Università
della
Svizzera
italiana

Facoltà
di
scienze
informatiche

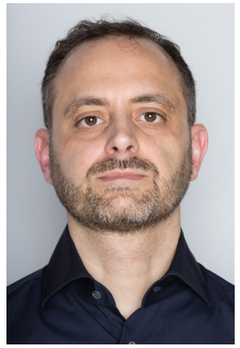
Edge Computing in the IoT

Introduction

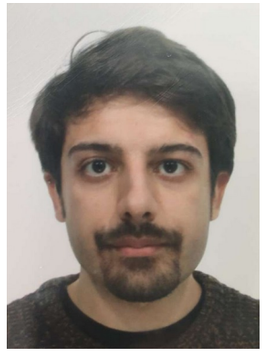
Alberto Ferrante (alberto.ferrante@usi.ch)

Course Organization

- Persons involved:



Alberto Ferrante
Course Director

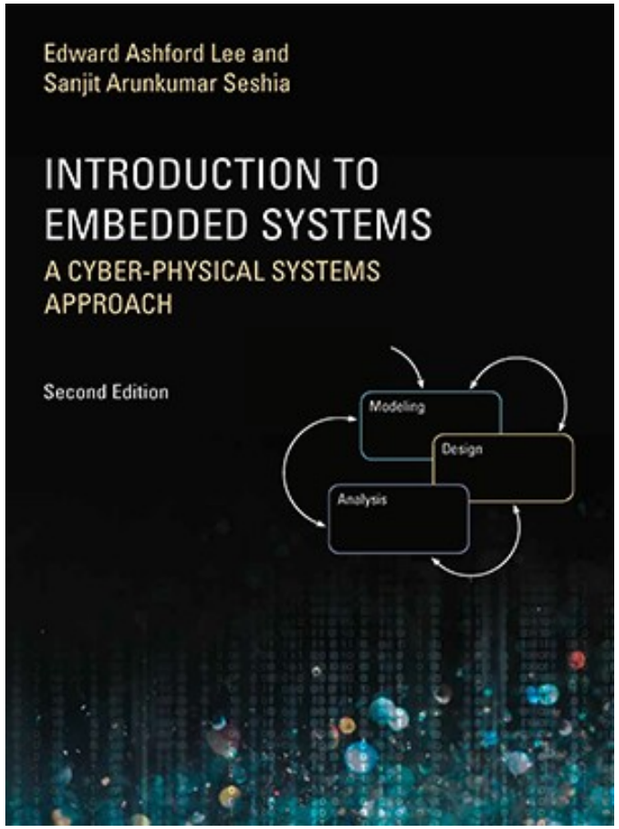


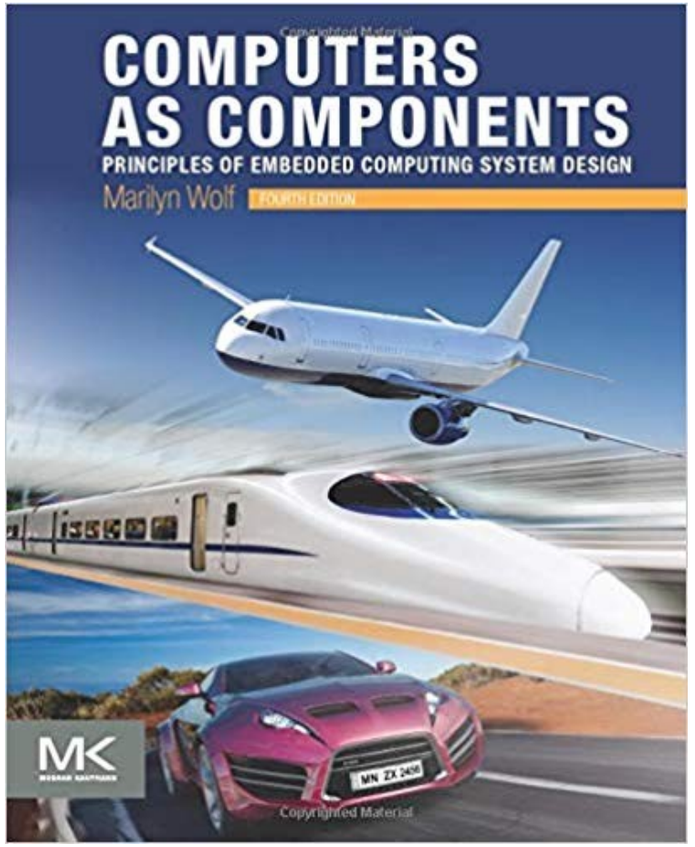
Luca Butera
Teaching Assistant

Program of the Course

- Introduction
- Design of cyber-physical systems: from requirements to the working system
- Embedded hardware
  - Introduction to embedded architectures
  - Sensor and actuators
  - Energy and power in IoT devices
  - Input and Output
- Software development platforms
- Embedded software and operating systems
  - Introduction to embedded programming
  - Software on bare metal vs. operating system
  - Embedded/IoT operating systems
  - RTOS
  - Application flow control: wait, polling, and interrupts
  - Basic execution time analysis
- Networking & communication
- IoT data analysis
  - Distributed architectures
  - Machine learning in IoT
  - Cloud Vs. edge computing for ML
  - ML algorithms for edge computing
- Security & reliability

Suggested Books





https://ptolemy.berkeley.edu/books/leeseshia/

Evaluation

Title	Type	Grade
Presentation about one IoT network protocol	Presentation during the course	M_1
Course project	Group report + presentation	M_2
Written exam	Open-ended + closed-ended questions	M_3

$$Final\ grade = \begin{cases} M_3 \geq 6 ; 0.2 * M_1 + 0.4 * M_2 + 0.4 * M_3 \\ M_3 < 6 ; M_3 \end{cases}$$

Agenda Of the Course

- Classes
  - Monday From 13:30 to 15:10
  - Friday from 11:00 to 12:30
- Course slides and other material: **iCorsi**
- Reception hours:
  - After the classes
  - Meetings at other times can be arranged by e-mail

Agenda Of the Course

- Sept 17th: Course project start
- Nov 17th: presentations on network protocols
  - Network protocols will be assigned to groups of students on Nov 4th
- Dec 22nd: final project presentations

Course Project

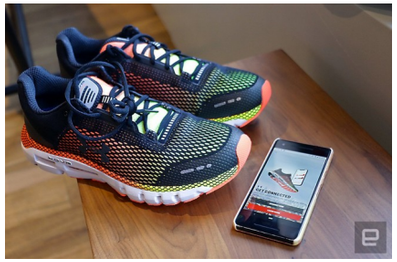
- The purpose of the project is to apply in practice what we learn during classes
  - **“Project” NOT equal to “programming an Arduino board”**
- Projects will be assigned to groups of 3-4 students
- Some time for the project will be allocated during the course (about 6-8 hours)
  - To interact in the project groups and with the instructors
- ++ Extra time at home
- Even though the projects will be defined at the beginning of the course, students are not required to work on it during the whole timespan of the course

Computers Are Everywhere!











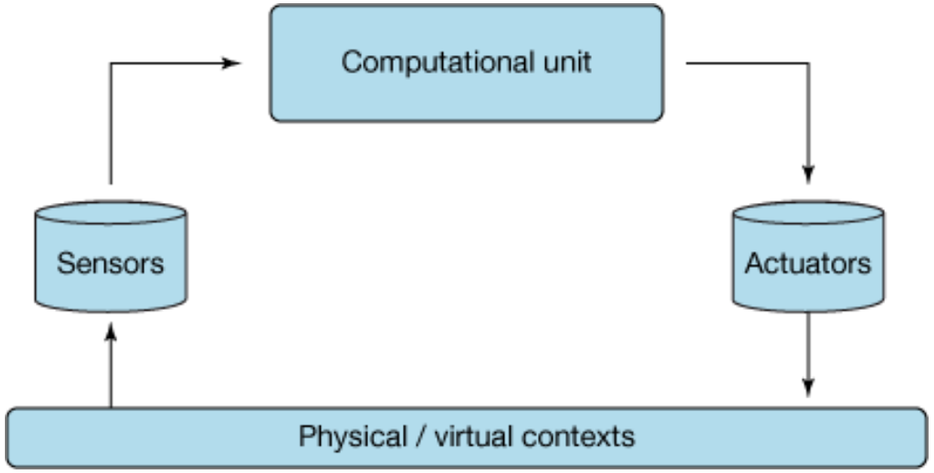


Embedded Systems

- Computers that are *embedded* into something else
  - The embedding objects do not look like computers
  - Human-machine interaction is different from standard computers
  - Limitations to their hardware and software

What's a Cyber-physical System?

- A system exhibiting synergy of computational and physical components
- A physical system with computational intelligence
  - Physical system = **the plant**



https://developer.ibm.com/articles/ba-cyber-physical-systems-and-smart-cities-iot/

What's a Cyber-physical System?

- Unlike more traditional embedded systems, a full-fledged CPS is typically designed as a network of interacting elements with physical input and output
  - Embedded systems can be seen as building blocks for CPSes
- Common applications of CPS typically fall under sensor-based communication-enabled autonomous systems

Internet of Things

- The Internet of things (IoT) is the extension of Internet connectivity into physical devices and everyday objects
  - Electronics + Internet connectivity + sensors and actuators
  - Devices can communicate and interact with others over the Internet
  - They can be remotely monitored and controlled

Internet of Things

- Convergence of multiple technologies
  - Real-time analytics
  - Machine learning
  - Embedded systems.
    - Traditional fields of embedded systems, wireless sensor networks, control systems, automation (including home and building automation), and others all contribute to enabling the Internet of things.

IoT Applications

- Connected vehicles and transport infrastructure
- Building and home automation
- Wearable technology
- Appliances with remote monitoring capabilities
- Internet of Medical Things
- Manufacturing
- Agriculture
- Energy management
- …
- ...

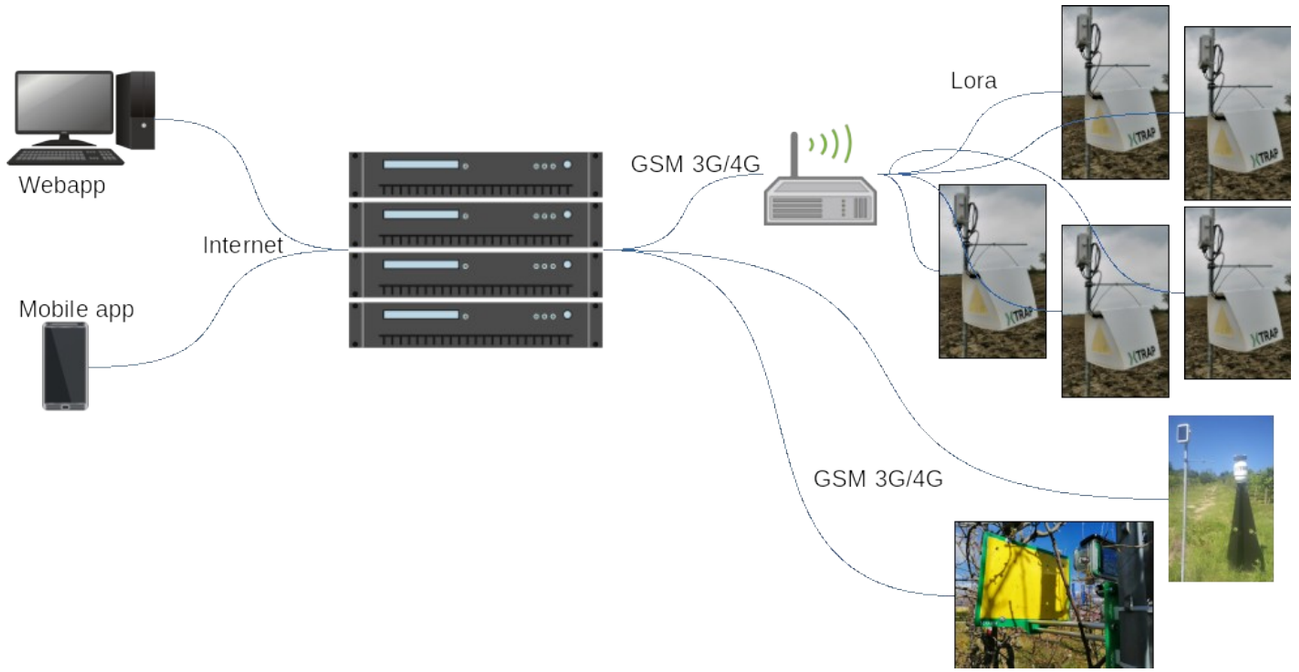
IoT Example – Smart Agriculture

- Data collection from planting to harvest
- Data are used to plan and optimize, e.g.
  - Planting with optimal distance among seeds (plants)
  - Generate crop performance reports and plan for the next season
- Real-time monitoring of processes

https://www.climate.com

IoT Example – Smart Agriculture

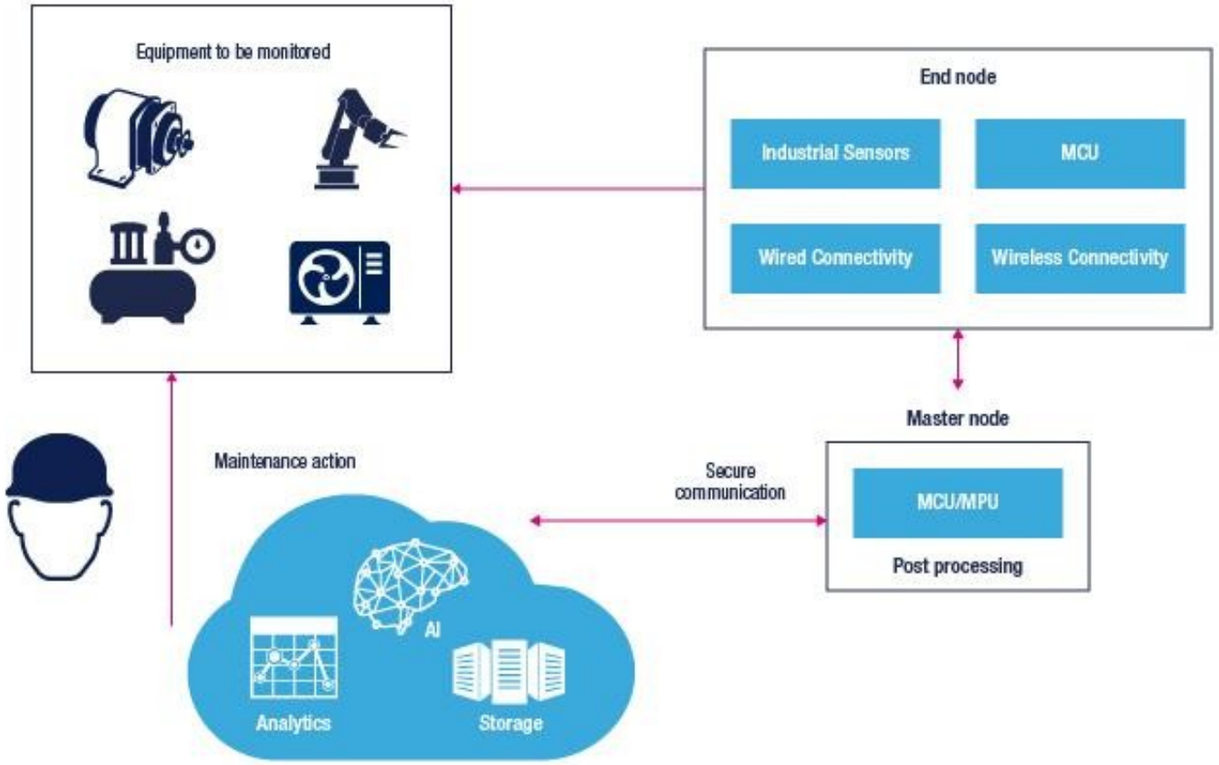
- Solutions for monitoring pest insects by means of camera-equipped traps
  - Machine learning is used to detect specific insects and to count them



Alippi C., Ferrante A. (2021) SmartTrap - Machine learning enhanced traps for pest insects monitoring

IoT Example – Smart Industry

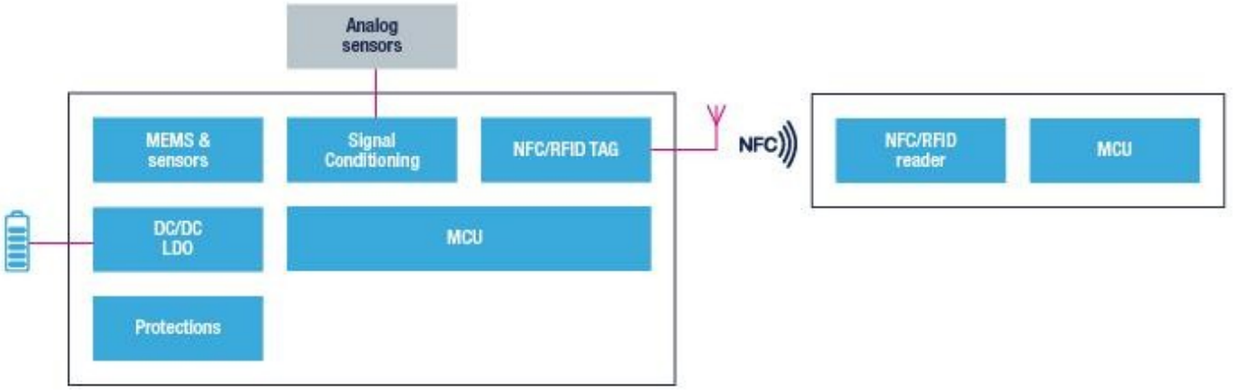
Predictive maintenance



https://www.st.com/en/applications/iot-for-smart-industry.html

IoT Example – Smart Industry

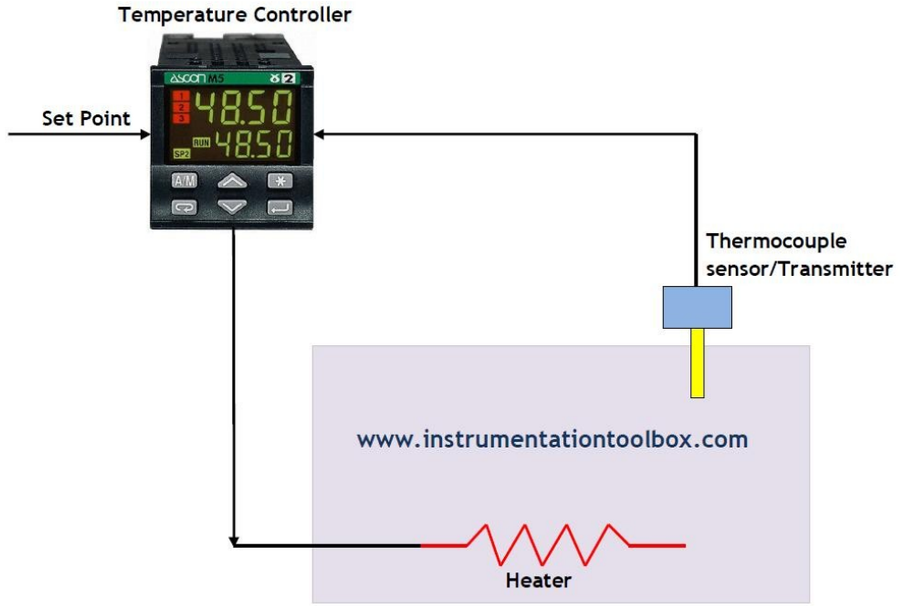
Tracking – Goods Guarantee



https://www.st.com/en/applications/iot-for-smart-industry.html

Interactions With the Physical World

- To interact with the physical world there is the need of
  - Measuring: sensors
  - Actuating: actuators
- We need to deal with all problems related to the physical world
  - Accuracy
  - Reliability
  - Safety
- We read a parameter (e.g., temperature) and we are able to control it



https://www.instrumentationtoolbox.com/2016/06/how-temperature-control-loop-works.html

Sensors and Actuators

- Sensor: a device, module, or subsystem whose purpose is to detect events or changes in the environment
  - Measure may be direct or indirect
    - Often performed **indirectly** by means of electric current
    - In some cases, the parameter of interest cannot be measured and it is estimated by means of other parameters
  - Can be attached directly to the considered system (e.g., the microcontroller board) or by means of a networked device
  - Examples
    - Light
    - Temperature
    - Biochemical
    - Presence

Sensors and Actuators

- Actuator: a component of a machine that is responsible for moving and controlling a mechanism or system
  - Control is usually performed on a control parameter that, in turn, influences the desired parameter
    - e.g., we control current in a resistor, this, in turn, controls generated heat
  - Can be attached directly to our local system or by means of a networked device
  - Examples
    - Electric motor
    - Valve
    - Resistor
    - Relay
    - LED

Embedded Architectures Vs. PCs

PCs	Embedded
Largely homogeneous	Heterogeneous
Often not energy-limited (unless on battery)	Often energy-limited
Efficient power dissipation: 100 and more W	Poor power dissipation, low power required
Plenty of computational and memory resources	Often limited computational and/or memory resources
Classic human-machine interaction (display, mouse, and keyboard)	Human-machine interaction often limited or non existent
High level programming languages and complex operating systems	High level programming languages, but used at lower level. Simpler and smaller operating systems
Use of virtual memory, 3 levels of cache, ...	In some cases, no VM and max 1-2 levels of cache
User driven	User driven or event driven / connection with the physical world

Design of Embedded Systems

- Each embedded system must meet a completely different set of requirements, any or all of which can affect the compromises and trade-offs made during the development of the product
  - Processing power
  - Memory
  - Energy / Power
  - Reliability
  - Lifetime
  - Development cost
  - Number of units
  - Cost

For example, let's list the main requirements for
- A smartphone (different purposes?)
- The main control system of a space rover
- A sensor node used for sensing humidity and temperature in a vineyard
- The Engine Control Unit of a vehicle

Programming Embedded Systems, 2nd Edition, Anthony Massa, Michael Barr
https://www.oreilly.com/library/view/programming-embedded-systems/0596009836/ch01.html

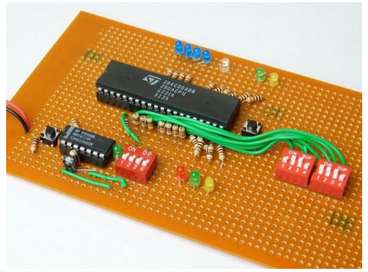
Challenges in Embedded Systems Design

- We need to find the best trade-off among conflicting requirements
  - Performance
    - Throughput, real-time capabilities, …
    - Accuracy
    - …
  - Hardware costs
    - CPU
    - Memory
    - Sensors
    - …
  - Power and energy
  - Security
  - Reliability and Safety

Challenges in Embedded Systems Design

- Embedded systems are
  - Complex to test: exercising an embedded system is more difficult as it may require physical inputs and conditions that are not always reproducible
    - Simulation or partial simulation
  - Limited observability and controllability:
    - Difficult to see what is going on inside the system as we might not have standard debug outputs
      - There exist protocols for debug
    - Debug real-time applications or application where timing matters is (extremely) complex

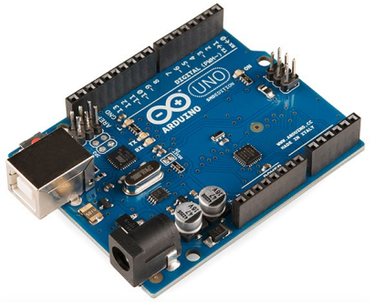
Evolution of Embedded Architectures and Platforms



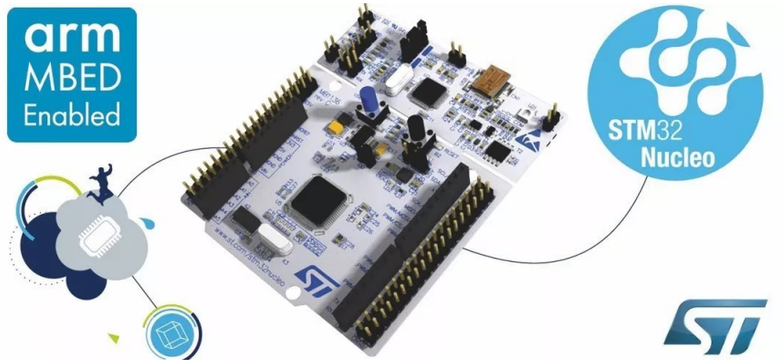
```
            org     1000h       ;Origin at 1000h
memcpy      public
loop        ld      a,b         ;Test BC,
            or      c           ;If BC = 0,
            ret     z           ;Return
            ld      a,(de)      ;Load A from (DE)
            ld      (hl),a      ;Store A into (HL)
            inc     de          ;Increment DE
            inc     hl          ;Increment HL
            dec     bc          ;Decrement BC
            jp      loop        ;Repeat the loop
            end
```

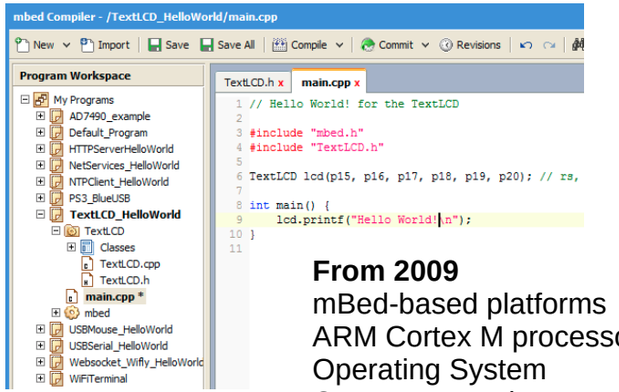
- Custom hardware solutions
- A plethora of operating systems and programming languages: evolved from assembly to more high-level languages

Late ‘70s
Zilog Z80
2.5 – 50MHz
Max 64kB of memory space



Early 2000s
Arduino
Simplified C and IDE
Simplified and standardized interface for sensors/actuators





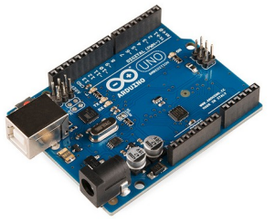
From 2009
mBed-based platforms
ARM Cortex M processors
Operating System
C++ programming
Drag & drop firmware deploys

Modern Embedded Devices

Playstation

Satellite electronics

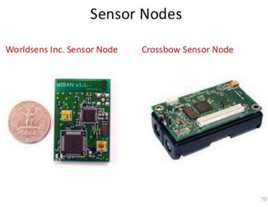
Development boards



ECU of an F1 car

Smartphone

Sensor node



Evolution of Embedded Architectures and Platforms

- Not all embedded systems are the same, we have systems
  - With/without operating system
  - Very limited computational resources / extensive computational resources
  - Battery operated / connected to main power
  - Very small / relatively big
  - Real time / non real time
  - Critical / non critical tasks
  - Custom / based on “general purpose” solutions
- In the early ages, only highly specialized personnel could design and program embedded systems
- With the availability of more modern development platforms and of systems with more computational resources, the task of designing embedded systems got *apparently* easier

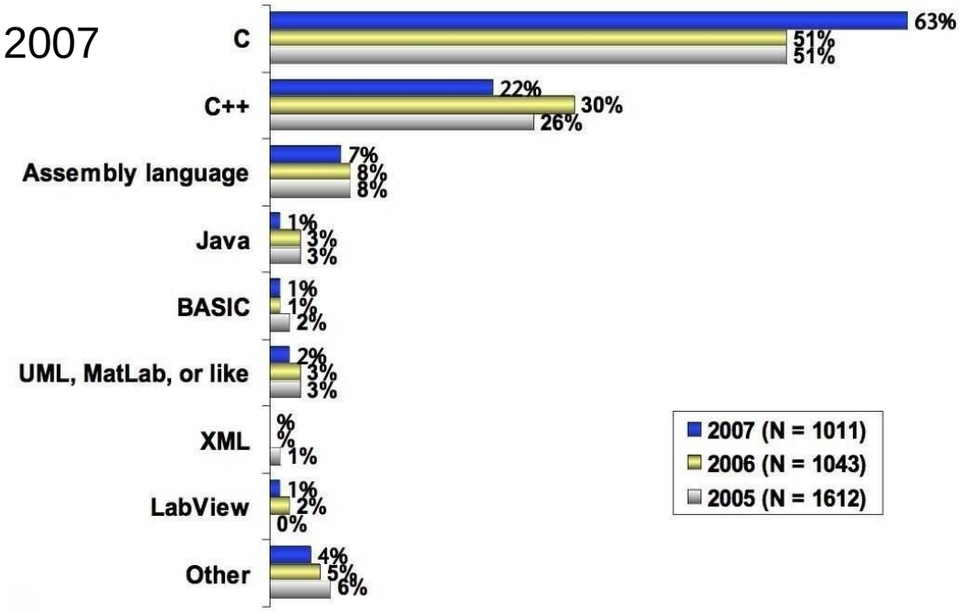
Evolution of Embedded Software Programming

1997

	Have used	Plan to use
Base: All respondents	409	409
	100.0	100.0
C	80.7	80.2
Assembly	70.4	67.5
C++	35.9	53.3
Visual Basic	13.0	15.9
Basic	12.5	8.8
Ada	6.4	8.3
Java	6.1	15.4
HDL/VHDL	6.1	7.8
Pascal	4.2	1.7
Fortran	3.7	2.0
Forth	3.4	2.4
PL/M	3.2	2.0
Cobol	1.0	1.0
Modula 2	1.0	0.2
Smalltalk	0.5	0.5
Other	2.9	3.4
Any programming language (Net)	96.8	97.3
Don't know	-	0.5
None	3.2	2.2

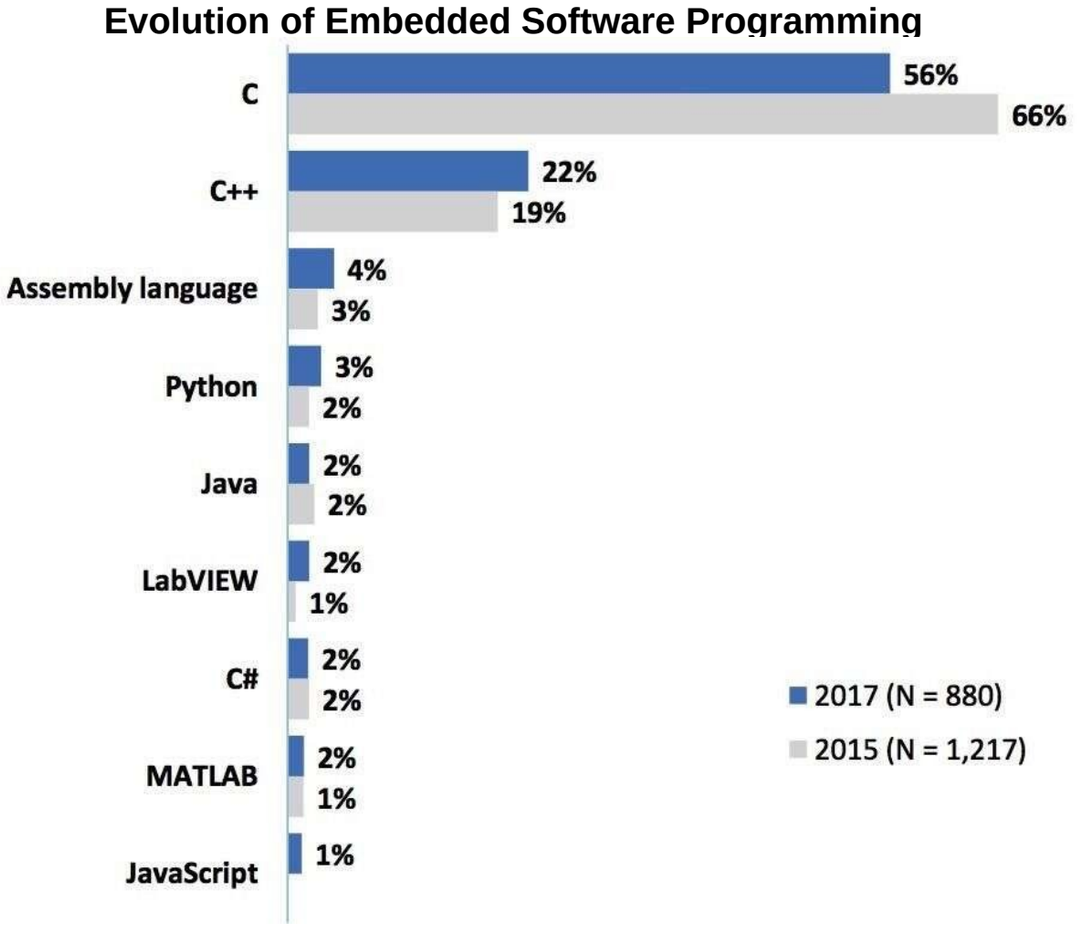
Note: Multiple response

2007



- C and assembly represented the predominant programming languages in 1997.
- Use of assemby dropped in the next 10 years

How embedded software development has evolved over 20 years https://www.embedded.com/electronics-blogs/embedded-view/4459113/How-embedded-software-development-has-evolved-over-20-years-



- In 2015, 2% of respondents said their next project would likely be programmed in Python. In 2017, that number jumped to 5%.

How embedded software development has evolved over 20 years https://www.embedded.com/electronics-blogs/embedded-view/4459113/How-embedded-software-development-has-evolved-over-20-years-

Evolution of Embedded Software Programming

- In general, embedded systems have become more complex
  - Network connectivity
    - Network management
    - Security
  - More tasks, sometimes more complex
    - More lines of code to manage
- In general, the mere act of programming embedded systems have become easier
  - Less low-level programming, more high level options available
  - Frameworks (e.g., mBed) that help programmers by providing libraries and APIs
    - Programmers do not need anymore to worry about all low-level details

Modern Embedded Architectures

- Many different solutions, depending on the system
- Most used processors are based on IP cores by *arm*
  - Arm Cortex-A series:
    - Range of solutions for devices undertaking complex computing tasks
    - Used in smartphones and down to other less powerful applications
  - Arm Cortex-R series:
    - Range of processors optimized for high performance, hard real-time applications
  - Arm Cortex-M series:
    - Smallest/lowest power processors build by Arm
    - Otimized for discrete processing and microcontrollers.
- Other processors used in the embedded world, such as
  - Risc-V based processors
  - PowerPC
  - ...

Hardware Accelerators

- Piece of hardware specifically made to implement a specific function
  - Cryptography
  - Artificial Intelligence/ Machine learning
  - Digital signal processing
  - …
- Hardware is usually more efficient than software
  - More performance
  - Less energy / power

Hardware Accelerators

- Hardware accelerators are coupled with software in the microprocessor
  - Hardware functions implement the most computationally intense parts
    - Trade-off among flexibility, hardware area, and performance
  - Reasons
    - Hardware can be much more efficient than software
    - Hardware is not as flexible as software is
- Current trends
  - Hardware accelerators for common functions are included in most microprocessors
  - Use of GPUs
  - Reconfigurable hardware (FPGA)
    - You can implement your own accelerator

Embedded Operating Systems

- OSes are very different from system to system, depending on the task
  - (Customized) general purpose OSes
    - Linux
    - Android
    - Windows IoT
  - Lightweight OSes
    - mBed OS
    - Contiki
    - Android Things

Embedded Operating Systems

- Realtilme OSes
  - RTLinux
  - RTEMS
  - Contiki
  - mBed-rtos
- No OS
  - Arduino (classic devices)
  - ...

Embedded Application Software

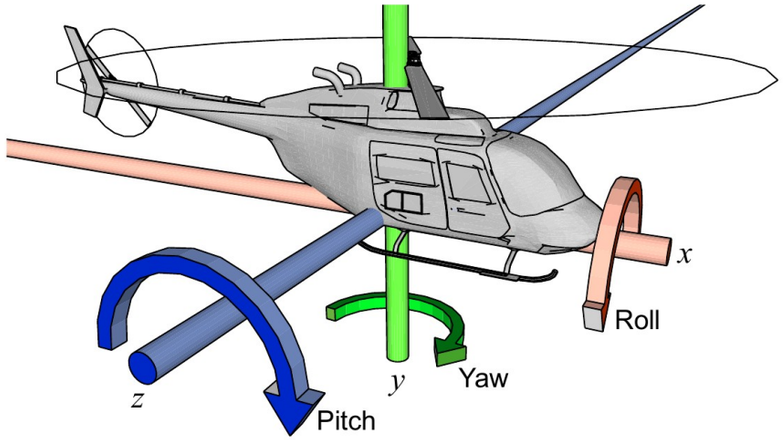
- Very difficult to define categories, as applications are potentially infinite and diverse
- We can identify some macro-categories that are representative:
  - Sensors monitoring and actuation
    - Simple (e.g., on-off control)
    - Control Engineering based
  - Digital signal processing
  - AI/Machine Learning
- Our system may include a mix of these macro-categories

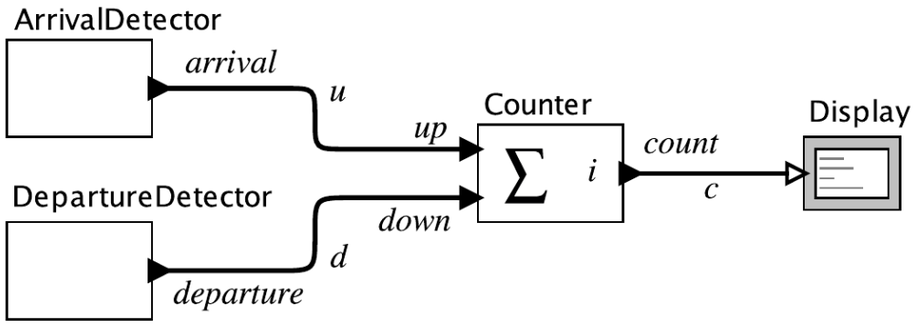
Embedded Software Applications – Sensor Monitoring and Simple Actuation

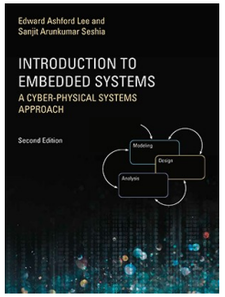
- Monitoring of certain parameters by means of sensors (e.g., temperature)
- Simple processing and / or communication of data to other devices
- Use of actuators with a simple control algorithm
- E.g., a sensor node that collects data on temperature and humidity and sends it to a remote server
- E.g., a device controlling lights in a room: if the room is too dark and the time of day is between 14:00 and 18:00, switches the light on

Types Of Systems

- Continuous dynamics:
  - Dynamics of physical systems, such as chemical processes, biological processes, mechanical systems
- Discrete dynamics:
  - A discrete system operates in a sequence of discrete steps.
  - Some systems are inherently discrete.





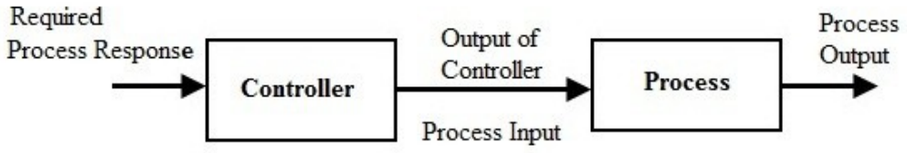


Continuous Dynamics Control

- In continuous dynamics control systems we require:
  - A model of the physical system
  - A controller that is suitable for the model
- We need to cope a system that is continuous in time with a controller that is discrete in time

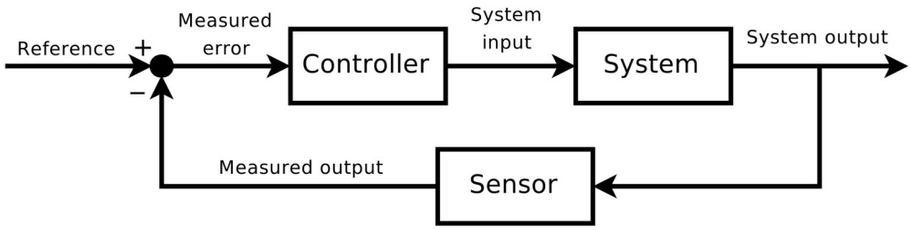
Embedded Software Applications – Control Systems Engineering

- Two types of control
  - Open loop
    - The control action from the controller is independent of the process output



https://www.elprocus.com/difference-between-open-loop-closed-loop-control-system/

- Closed loop
- The control action from the controller is dependent on feedback from the process



By Orzetto - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=5000019

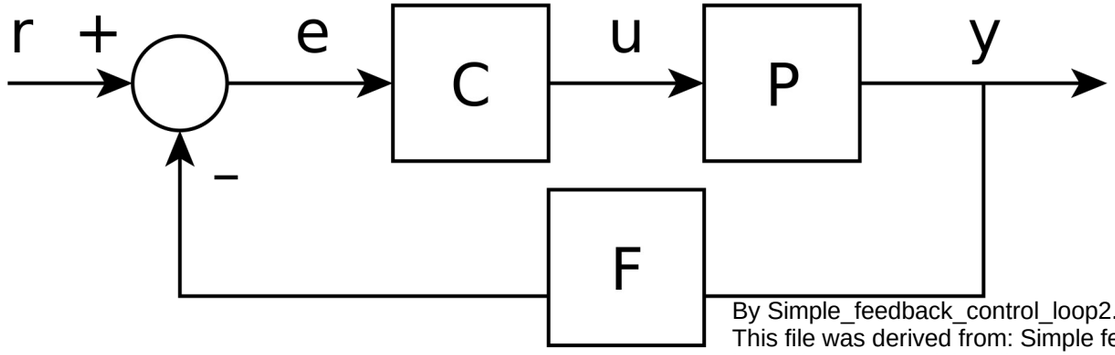
Embedded Software Applications – Control Systems Engineering – Example

- Let’s consider e mechanical arm actuated by a step-by-step motor (an electric motor that divides a full rotation into a number of equal steps)
- These types of motors are normally controlled in open loop
  - The command is “move by X steps” (equivalent to Y degrees)
    - There is no check and no guarantee that the reached position is the desired one
      - Errors
      - Initial calibration
      - Accumulated position errors
- A closed loop control would require sensors to check the real position
  - Control is performed in such a way that the correct position is reached despite of actuation errors

Embedded Software Applications – Control Systems Engineering – Example

- Let’s consider an electric heater for water (e.g., a boiler)
- We can control it in open loop:
  - The heater is activated for X minutes
    - Final water temperature is not precise and may be in a large range, depending on:
      - Initial and ambient temperature
      - Boiler efficiency (e.g., limestone can limit efficiency)
      - Electric current (e.g., departures from nominal values of the electric power network)
- A closed loop control would require a temperature sensor:
  - Control is performed in such a way that the correct temperature is reached regardless the external conditions (within the functional limits of the device)
  - Proper control may guarantee that the temperature is maintained with a predetermined precision

Embedded Software Applications – Control Systems Engineering – Closed Loop Transfer Function

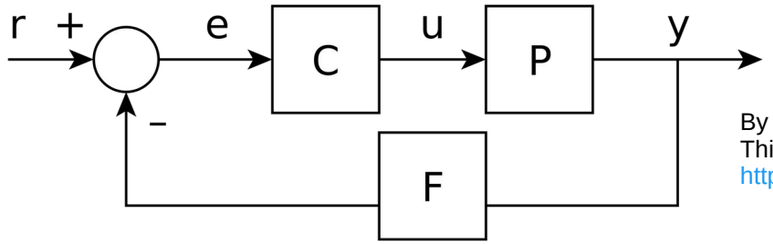


By Simple_feedback_control_loop2.png: Coronaderivative work: Rehua (talk) - This file was derived from: Simple feedback control loop2.png:, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=18126556

- The output of the system y(t) is fed back through a sensor measurement F to a comparison with the reference value r(t)
- The controller C uses the error e between the reference and the output to change the inputs u to the system under control P
  - This control scheme can be adapted to take into account disturbances (e.g., accuracy problems of sensors)

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

Embedded Software Applications – Control Systems Engineering – Closed Loop Transfer Function

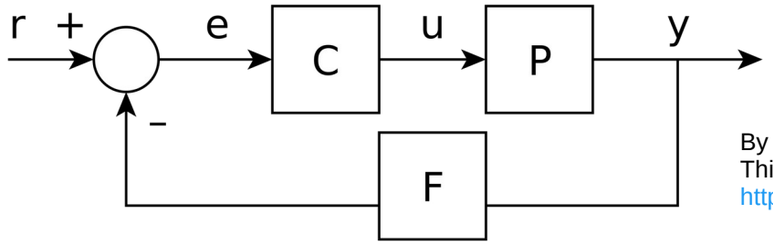


By Simple_feedback_control_loop2.png: Coronaderivative work: Rehua (talk) - This file was derived from: Simple feedback control loop2.png:, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=18126556

- Single-input-single-output (SISO) control system / MIMO (i.e., Multi-Input-Multi-Output) system
- We assume the controller C, the plant P, and the sensor F are
  - Linear
  - Time-invariant

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

Embedded Software Applications – Control Systems Engineering – Closed Loop Transfer Function



The systems can be analyzed by using the Laplace transform on the variables

- An integral transformation that transforms a function of a real variable *t* (often time) to a function of a complex variable *s* (complex frequency)

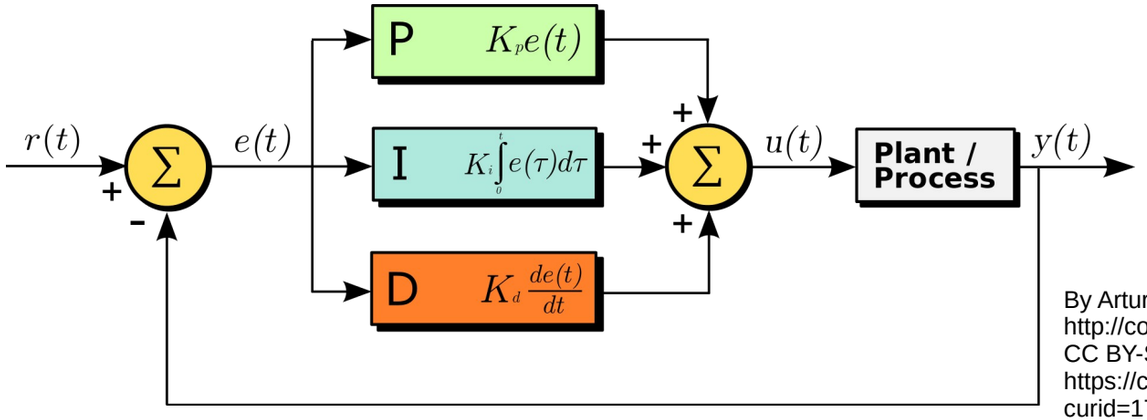
$$Y(s) = \left(\frac{P(s)C(s)}{1 + P(s)C(s)F(s)} \right) R(s) = H(s)R(s).$$

$$H(s) = \frac{P(s)C(s)}{1 + F(s)P(s)C(s)}$$

- The numerator is the forward (open-loop) gain from r to y
- The denominator is one plus the gain in going around the feedback loop, the so-called loop gain

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

Embedded Software Applications – Control Systems Engineering – PID Controller

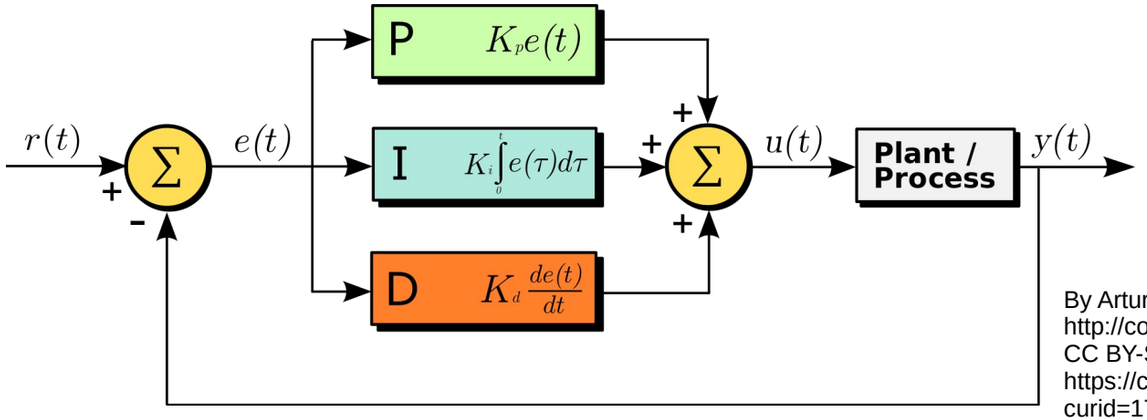


By Arturo Urquizo - http://commons.wikimedia.org/wiki/File:PID.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17633925

- A proportional–integral–derivative controller (PID controller) is a widely-used control technique
- It applies a correction based on proportional, integral, and derivative terms

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

Embedded Software Applications – Control Systems Engineering – PID Controller



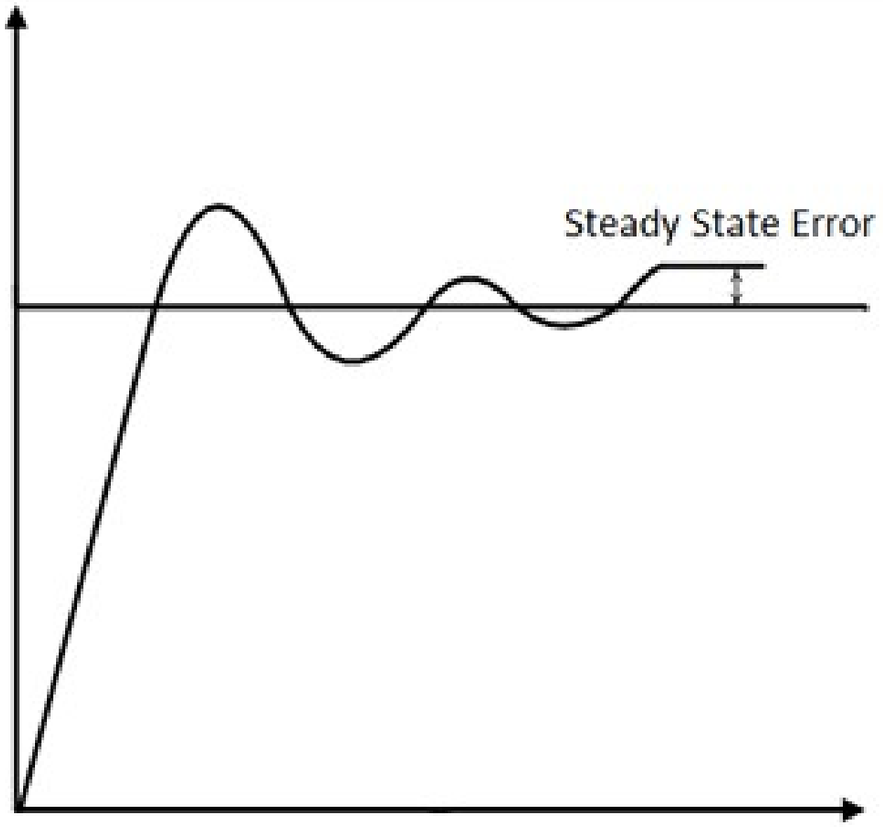
By Arturo Urquizo - http://commons.wikimedia.org/wiki/File:PID.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17633925

- The desired closed loop dynamics is obtained by adjusting the three parameters, often iteratively, by "tuning" and without specific knowledge of the process model
  - K_P: influence of the proportional therm (control is proportional to the error)
  - K_I: influence of the integral therm (control proportional to the integral of the error: not only to the error but also the time for which it has persisted)
  - K_D: influence of the derivative therm (control proportional to the derivative of the error: the rate of change of erro)
- Some applications use only one or two terms to provide the appropriate control by setting the unused parameters to zero

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

Embedded Software Applications – Control Systems Engineering – PID Controller Tuning

- Tuning is about setting the K_P,K_I, and K_D parameters to obtain the desired system response

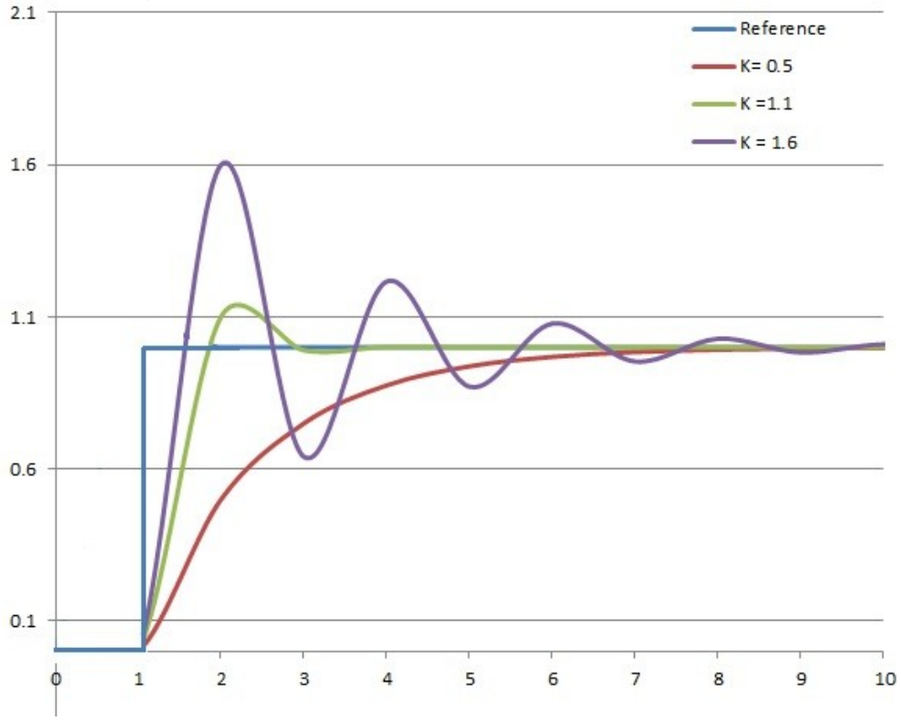


https://circuitdigest.com/article/what-is-pid-controller-working-structure-applications

Embedded Software Applications – Control Systems Engineering – PID Controller

- A high proportional gain results in a large change in the output for a given change in the error
  - Makes use of the present (current error)
  - If the proportional gain is too high, the system can become unstable
- A small gain results in a small output response to a large input error, and a less responsive or less sensitive controller

Response to step change of input vs time, for three values of Kp (Ki and Kd held constant)



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

Embedded Software Applications – Control Systems Engineering – PID Controller

- A pure "I" controller could bring the steady state error to zero, but it is slow reacting
  - Makes use of the past (accumulated error)
  - Since the integral term responds to accumulated errors from the past, it can cause the present value to overshoot the setpoint value
    - Some amount of overshoot is always necessary for a fast system so that it could respond to changes immediately

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

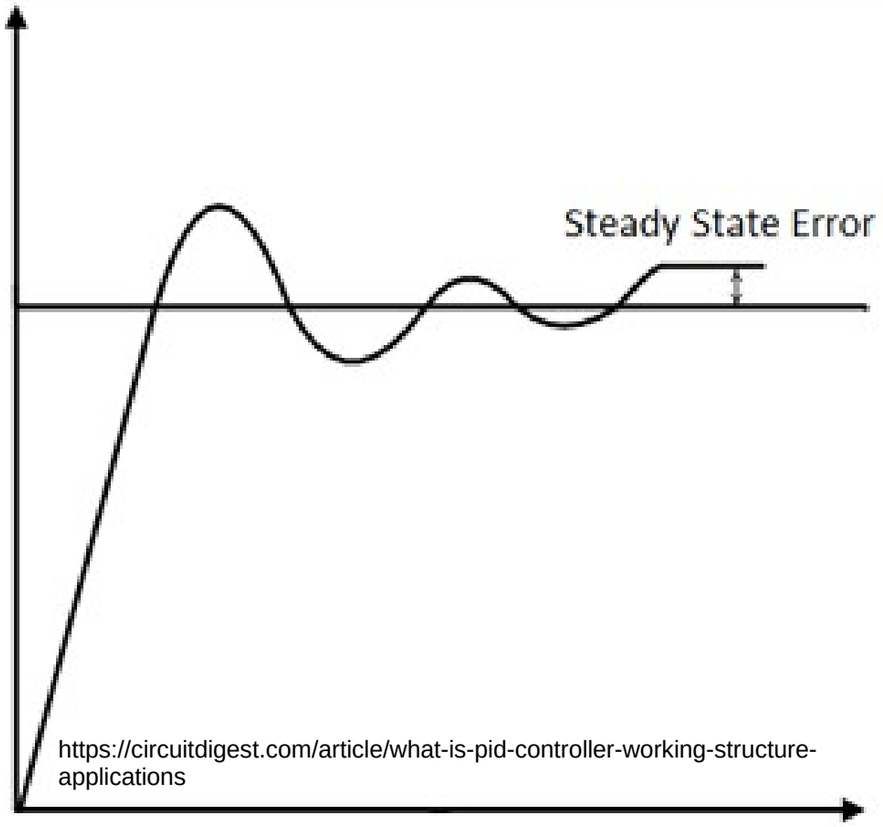
Embedded Software Applications – Control Systems Engineering – PID Controller

- The derivative controller aims at flattening the error trajectory into a horizontal line and so it reduces overshoot
  - Makes use of the predicted future error (error trend)
  - Decreases overshoot and yields higher gain with stability
  - Causes the system to be highly sensitive to noise
  - Seldom used in practice

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

Embedded Software Applications – Control Systems Engineering – PID Controller Tuning

- Given the system model, the gains can be computed
- Trial and error methods can be used to tune the gains
  - K_I and K_D set to zero
  - Choose K_p to obtain the desired response time
  - Increase K_D to reduce overshoot
  - K_I is tweaked to achieve a minimal steady state error
  - Check, for example, https://pidexplained.com/how-to-tune-a-pid-controller/ for more in depth info



https://circuitdigest.com/article/what-is-pid-controller-working-structure-applications

https://www.ni.com/en-ie/innovations/white-papers/06/pid-theory-explained.html#section--423985065
https://www.machinedesign.com/sensors/introduction-pid-control

Embedded Software Applications – Control Systems Engineering – PID Controller Tuning

- Ziegler-Nichols: a heuristic method of tuning:
  - K_i and K_D are initially set to zero
  - K_p is increased until it reaches the ultimate gain K_u at which the output of the control loop has stable and consistent oscillations
  - K_u and the oscillation period T_u are then used to set K_i and K_D depending on the type of controller used and behavior desired

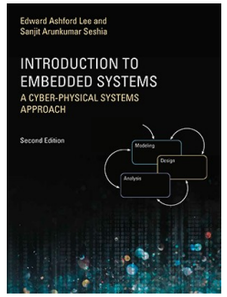
Control Type	K_p	T_i	T_d	K_i	K_d
P	$0.5K_u$	–	–	–	–
PI	$0.45K_u$	$0.80T_u$	–	$0.54K_u/T_u$	–
PD	$0.8K_u$	–	$0.125T_u$	–	$0.10K_uT_u$
classic PID[2]	$0.6K_u$	$0.5T_u$	$0.125T_u$	$1.2K_u/T_u$	$0.075K_uT_u$

- Simulator: http://grauonline.de/alexwww/ardumower/pid/pid.html
- ArduPID for Arduino
  - There exist sample codes for autotuners based on Ziegler-Nichols

https://en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method
https://www.ni.com/en-ie/innovations/white-papers/06/pid-theory-explained.html#section--423985065
https://www.machinedesign.com/sensors/introduction-pid-control

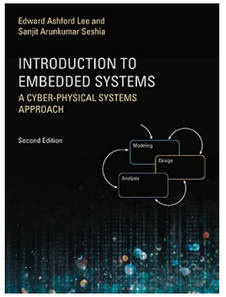
Discrete Dynamics

- In discrete dynamics systems we deal with the state of the system
- Intuitively, the state of a system is its condition at a particular point in time
- The state affects how the system reacts to inputs
- We define the state to be an encoding of everything about the past that has an effect on the system's reaction to current or future inputs



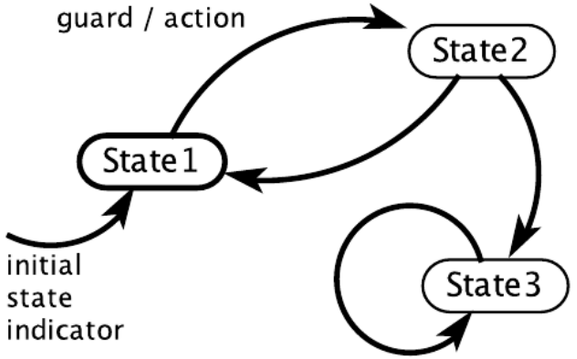
Discrete Dynamics – Finite State Machines

- Discrete dynamics systems are naturally represented by finite state machines
- A state machine is a model of a system that at each reaction maps valuations of the inputs to valuations of the outputs
  - The map may depend on its current state
- A finite-state machine (FSM) is a state machine where the set of possible states is finite
- FSM models are amenable to formal checking

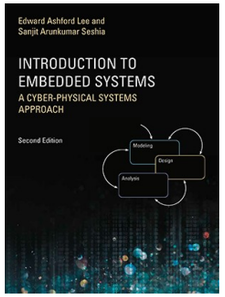


Finite State Machines

- Transitions between states govern the discrete dynamics of the state machine and the mapping of input valuations to output valuations
- A guard is a predicate (a boolean-valued expression) that evaluates to true when the transition should be taken
- When a guard evaluates to true we say that the transition is enabled
- An action is an assignment of values (or absent) to the output ports

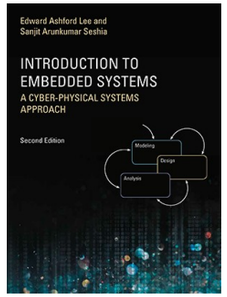


- Any output port that is not mentioned in a transition is implicitly absent
- We use the notation name := value to assign a value to an output port
- The assignment from a predicate is written name = value

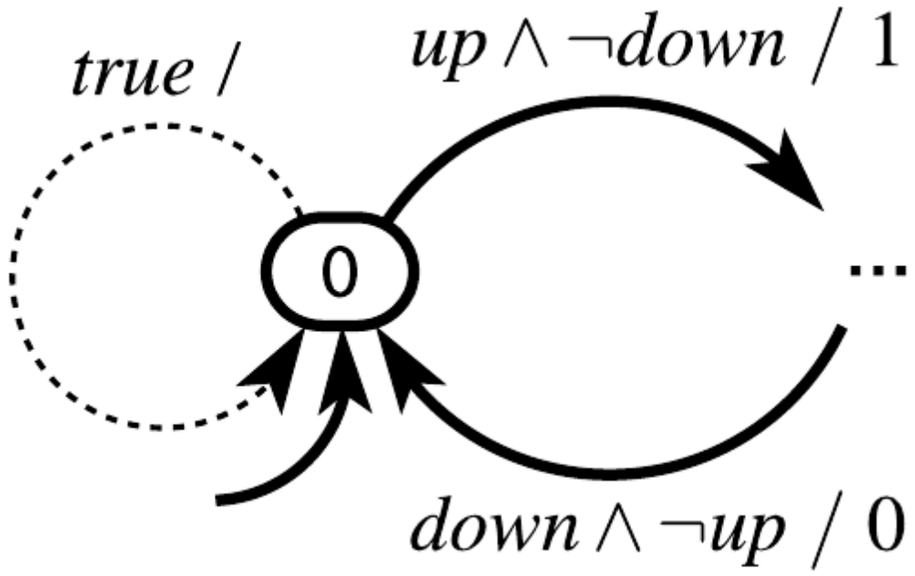


Finite State Machines

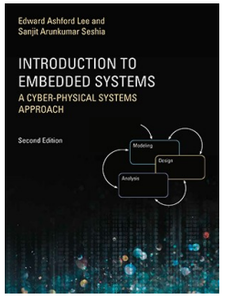
- An FSM can be
  - Event triggered
  - Time triggered, meaning that it reacts at regular time intervals
- Some state machines will have states that can never be reached:
  - the set of reachable states may be smaller than the set of states
- Inputs to FSM come from its environment



Finite State Machines

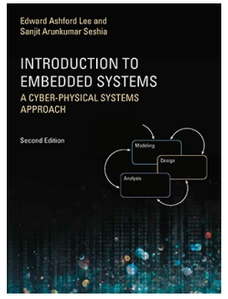


- Default transitions are denoted with dashed lines and labeled with true/
- A default transition define explicitly the behavior if no other transitions are enabled
  - An ordinary transition has priority over a default transition
- Default transitions provide a convenient notation, but they are not really necessary:
  - Any default transition can be replaced by an ordinary transition with an appropriately chosen guard



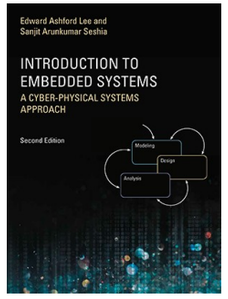
Nondeterminism

- If for any state of a state machine, there are two distinct transitions with guards that can evaluate to true in the same reaction, then the state machine is **nondeterministic**
- A nondeterministic FSM specifies a family of possible reactions rather than a single reaction
  - Operationally, all reactions in the family are possible
  - The nondeterministic FSM makes no statement at all about how likely the various reactions are
- Uses:
  - Environment Modeling: hiding irrelevant details about how an environment operates often results in nondeterminism
  - Specification: system specification impose requirements on some system features, while leaving other features unconstrained



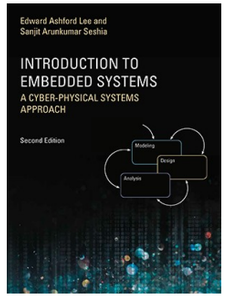
Moore Machines and Mealy Machines

- Mealy machines: output is defined by the transition taken
- Moore machines: output is defined by the current state
  - At each reaction, the output produced is defined by the current state (at the start of the reaction)
    - The output at the time of a reaction does not depend on the input at that same time; the input determines which transition is taken, but not the output
- Any Moore machine may be converted to an equivalent Mealy machine
- A Mealy machine may be converted to an almost equivalent Moore machine that differs only in that the output is produced on the next reaction rather than on the current one



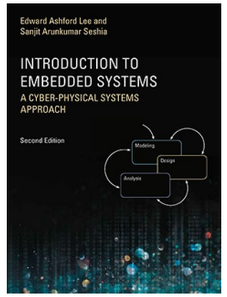
Hybrid Systems

- Cyber-physical systems integrate physical dynamics and computational systems, so they commonly combine both discrete and continuous dynamics: **hybrid systems**



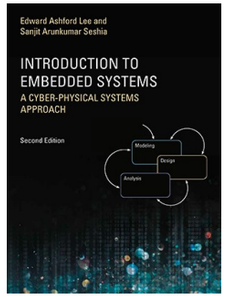
Hybrid Systems

- A control system involves four components:
  - A system called the plant: the physical process that is to be controlled
  - The environment in which the plant operates
  - The sensors that measure some variables of the plant and the environment
  - The controller that determines the mode transition structure and selects the time-based inputs to the plant



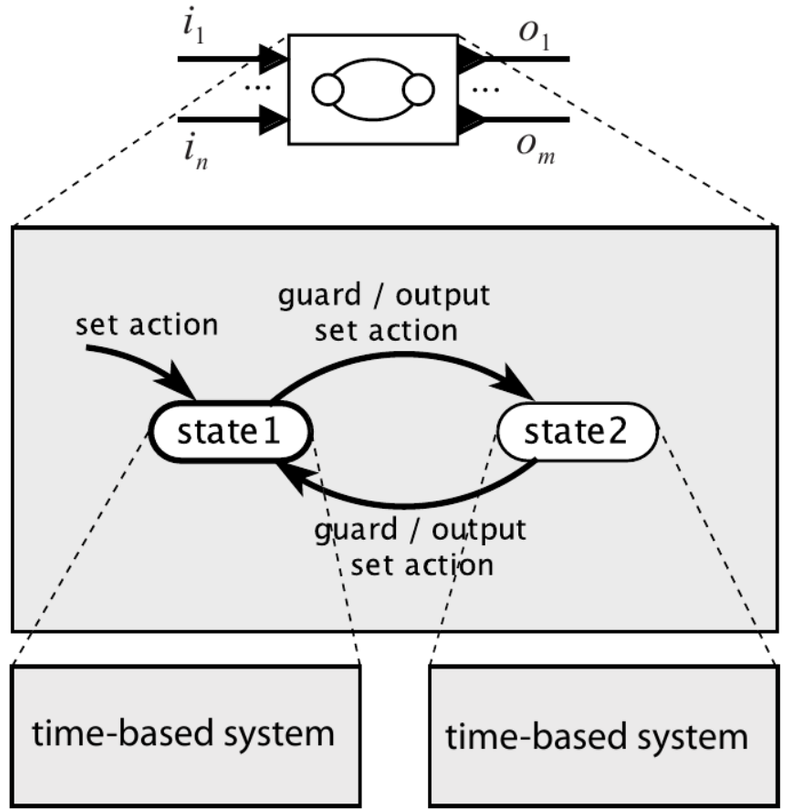
Hybrid Systems

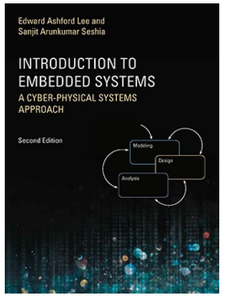
- The controller has two levels:
  - The supervisory control that determines the mode transition structure
  - The low-level control that determines the time-based inputs to the plant



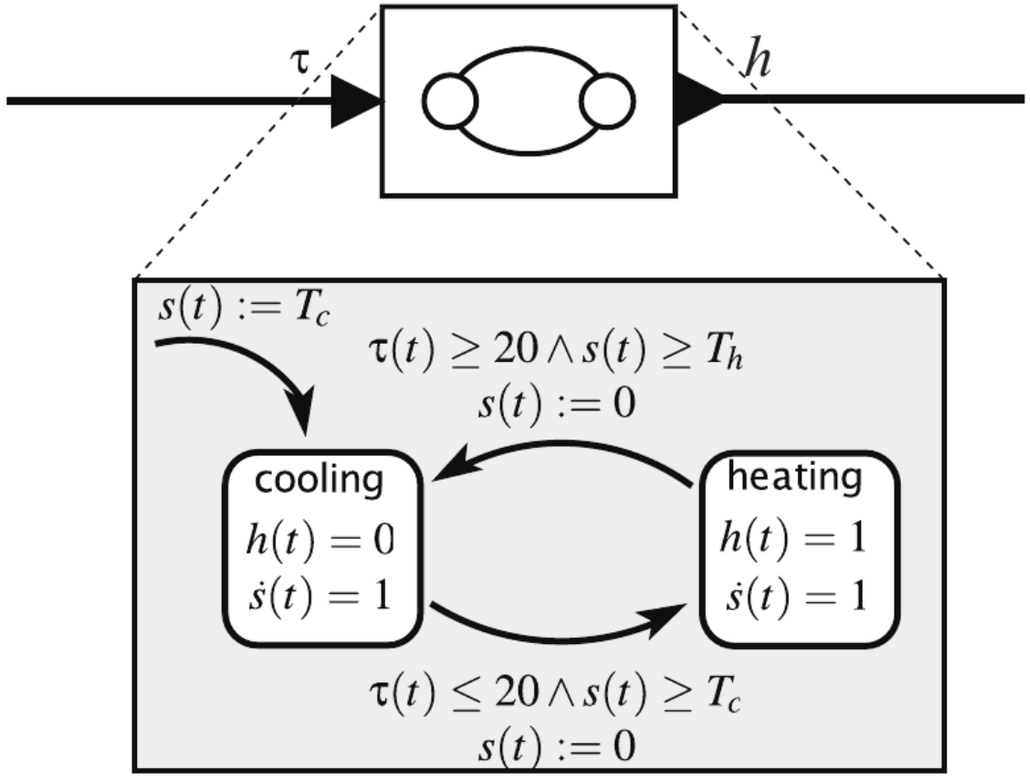
Hybrid Systems – State Refinement

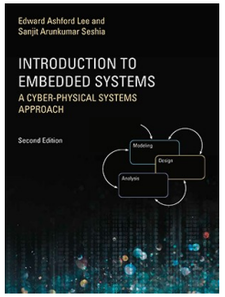
- The current state of the state machine has a state refinement that gives the dynamic behavior of the output as a function of the input





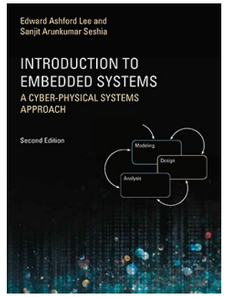
Hybrid Systems – Example





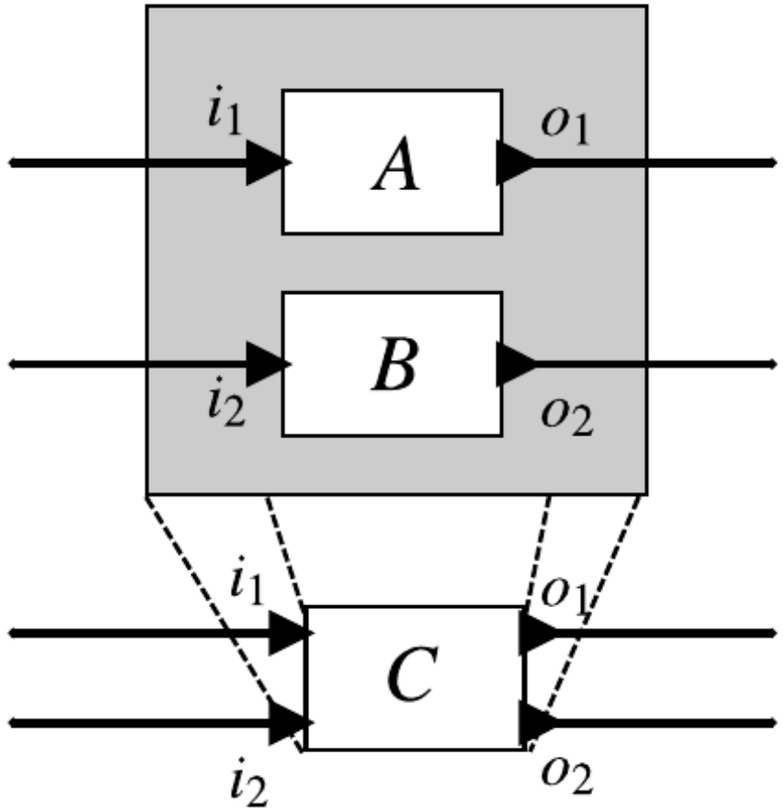
Composition Of State Machines

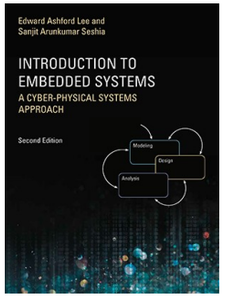
- Complex systems can be represented by a composition of state machines
  - We decompose a complex model into simpler ones
- Concurrent composition
- Hierarchical composition



Concurrent composition

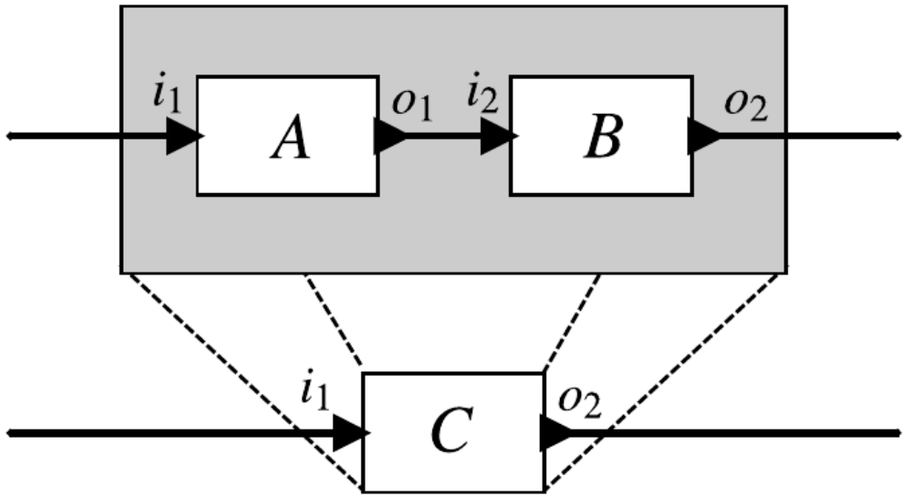
- Two or more state machines that react
  - Simultaneously: synchronous composition
  - Independently

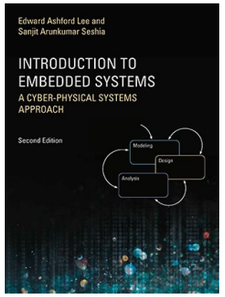




Cascade composition

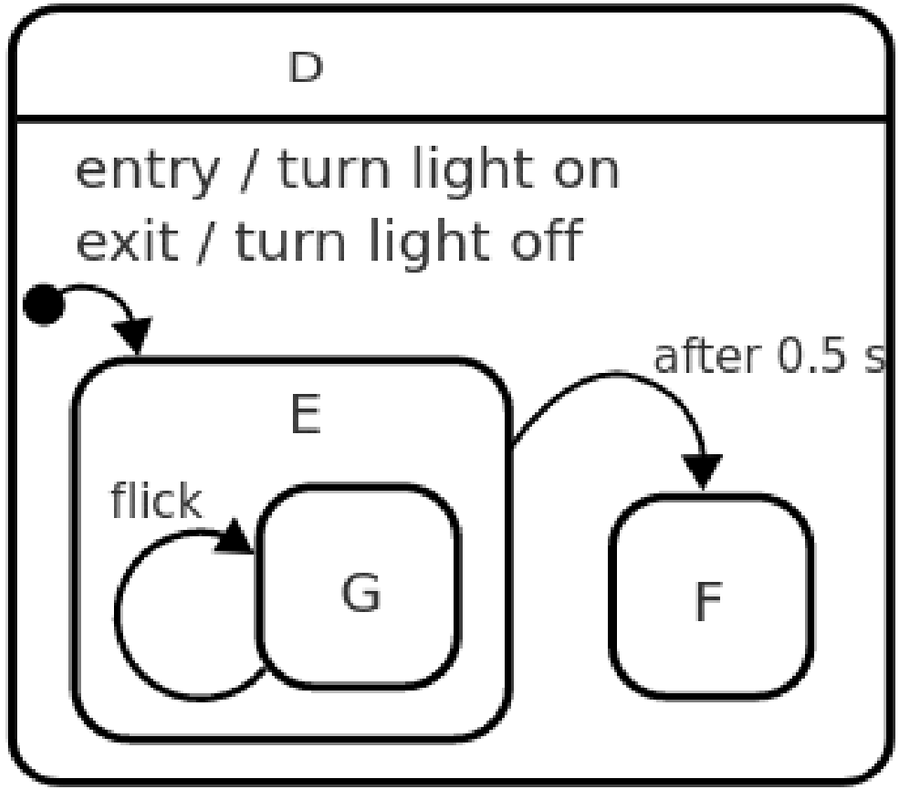
- Side-by-side synchronous/asynchronous composition



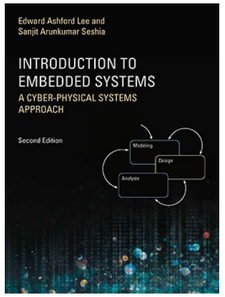


Hierarchical State Machines – Statecharts

- A statechart is a state machine where each state may define its own subordinate state machines, called substates
  - Those states can again define substates.
- When a state is entered, its sub state machine starts and therefore, a substate is entered
- When a state is exited, its sub state machine is exited too
- Part of UML (Unified Modeling Language)

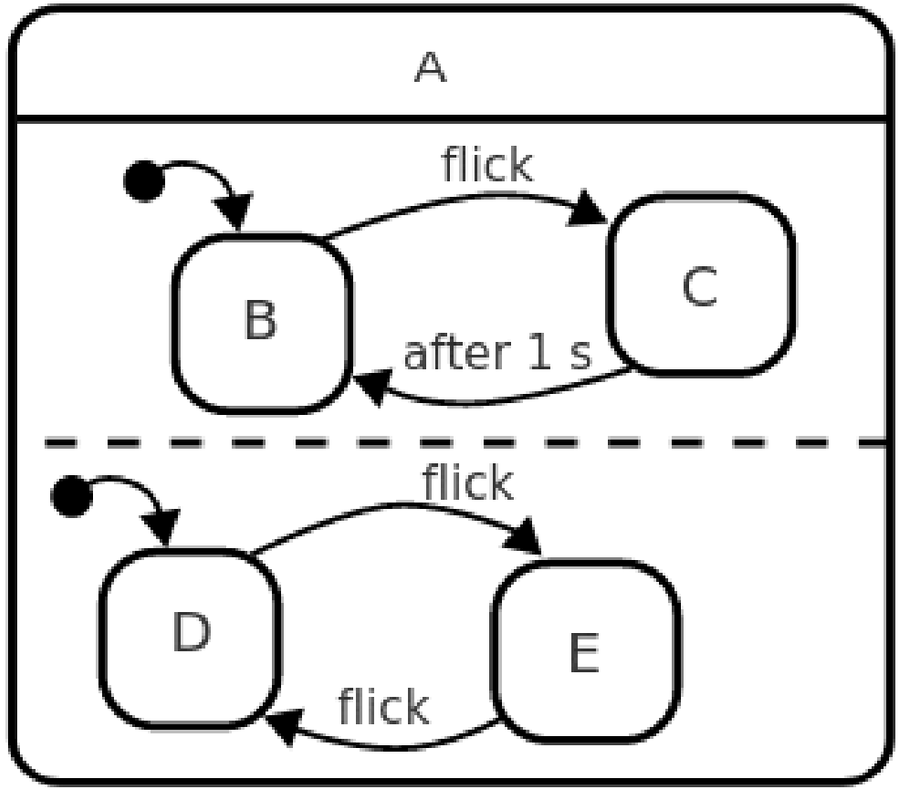


https://statecharts.github.io/what-is-a-statechart.html

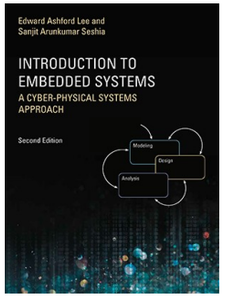


Hierarchical State Machines – Statecharts

- A compound state can be split up into completely separate (“orthogonal”) regions
  - Each region specifies its own state machine
  - When a state machine enters such a state, it also enters all of the regions at the same time
- Such a state is called a *parallel state*
- The regions are often called *orthogonal regions*



https://statecharts.github.io/what-is-a-statechart.html



What have we discussed in these classes?

- What embedded systems, cyber-physical systems, and IoT are
- Main peculiarities of embedded systems
- Continuous dynamics/discrete systems