instruction is converted to machine language immediately before it is executed
  - The diffusion of architectures based on conventional processors and the use of personal computers as programming terminals has led to the diffusion of PLCs for which a *compilation* phase is foreseen before the program execution
- Memory
  - It can typically be divided into 5 macro areas:
    - Operating system area
    - Reserved for permanent storage of the operating system
    - Non-volatile type
    - Working area of the operating system
    - Reserved for intermediate data storage by operating system programs
  - It must allow data to be read and written (RAM - Random Access Memory - volatile memory)
  - Input/output area
  - Reserved for storing input and output states

Control devices: Program Logic Counter-PLC

- User area
  - Reserved for storing user programs
  - RAM memory during programming, which can be replaced with a PROM memory once the program has been created and verified
  - User data area
  - Reserved for the storage of user program data

- RAM type memory
  - RAM type areas can be partially or all powered by buffer batteries to avoid data loss in case of power failure
  - CMOS technology to limit consumption
  - In case of power failure, the operating system and the user programs can manage the reset mode thanks to special status bits

Control devices: Program Logic Counter-PLC

- Number and quality of communication ports available
  - Serial, parallel, network
  - Type of languages supported and complexity of the set of instructions
  - Multitasking possibilities
  - Possibility to manage interrupt routines

- Safety PLC
  - They are designed to be used in applications requiring very high levels of safety (e.g. press automation)
  - They foresee a redundancy of processing units
  - They run the same program and enable the outputs only if there is full agreement between them

Control devices: Program Logic Counter-PLC

- Input/output modules
  - They are the modules with which the PLC communicates with the physical process
  - Detect events and data from sensors
  - Command actions to actuators
- Modularity and variety
  - PLC systems tailored to the application
- Addressing
  - Depends on where the modules are located in the PLC cabinet or cabinets
  - Mechanical precautions are provided to prevent a module from being installed in positions other than those provided for
- Digital Input Modules
  - Equipped with filtering circuits against noise and bounce introduce delays of the order of a few milliseconds in input detection
  - Provided with status indicators before and after isolation circuits
  - Technical information on the number of manageable inputs, the reference voltages, the signaling delay introduced is shown

Control devices: Program Logic Counter-PLC

- Digital output modules
  - Outputs normally protected by fuses
- Analog input/output modules
  - Perform the digital/analog or analog/digital conversions required to directly interface analog signals with the PLC
  - They handle a wide range of signals
  - Programmable working ranges and filter characteristics
  - Automatic scaling in engineering units of the data
  - They have status indicators
  - Technical values reported by manufacturers:
    - Values of the treated signals
    - Possibility to accept single-ended or differential inputs
    - The conversion resolution
    - The data representation provided
    - The conversion speed
  - Some output modules can assume a fixed value if no new information is received from the PLC within a fixed time
  - Considering their diffusion and relative management complexity, there are specific modules for the direct use of metal resistive temperature sensors (RTD) and thermocouples

Control devices: Program Logic Counter-PLC

- Power supply module
  - Provides through the cabinet the stabilized power supply necessary for the operation of the other modules
  - It must ensure a constant voltage even in the presence of micro interruptions or fluctuations in the power supply
  - It consists of
    - A transformer
    - One rectifier circuit
    - A filter
    - A stabilizer circuit
    - One circuit for overcurrent or short circuit protection
  - Technical characteristics
    - Maximum deliverable power
    - Possibility of parallel connection to increase maximum power or realize a power redundancy
    - Possibility to send a shutdown signal to the PLC -> if the input power supply drops below limits, the PLC will start the programmed procedures before the actual shutdown
    - Presence of indicators on its status
  - Sizing
    - This should be done by taking into account the sum of the powers required by the forms provided for, increased by a certain percentage to take account of any future expansion

Control devices: Program Logic Counter-PLC

- Closet (Rack)
  - It must contain the modules of a PLC system and ensure electrical/mechanical connection and shielding
  - The electrical connection is made through the bus: a set of power lines, grouped by functions (address lines, data lines, control lines, power supply lines) and associated protocols, through which a module can communicate with others
  - These are proprietary buses: it is impossible to use modules from different manufacturers
  - Mechanical characteristics (number of slots, overall dimensions, fixing method, ...)

Control devices: Program Logic Counter-PLC

- Programming Terminal
  - The PLC does not provide keyboards and screens for communication with the programmer -> special devices are required for programming
  - Small PLCs still use keypad terminals
  - They connect directly to the PLC through a communication port (serial)
  - They present the operator with a keyboard and a small liquid crystal display
  - Programming directly into PLC memory
- Development systems based on personal computers
  - Off-line programming
  - They have very complex program composition features
  - Permanent storage capacity of developed programs
  - Are connected to the PLC directly or via a computer network
  - With just one personal computer you can program all the PLCs connected to the network
  - Monitoring functions are provided to monitor the execution of the program and the memory areas that can be executed even during the normal operation of the device

Control devices: Program Logic Counter-PLC

- Special modules
  - Remote I/O modules
    - Useful when the number of inputs and outputs is high and they are located on large plant areas
    - I/O cabinets are installed scattered in the system, connected to the PLC through a remote I/O module that sends the status of the inputs and outputs of the cabinet in which it is mounted through a serial line or a computer network
  - Network connection modules
    - They manage the communication protocols for the different types of computer networks that may involve the PLC
    - Fieldbus, proprietary networks, Ethernet
  - Coprocessor modules
    - They are modules containing a conventional computer that can directly access the PLC memory
    - Possibility to perform complex elaborations
    - Programming in high-level languages (C or Basic)
    - They may include a mass storage unit
  - PID modules
    - Allows you to have a few PID loops to which the PLC only provides references
    - Can be self-synchronized with neutral mode change

Control devices: Program Logic Counter-PLC

- Special modules
  - Servo modules
    - They realize directly and autonomously the servo-driving of one or more stepper motors, hydraulic motors, direct current motors with incremental encoders
  - Encoder modules
    - Provide the necessary functionality to use one or more incremental or absolute encoders
    - Contain high speed counters
  - Operator interface modules
    - They allow the interfacing of an operator with the PLC by means of a keyboard and an alphanumeric display
    - Nowadays not very widespread and replaced by common networked PCs
  - Backup Modules
    - They are modules that must be inserted in two separate PLCs and allow to have a backup functionality
    - Maximum availability system -> the backup processor module, through these modules, is informed about the status of the main processor module and can replace it in a very short time, in case of malfunction of the latter, in the management of inputs and outputs
    - Possibly, you can also have a duplication of the other modules that make up the system
    - Intrinsically safe system -> requires the two processor modules to agree on the status of the outputs before they are actually applied.
  - Barcode reader modules, vision systems, ...

Control devices: Program Logic Counter-PLC

- PLC classification
  - Micro PLC
    - They process up to 64 input/output points, generally digital
    - Memories of 1 or 2 Kilowords
    - They do not have a modular cabinet structure (monolithic systems)
    - They may provide for I/O expansions
    - Programmable with only one programming language
      - Contact language
      - Limited set of instructions, functions and function blocks
    - Normally used to replace relay logic in applications such as the control of operating machines, elevators, washing machines
  - Small PLC
    - They deal with 64 to 512 input/output points, in digital dominance
    - Memories up to 4 Kilowords
    - They have a modular cabinet structure
    - They can have networking and remote I/O management capabilities
    - Programmable with different programming languages
    - More complete set of instructions, functions and function blocks

Control devices: Program Logic Counter-PLC

- PLC classification
  - Medium PLC
    - They treat from 256 to 2048 input/output points
    - Memory up to a few dozen Kilowords
    - Modular cabinet structure
    - Manage remote I/U
    - They can be enriched with special modules
    - They have high communication capabilities in the computer network
    - Very advanced programming possibilities
  - Large PLCs
    - Several entry/exit points (several thousand)
    - Memory of hundreds of Kilowords
    - Considerable information processing capacity
    - Used as supervisors of automated cells and as interface between lower performance PLCs and management calculators

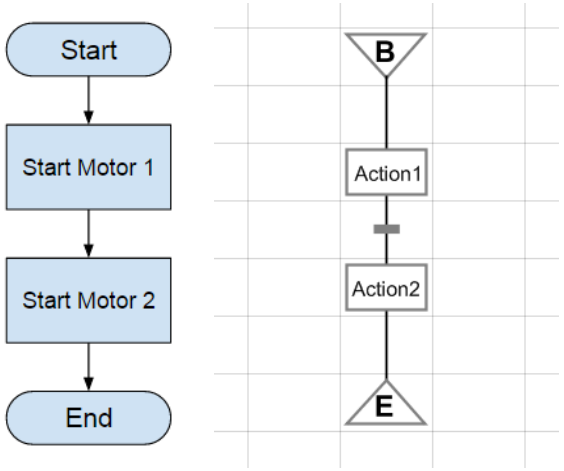
Programming: Languages

- Programming languages*
  - The IEC 61131 standard defines the standardization of programming languages for PLCs
  - The standard is a non-binding reference model
  - There are old PLCs still installed that do not meet the standard
  - The new ones don't always fully respect it
  - The most common language for PLC programming remains the contact language (ladder diagram)
  - There are different "dialects" for the various manufacturers
  - The instruction list is also very popular
  - Also in this case there are various dialects

  - *A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks

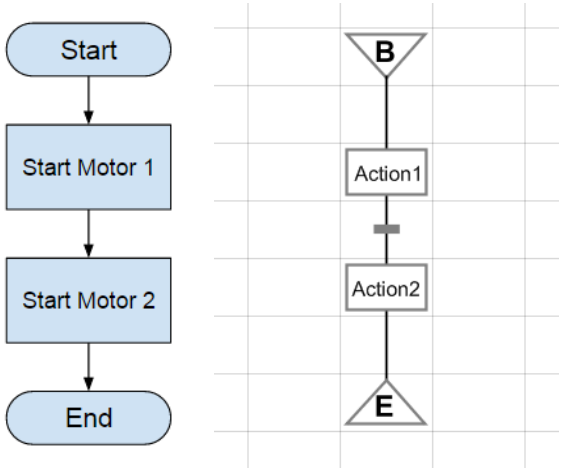
Programming: Languages

- Programming languages
  - Structured text and SFC (Sequential Functional Chart) are also spreading
  - Only a few PLCs, the most advanced, currently provide the possibility to be programmed directly with the SFC
  - SFC is the best language for programming logic/sequential control algorithms typical of industrial automation
  - Because of its inherently visual depiction, it helps to illuminate logic to users, and facilitate intuitive development and refinement. Charts can be monitored as they run visually, making troubleshooting easier than with scripting alone



Programming: Languages

- Programming languages
  - Structured text and SFC (Sequential Functional Chart) are also spreading
  - Only a few PLCs, the most advanced, currently provide the possibility to be programmed directly with the SFC
  - SFC is the best language for programming logic/sequential control algorithms typical of industrial automation
  - Because of its inherently visual depiction, it helps to illuminate logic to users, and facilitate intuitive development and refinement. Charts can be monitored as they run visually, making troubleshooting easier than with scripting alone

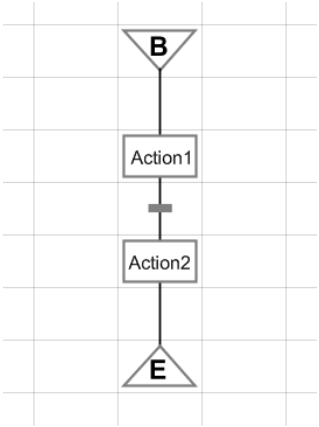


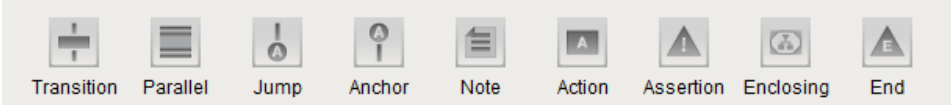
SFC: Introduction

- Innovative language suitable for writing logic/sequential control algorithms
- Born as a result of a special commission set up in 1975 in France
- Purpose: to look for a way to describe complex discrete event-driven industrial automation systems
- Discrete event system have a discrete state space, not continuous, and whose evolution depends on whether or not special conditions occur, the events
- Result: definition of the GRAPHe de Coordination Etapes-Transitions (GRAFCET)
- Adopted by the CEI in 1988 in the international standard 848 as a language for the description of industrial automation systems
- This is a simplification of Petri Networks, a more general graphical tool for the representation and analysis of discrete event systems
- The GRAFCET has been included under the name Sequential Functional Chart (SFC) among the programming languages of standard 61131-3

SFC: Introduction

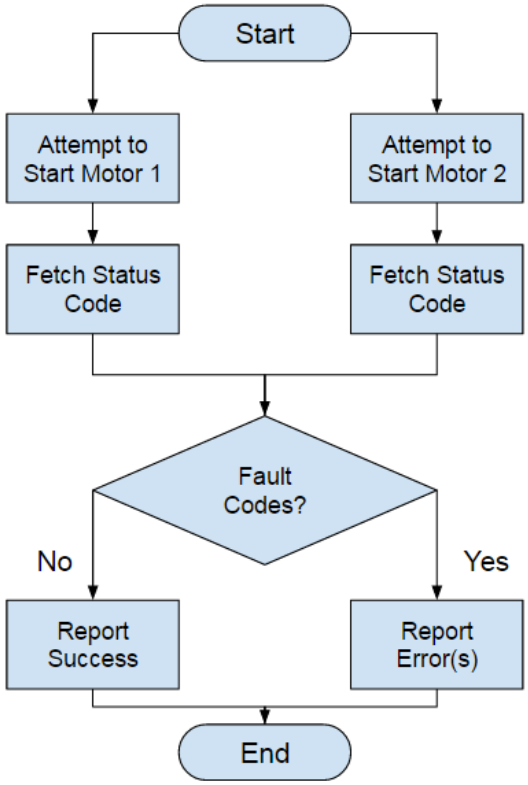
- It is not a language equal to others in characteristics and purposes
- Its main use is programming logic/sequential control algorithms
- Allows simple segmentation of control algorithms into phases and transitions between phases
- The behavior of an algorithm written in SFC depends on the previous state -> it can be used for programming function blocks and programs, not functions
- Basic elements of the Sequential Functional Chart:
  - Phase (or stage or step) with any associated actions
  - Transition with the associated condition
  - Oriented arc connecting phases and transitions





SFC: Introduction

- *Example*: Run two motors



SFC: Variables and type of variables

- They are the means by which it is possible to represent the data inside the control device
- The standard includes some predefined types of variables
- They can be directly used in the declaration of variables
- Definition in text form regardless of the programming language used
- Exception: in the definition of functions and function blocks, input and output variables can be defined in graphical form

SFC: Variables and type of variables

- *Whole numbers*
  - Type INT
    - Represents integers in the range[–32.768, 32.767]
  - Type UINT
    - Represents non-negative integers in the range[0, 2^{16}–1]
  - There are other whole types with different more or less extended intervals

- *Real numbers*
  - Type REAL
    - Represents real numbers in the range $\pm 10^{\pm 38}$
  - Type LREAL
    - Represents real numbers in a wider range with greater accuracy

SFC: Variables and type of variables

- *Time variables*
  - Type TIME
    - Represents a time duration in units ranging from days to milliseconds
    - Representation format T#αdβhγmδsεms to indicate a duration of α days, β hours, γ minutes, δ seconds and ε milliseconds
    - *Examples*:
      - T#2d5h30m to indicate a duration of 2 days, 5 hours e 30 minutes
      - T#5m100ms to indicate a duration of 5 minutes and 100 milliseconds
      - T#120s or T#2m to indicate a duration of 120 seconds
  - Types: DATE and TIME_OF_DAY
    - They represent the date and time, respectively
  - Type: DATE_AND_TIME
    - Represents date and time

SFC: Variables and type of variables

- *Strings of characters*
  - Type: STRING
    - It represents a string of characters

- *Strings of bit*
  - Type: BOOL
    - Represents a single bit or a logical variable
  - Type: BYTE
    - Represents a string of 8 bit
  - Type: WORD
    - Represents a string of 16 bit

SFC: Operators

- Arithmetic operators
  - \+ sum
  - \- difference
  - * product
  - / division
  - MOD rest of an entire division
  - ** power elevation

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

- Operators on the bit
  - AND (or &) for the logical product function
  - OR for the logical sum function
  - NOT for denial

SFC: Operators

- Arithmetic operators
  - \+ sum
  - \- difference
  - * product
  - / division
  - MOD rest of an entire division
  - ** power elevation

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

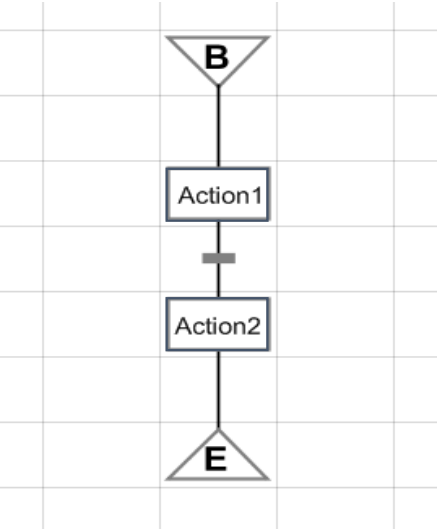
- Operators on the bit
  - AND (or &) for the logical product function
  - OR for the logical sum function
  - NOT for denial

SFC: Operators

- Relational operators
  - < minor
  - <= minor or equal
  - > major
  - >= major or equal
  - = equal
  - <> different

SFC: Phase

- The phase
  - *Definition*: invariant condition of the system, modifiable only by the occurrence of a certain event, which generates a transition that brings the system into a new phase
  - A phase can be, in a given moment of time, active or inactive
  - A condition of the SFC is defined as the set of active phases
  - If a phase is active, the behavior of the program (function block), described by the SFC, is defined by the actions associated with that phase
  - Graphic representation: a phase is represented by a rectangle inside which the name of the phase is written
  - The name must be unique in the context in which the phase is defined
  - The rectangle can be connected to other elements, the transitions, through connectors on the upper or lower edge



SFC: Phase

- *Initial phases*: these are the phases activated at the beginning of execution
  - They are identified graphically by two vertical lines placed in the rectangle
  - The standard requires each SFC graph to have only one initial stage
- A function block or program may consist of several unconnected SFC graphs
- Signaling variable or phase marker
  - The definition of a phase with the name name_phase implies the definition of the boolean variable (true or false, 0 or 1) name_phase.X associated with it, which will assume the logical values:
    - 1 in case of active phase
    - 0 in case of inactive phase
    - The value of this variable is available, for the graphical connection, on the right of the rectangle representing the phase
    - The signaling variable is initialized to 1 for the initial phases and to 0 for all other phases
    - To graphically indicate the active phases you can draw a dot within the phase

SFC: Phase

- Timer variable
  - The definition of a phase with the name name_phase implies the definition of the variable of type TIME from the name name_phase.T which represents the duration of the last phase activation:
    - Active phase → represents the time elapsed since its activation
    - Inactive phase → represents the duration of its last activation
  - This variable is initialized to the value T#0s for all phases
  - The signalling and timer variables cannot be modified by the user but only used by him
    - Must be read-only variables for the user
    - These are local variables to the programming unit in which the SFC is defined

SFC: Transition

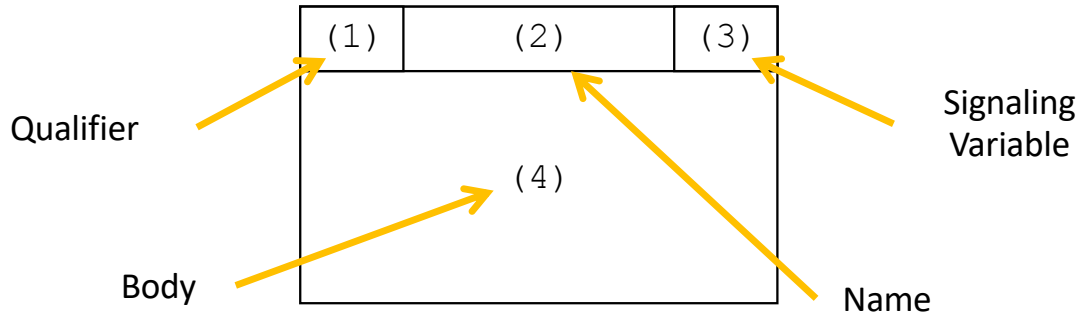
- The transition
  - It is indicated graphically with a crossbar placed on the oriented arch
  - It represents the condition that could change the state of the active phases
  - Each transition must be associated with its condition expressed as a Boolean function of Boolean variables, or equations that return Boolean values or assertions
  - The standard provides various possibilities for defining the conditions associated with transitions. We introduce only the following:
    - Expression in structured text placed to the right of the bar

SFC: Transition

- Oriented arcs
  - The orientated arcs connect the phases to each other, establishing their sequence
  - They are interrupted by the transition bars that determine the conditions to be met in order to have the activation and deactivation of the phases
  - The orientation of the arches always goes from the lower edge of one or more phases to the upper edge of one or more phases
  - The arrows on the arcs are not indispensable but are recommended to increase the legibility of the graphs in situations where the orientation would be ambiguous
  - *Example*: arc oriented from bottom to top
  - If several phases converge in the same transition (*synchronization*) or if several phases (*parallelism or competition*) occur in a transition, it is advisable to use a double horizontal line to highlight the end and beginning of sequences that must evolve in parallel

SFC: Actions

- Actions
  - Actions can be associated with each phase
  - *Definition*: in graphic form indicating the name and body (instructions that determine what must be done)
  - Each Boolean variable can be an action itself
    - If the action is executed, the value of the associated boolean variable is set to 1, otherwise it is set to 0
    - The name of the action coincides with that of the Boolean variable and the body of the action is not present
  - Full graphic representation:



SFC: Actions

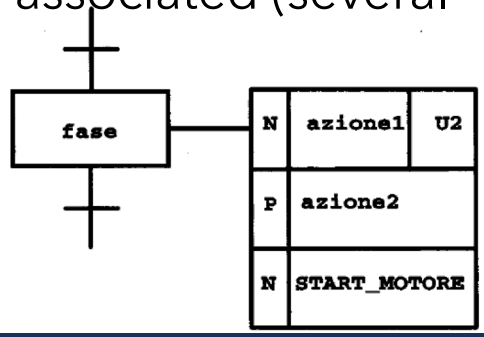
- Actions
  - Field (1): Action qualifier
  - Absent or equal to N, to qualify the action as not memorized, to be performed as long as the phase is active and again when it is deactivated
  - Equal to P, to qualify the action as impulsive, to be performed once when the phase is activated
    - Some implementations of the standard may assume that the action is performed another time when the phase is disabled
  - Equal to D, together with a TIME type constant, to qualify the action as delayed in time, i.e. performed after the indicated duration, if the phase is still active, and until the end of the activation, and then once again

SFC: Actions

- Actions
  - Equal to L, together with a TIME constant, to qualify the action as limited in time, i.e. performed for the duration indicated, or for the duration of the activation of the phase if less, and then once again
  - Equal to S or R to indicate the set and reset of a stored action, i.e. that if set in a phase it remains running even after the phase is deactivated until it is explicitly reset, and then again for one scan cycle
  - Equal to DS, together with a TIME type constant, to qualify the action as delayed and stored, i.e. stored after the indicated duration if the phase is still active.
  - Equal to SL, together with a TIME constant, to qualify the action as memorized and limited in time, i.e. performed for the indicated time, even if the phase is deactivated
  - Field (2): Name of the action

SFC: Actions

- Actions
  - Field (3): Signaling variable
  - Boolean variable which is placed at 1 in the execution of the action to indicate the execution of the action
  - It has documentation purposes and may be omitted
  - Field (4): Body of action
  - Description of what must be done using one of the languages defined by the standard including SFC
  - The field can be omitted if the action is defined separately
  - The action block must be linked to the phase with which it is associated (several actions may be associated to one phase)
  - Graphically: with a link to the right of the phase symbol establishing a relationship with the phase signaling variable



SFC: Actions

- Each action is associated with an implicit boolean variable name_action.Q which is true when the action is to be executed
- The body of the actions can be written in any of the languages provided by the standard, even SFC
  - It is possible to develop a programme in SFC with a top-down methodology → The macro-phases in which the system passes are identified → In the actions the detail of the sequences is developed
- Stages without actions can be qualified as waiting phases
  - The system remains there, doing nothing, until the conditions are met to move on to another phase

SFC: Rules

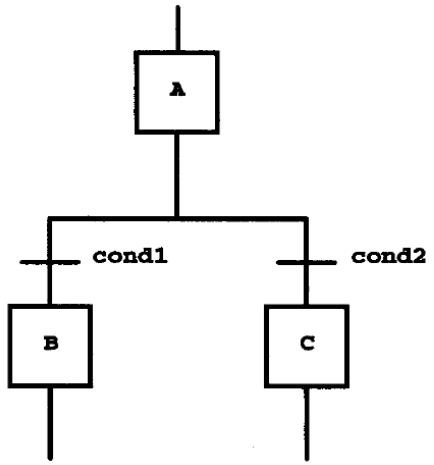
- Rules for the composition of an SFC:
  1. Two phases cannot be directly connected, there must always be a transition between them
  2. Two transitions cannot be directly connected, there must be at least one phase between them
- Evolution rules
  - The condition of an SFC is described by its active phases
  - An SFC can change condition by overcoming transitions
  - A transition is said to be enabled if all phases upstream of it are active
  - A transition is said to be superable if it is enabled and the condition associated with it is true
  - *Evolution rule*: if a transition is superable it is actually passed → all upstream phases are de-activated and all downstream phases are activated

SFC: Rules

- Evolution rules
  - The deactivation and activation operations follow one another in the exact order indicated
  - Their duration is linked to the particular implementation and is given by the time that elapses between two successive evaluations of the graph
  - Behavior ambiguity: it may happen that distinct transitions can be passed at the same time -> the rule states that if several transitions become superable at the same time, they are all passed at the same time
  - Unstable phase (in the sense of asynchronous machines): the condition associated with the output transition is already true when the phase is activated -> the rule states that the associated actions are anyway performed before phase deactivation
  - The activation duration of a phase can be null

SFC: Structures

- Classic programming structures
  - Simple sequence: alternating phases and transitions in series
    - Describes simple sequential (serial) command structures
  - Choice (or divergence) between several activities: when a phase is followed by several transitions
    - *Example:*
      - If phase A is active, phase B is activated if condition cond1 is true and cond2 is false
      - phase C if cond1 is false and cond2 is true
      - cond1 and cond2 express the choice between two possibilities

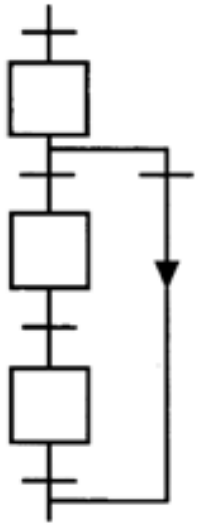


SFC: Structures

- Classic programming structures
  - Mutually exclusive choice
    - The conditions of choice must always be mutually exclusive: at most only one condition at a time may be true
  - Mutual natural exclusion
    - The conditions of choice are never true at the same time because of their nature
    - *Example*: presence of the same object on the left and on the right
  - Mutual exclusion imposed
    - The mutual exclusion is implemented in the construction of the conditions, also assigning a priority of superability
    - The standard provides that the branches of the divergence can be numbered in order of priority or priority can be imposed by acting on the conditions of the transitions
    - *Example*: A transition is activated only if the relative condition occurs and, at the same time (AND), the condition of another transition is not verified

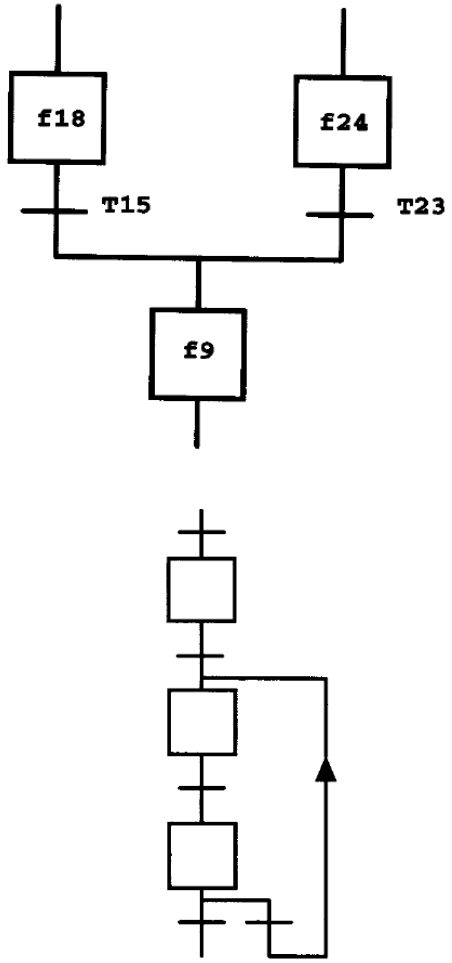
SFC: Structures

- Classic programming structures
  - Special case: sequence jump



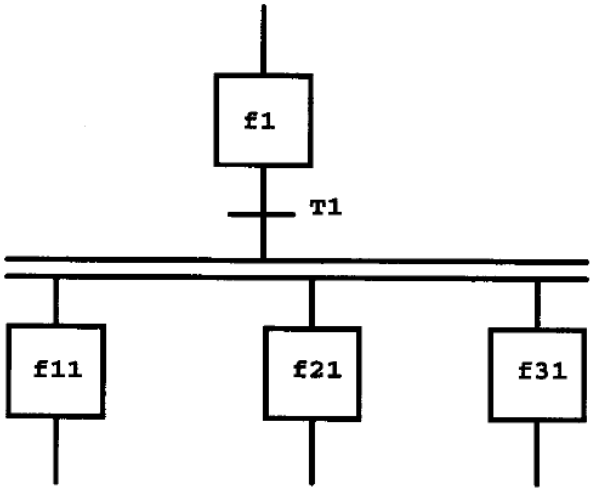
SFC: Structures

- Classic programming structures
  - Convergence of several activities: when several sequences end in the same phase through different transitions
  - *Example*: phase f9 will be activated if the T15 transition and/or the T23 transition will be overcome
  - Convergence is closure natural choice
  - *Special case*: cycle of a sequence



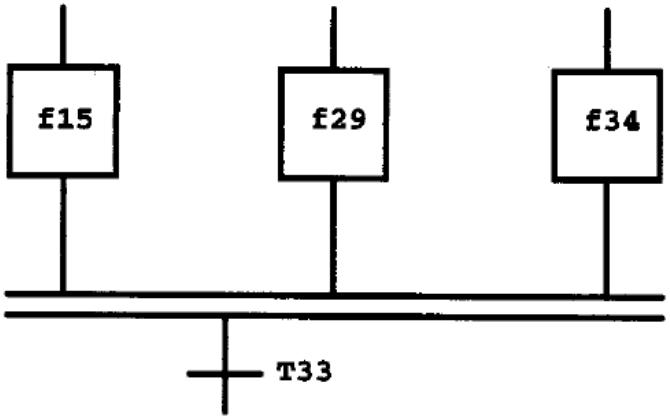
SFC: Structures

- Classic programming structures
  - Parallelism (or competition) between several activities: when a transition is followed by several phases
  - *Example*: If the T1 transition becomes superable will be activated simultaneously phases f11, f21 and f31 giving place to several sequences that will evolve independently of each other



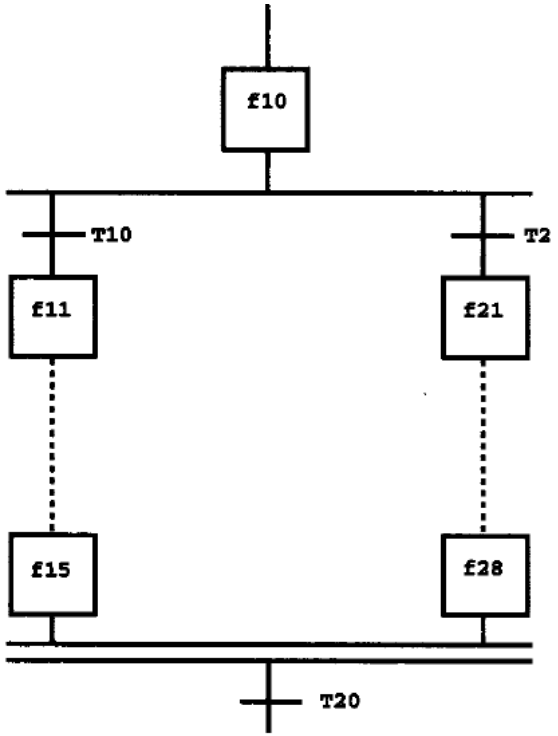
SFC: Structures

- Classic programming structures
  - Synchronization between several activities: when several phases precede the same transition
  - Condition for the transition to be overcome is that all parallel sequences are finished and that so the final stages are all active
  - Example: the final stages f15, f29 and f34 are synchronized by the transition T33
  - Synchronization is the natural closing of a parallelism



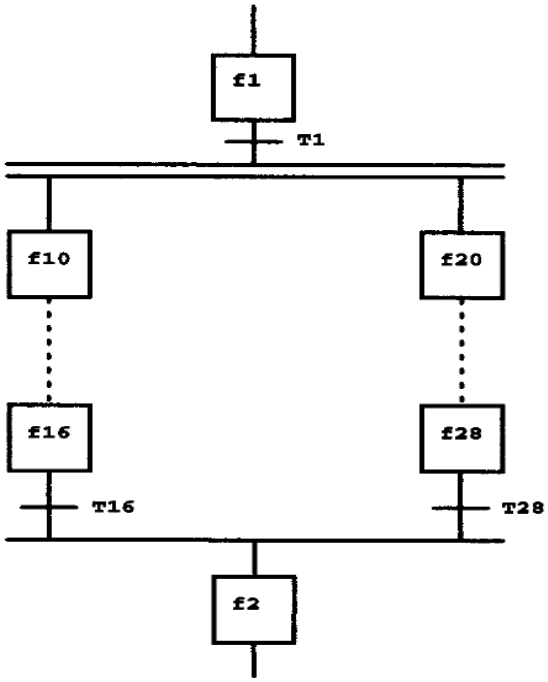
SFC: Structures

- Classic programming structures
  - Programming structures to be avoided
    - Wrong structure: choice with synchronization
    - The T20 transition will never be overcome



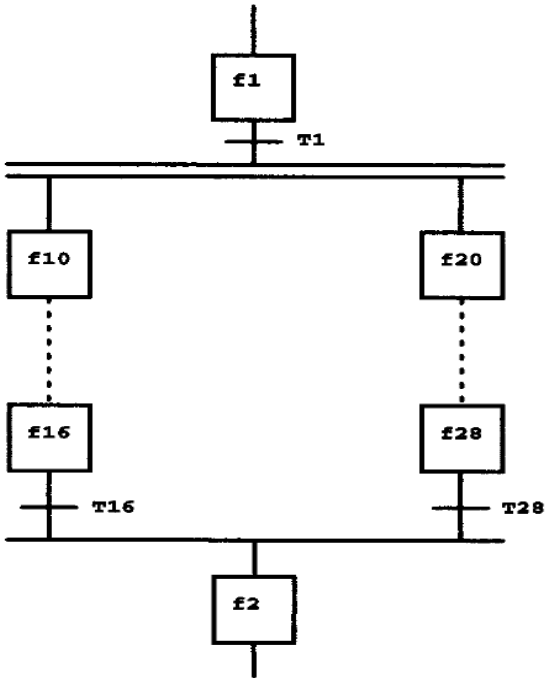
SFC: Structures

- Classic programming structures
  - Programming structures to avoid
    - Ambiguous structure: parallelism with convergence
    - Phase F2 becomes active if one between T16 and T28 becomes passable



SFC: Structures

- Classic programming structures
  - Programming structures to avoid
    - Ambiguous structure: parallelism with convergence
    - Phase F2 becomes active if one between T16 and T28 becomes passable



SFC: Discussion

- Advantages of the SFC in the control design
  - The SFC has great expressive power
  - It is the best logic/sequential control programming language for discrete event systems
  - The control is achieved through sequences of activities, whose evolution depends on logical conditions
  - Sequential logic control of discrete event systems is not unique to industrial automation alone
  - *Examples*: traffic light system, train station, access to a communication channel, access to databases, server queue, agricultural systems, etc.
  - Programming in SFC corresponds to describing, according to its syntax, the desired behavior of the system

SFC: Discussion

- Advantages of the SFC in the control design
  - SFC is used for the description of the functional specifications
  - The identification of the functional specifications for an automation system is preparatory to the design of the control algorithms that must realize it
  - They must be expressed in a language that does not allow ambiguity
  - The SFC can usefully be used to describe the functional specifications -> the actions to be performed and the conditions to be evaluated are written *in natural language*

SFC: Discussion

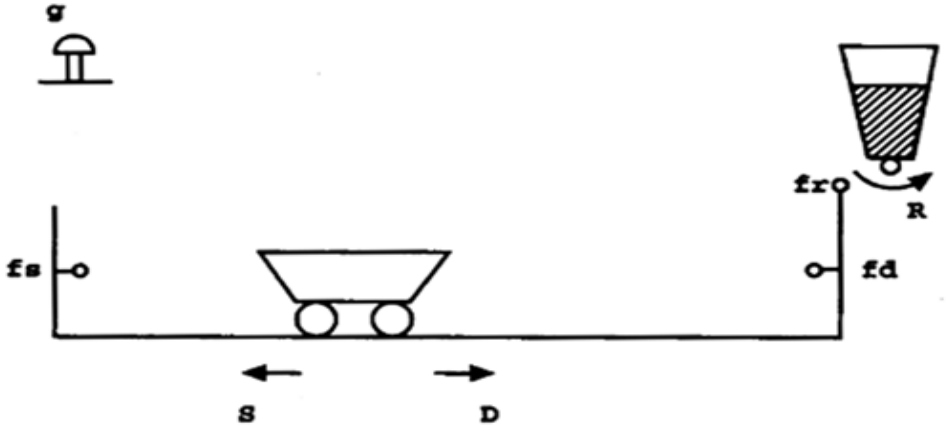
- Advantages of the SFC in the control design
  - *Examples*: "open valve", "go right", "temperature higher than 20°C", "end of stroke reached", etc.
  - The unambiguousness of the SFC allows to identify any specifications not clearly defined
  - The SFC produced is self-documenting
  - It describes exactly what the system needs to do, the steps the system needs to go through and the reasons why it might need to change phase
  - Implementation efficiency
  - The programs written in SFC are highly efficient because in each graph evaluation only the actions associated to the active phases have to be actually executed and only the outgoing transitions to these phases have to be evaluated

SFC: Examples

- SFC programming examples
  - The proposed examples are for teaching purposes only
  - Real examples should include the management of emergency conditions, switching from automatic to manual mode and vice versa, etc.
  - The proposed solutions are only one of the possible solutions for the problems considered
  - There are more and more solutions to the same problem, which may differ in clarity, length, memory occupation, etc.
  - For the sake of brevity, only the body of the programs will be presented in the following
  - Variables definition phases and their connection to the physical inputs and outputs of the device are omitted

SFC: Examples

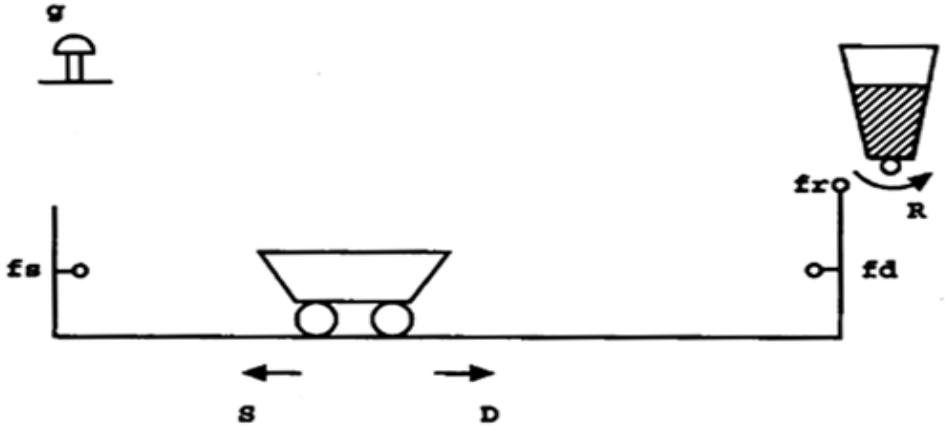
- SFC programming examples
  - The automatic trolley
  - At the request of an operator, the trolley must move from the left to the right of a track, be loaded by tipping a tank and return to the left to unload the material



- Digital input signals:
- Limit switch for the carriage: fs and fd
- Limit switch for the tank: fr
- Signal coming from the cycle activation button: g
- Digital control signals:
- Trolley motion: S and D
- Tank tipping: R

SFC: Examples

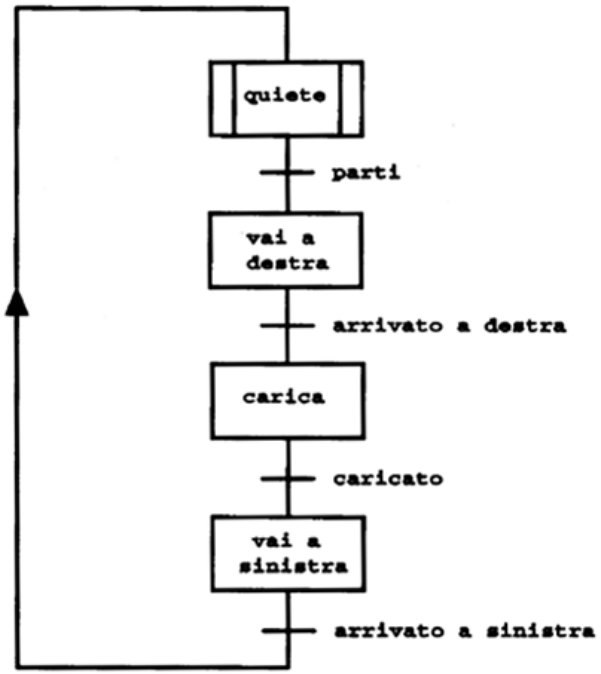
- SFC programming examples
  - The automatic trolley
  - At the request of an operator, the trolley must move from the left to the right of a track, be loaded by tipping a tank and return to the left to unload the material

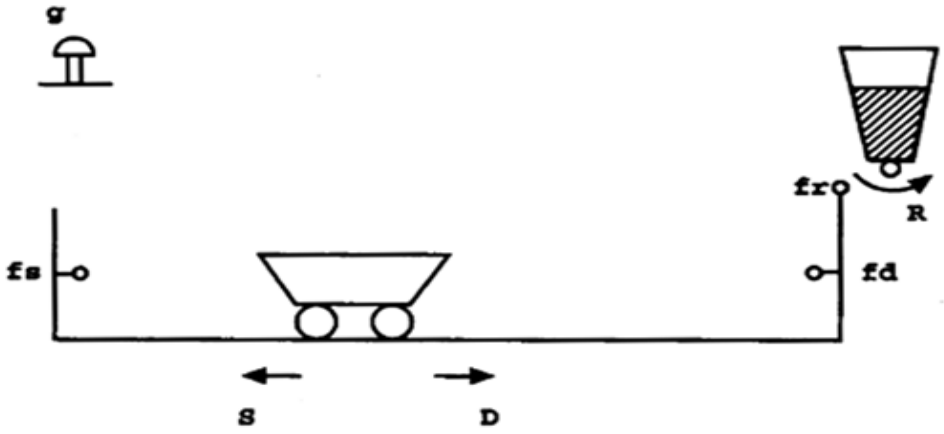


- Digital input signals:
- Limit switch for the carriage: fs and fd
- Limit switch for the tank: fr
- Signal coming from the cycle activation button: g
- Digital control signals:
- Trolley motion: S and D
- Tank tipping: R

SFC: Examples

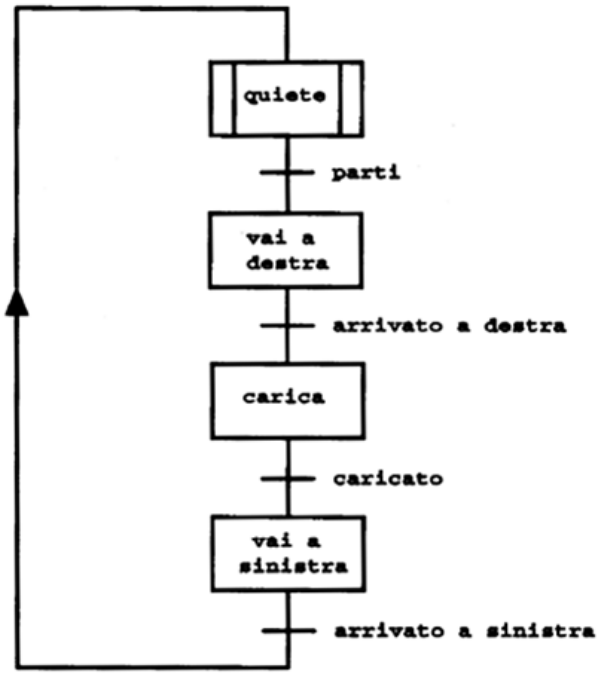
- SFC programming examples
  - Functional SFC

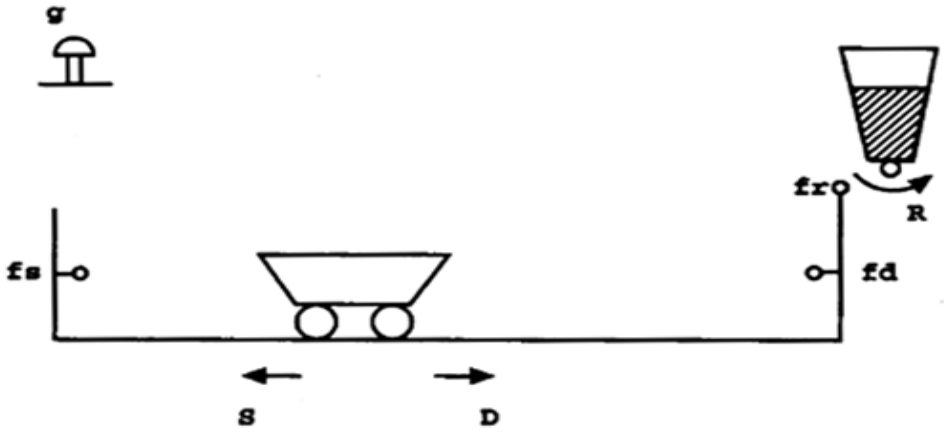




SFC: Examples

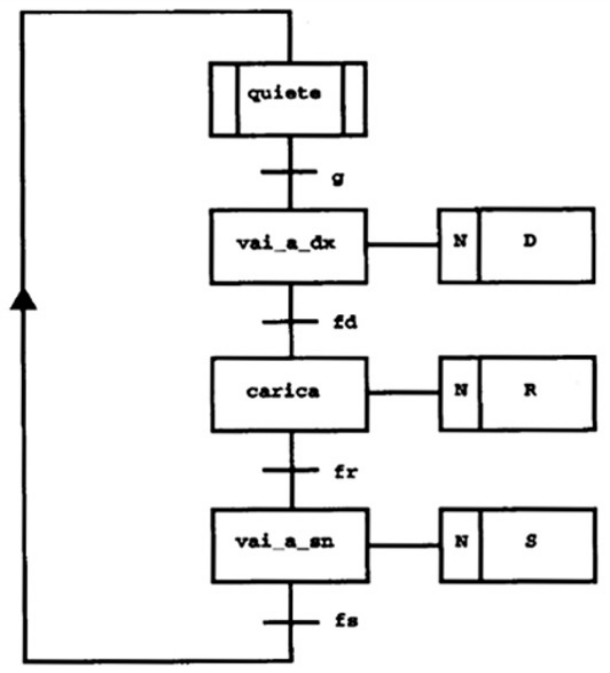
- SFC programming examples
  - Functional SFC

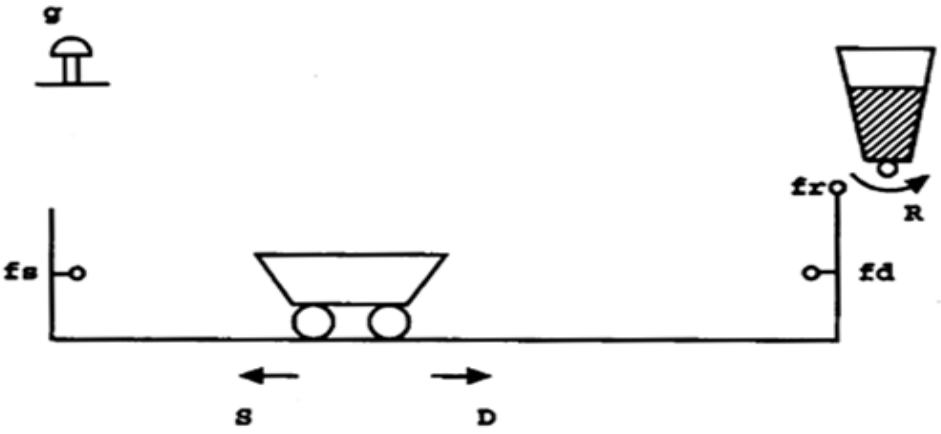




SFC: Examples

- SFC programming examples
  - Control SFC





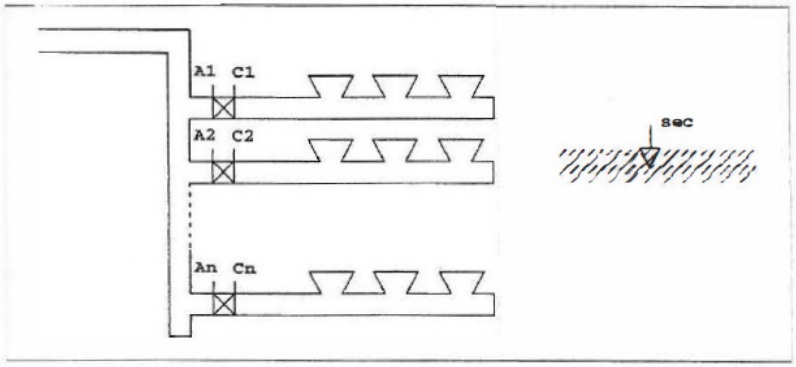
SFC: Examples

- SFC programming examples

4.8.2 Il sistema di irrigazione

Si abbia il sistema di irrigazione da automatizzare schematizzato in Figura 4.19. La partenza del ciclo di irrigazione viene data da un sensore, `sec`, di umidità che rileva la secchezza del suolo; si deve comunque prevedere un ciclo di irrigazione ogni tre giorni al massimo. La rete irrigatrice è costituita da *n* rami, ciascuno comandato da un'elettrovalvola, i quali devono essere alimentati in sequenza data la limitatezza della portata d'acqua disponibile. L'*i*-esima elettrovalvola è comandata in apertura dal segnale `Ai` che deve durare 400 millisecondi e in chiusura dal segnale `Ci` che deve durare 200 millisecondi. Per ogni ramo deve essere prevista un'ora di annaffiatura.

La Figura 4.20 riporta due possibili SFC per la realizzazione dell'automatismo. Il grafo di sinistra utilizza solo azioni con qualificatore `N` (omesso), men-



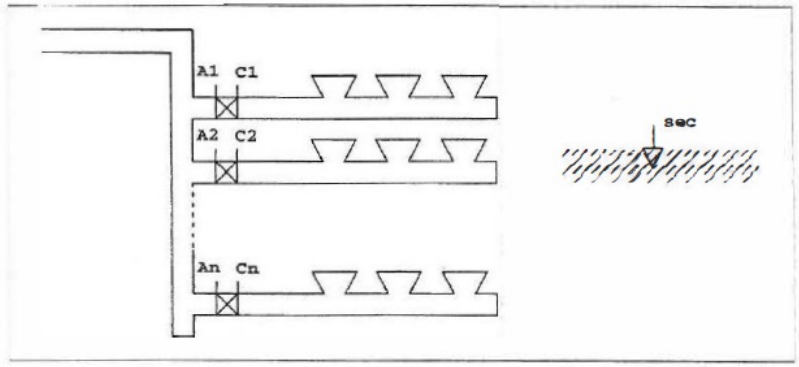
SFC: Examples

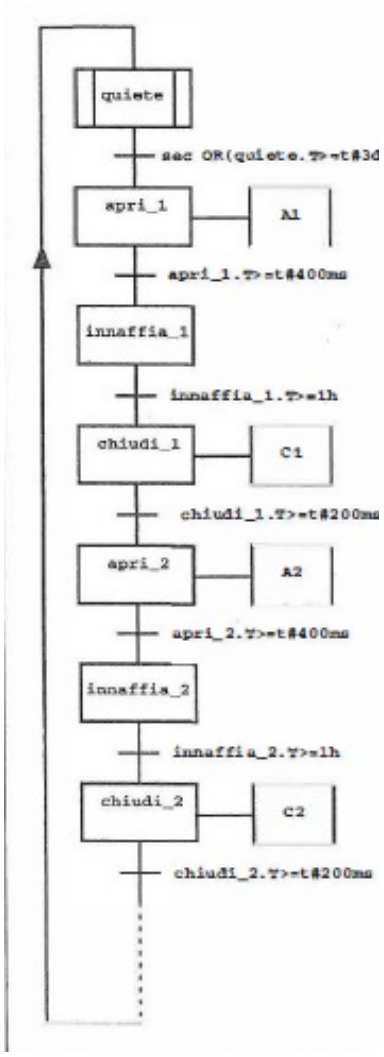
- SFC programming examples

4.8.2 Il sistema di irrigazione

Si abbia il sistema di irrigazione da automatizzare schematizzato in Figura 4.19. La partenza del ciclo di irrigazione viene data da un sensore, sec, di umidità che rileva la secchezza del suolo; si deve comunque prevedere un ciclo di irrigazione ogni tre giorni al massimo. La rete irrigatrice è costituita da *n* rami, ciascuno comandato da un'elettrovalvola, i quali devono essere alimentati in sequenza data la limitatezza della portata d'acqua disponibile. L'*i*-esima elettrovalvola è comandata in apertura dal segnale Ai che deve durare 400 millisecondi e in chiusura dal segnale Ci che deve durare 200 millisecondi. Per ogni ramo deve essere prevista un'ora di annaffiatura.

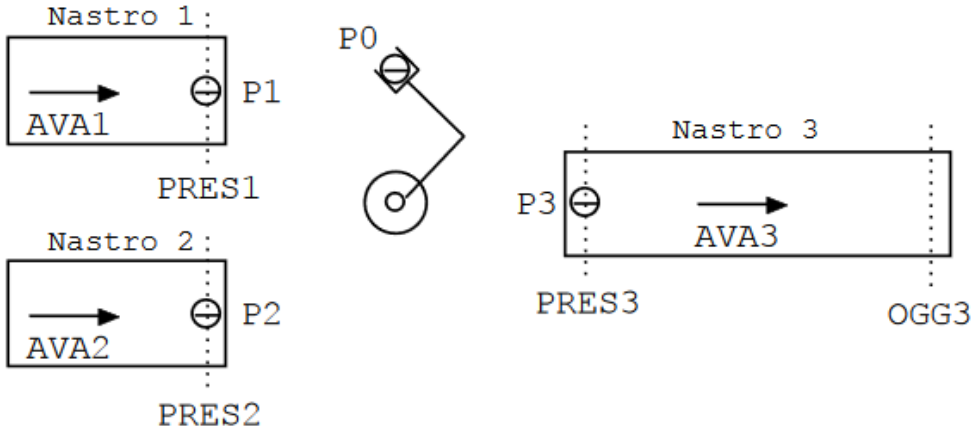
La Figura 4.20 riporta due possibili SFC per la realizzazione dell'automatismo. Il grafo di sinistra utilizza solo azioni con qualificatore N (omesso), men-





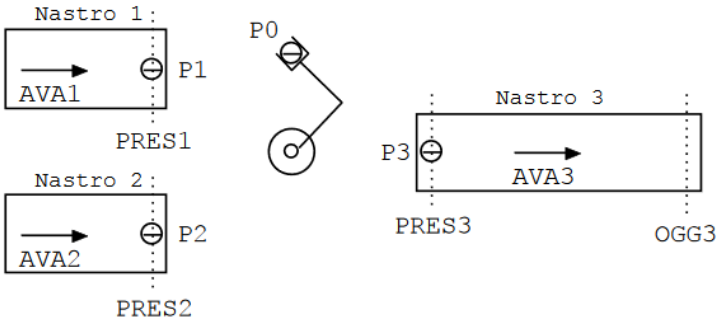
SFC: Examples

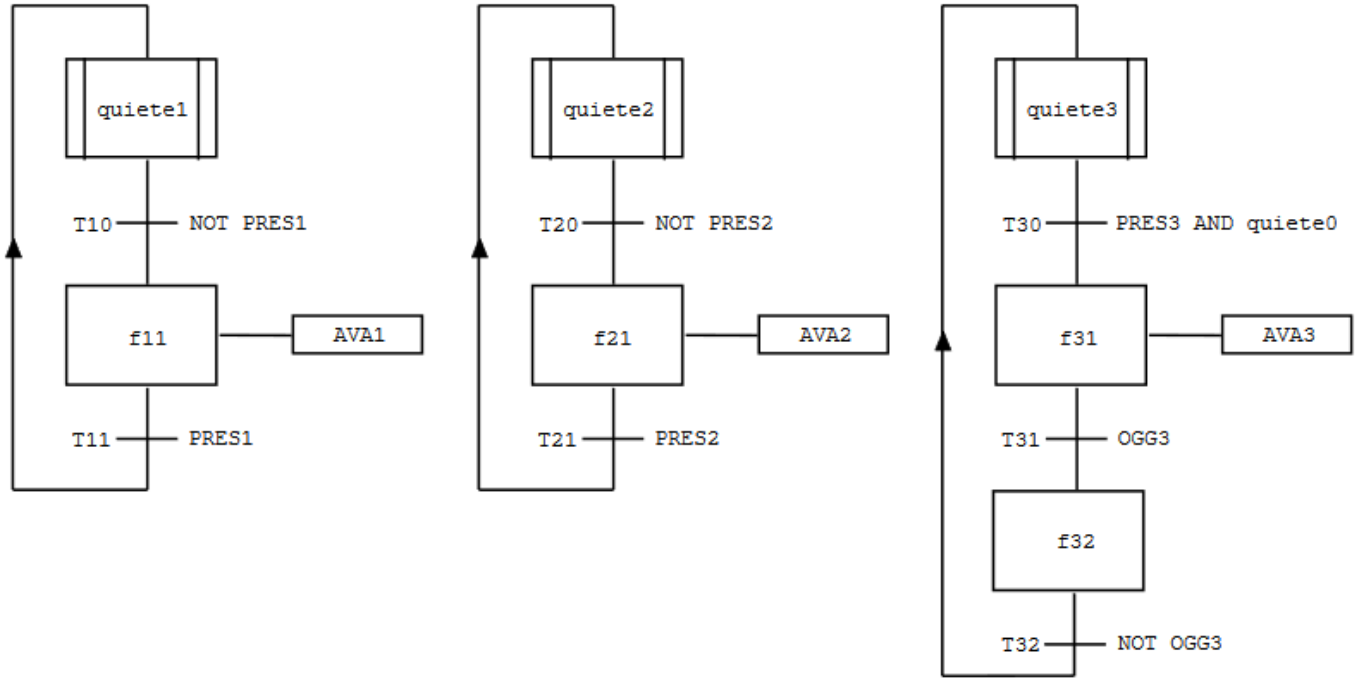
- SFC programming examples
  - Robotic cell
    - 2 input transporting belts
    - 1 output transporting belt
    - Robot must move boxes randomly arriving from input belts on the output belt
  - Input control
    - PRESi: object presence in Pi, i=1..3
    - OGG3: object at the end of belt 3
    - OKi: robot is in Pi, i=0..3
  - Output control:
    - AVAi: moving belt i, i=1..3
    - VAIi: moving robot in Pi, i=0..3
    - CLOSE: close robot gripper
    - OPEN: open robot gripper



SFC: Examples

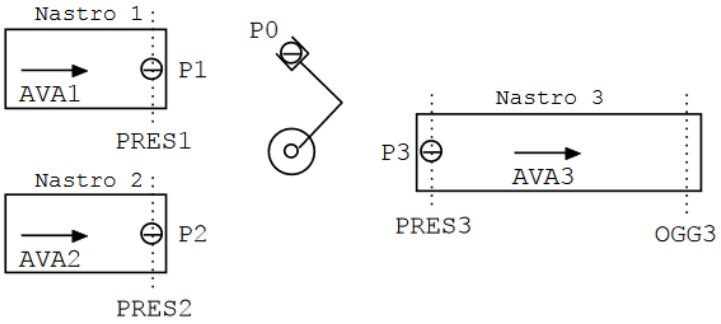
- SFC programming examples
  - Belt control

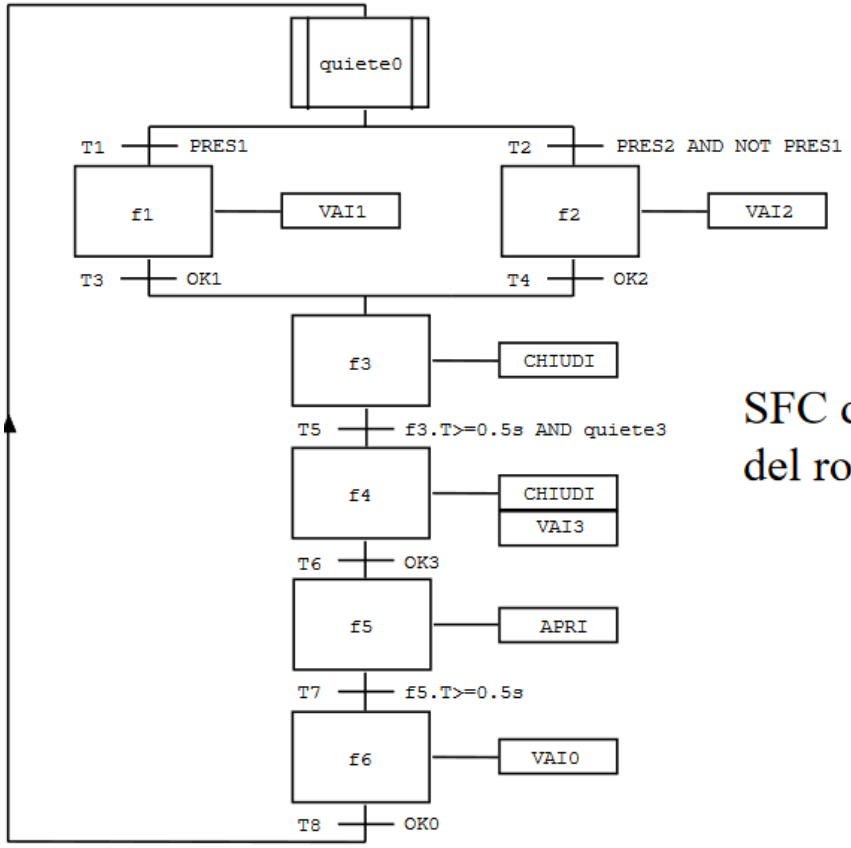




SFC: Examples

- SFC programming examples
  - Robot control





SFC di controllo del robot

MEIM | MASTER IN ENTREPRENEURSHIP INNOVATION MANAGEMENT IN COLLABORATION WITH MIT SLOAN

IN COLLABORATION WITH MIT MANAGEMENT SLOAN SCHOOL | UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

MASTER MEIM 2023

Thank you!