

Robotic Agent Architectures

Robótica Móvel e Inteligente

José Luís Azevedo, Bernardo Cunha, Pedro Fonseca,
Nuno Lau e Artur Pereira

Ano Lectivo 2022/2023
IEETA – DETI – Universidade de Aveiro

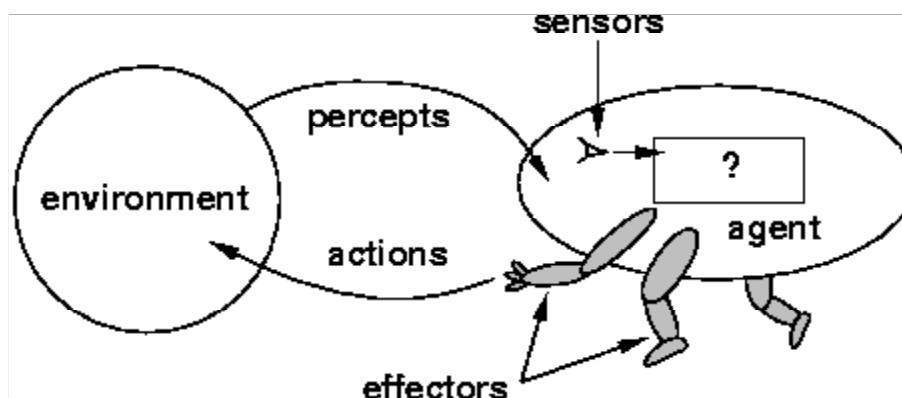
Outline

- Introduction to Robotic Agents
- Deliberative Architectures
- Reactive Architectures
- Behavior-Based Architectures
 - Subsumption Architecture
- Hybrid Architectures

Autonomous Agents

- **Traditional Definition:**

“Computational System, situated in a given **environment**, that has the ability to **perceive** that environment using **sensors** and **act**, in an **autonomous way**, in that environment using its **actuators** to fulfill a given **function**.”



Russel and Norvig, AI: Modern Approach

Agent Requisites

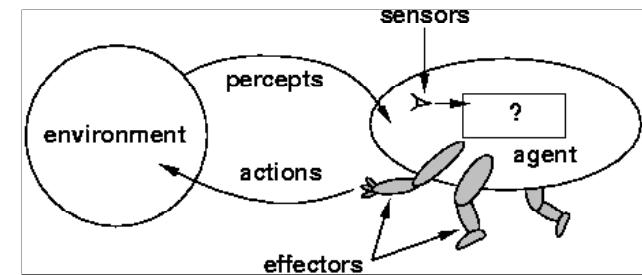
- Traditional definition include too much or leaves “holes”!
- **Requisites:**
 - Perceive its environment (sensors)
 - Decide actions to execute (“think”)
 - Execute actions in environment using its actuators
 - Communicate?
 - Perform a complex function?
- **Agents vs Objects:**
 - Agents decide what to do
 - Object methods are called externally
 - Agents react to sensors and control actuators

“Objects do it for free; agents do it for money”

Robotic and Human Agents

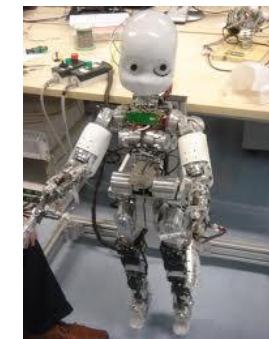
- **Agent:**

- Perceive its environment using sensors and executes actions using its actuators
- Sensors:
 - Eyes, ears, nose, touch, ...
- Actuators:
 - Legs, Arms, hands, vocal cords, ...



- **Robotic Agent:**

- Sensors:
 - Cameras, sonar, infra-red, microphone
- Actuators:
 - Motors, wheels, manipulators, speakers



- **Robotics**

- Science and technology for **projecting, building, programming and using Robots**
- Study of **Robotic Agents (with body)**
- Increased Complexity:
 - **Environments:** Dynamic, Inaccessible, Continuous and Non Deterministic!
 - **Perception:** Vision, Sensor Fusion
 - **Action:** Robot Control
 - **Robot Architecture (Physical / Control)**
 - **Navigation** in unknown environments
 - **Interaction** with other robots/humans
 - **Multi-Robot Systems**



Definition of Robot

- Notion derives from 2 strands of thought:
 - Humanoids: human-like
 - Automata: self-moving things
- “Robot” - derives from Czech word *roboťa*
 - “*Robota*”: forced work or compulsory service
 - Term coined by Czech playwright Karel Čapek (1920)
- Current notion of robot:
 - Programmable
 - Mechanically capable
 - Flexible



Best Definitions of Robot

- Electromechanical device which can perform tasks on its own, or with guidance
- Physical agent (with body) that generates intelligent/autonomous connection between perception and action
- Autonomous system in the physical world which may sense its environment and act on it to achieve a set of goals

Robotic Architecture - Definition(s)

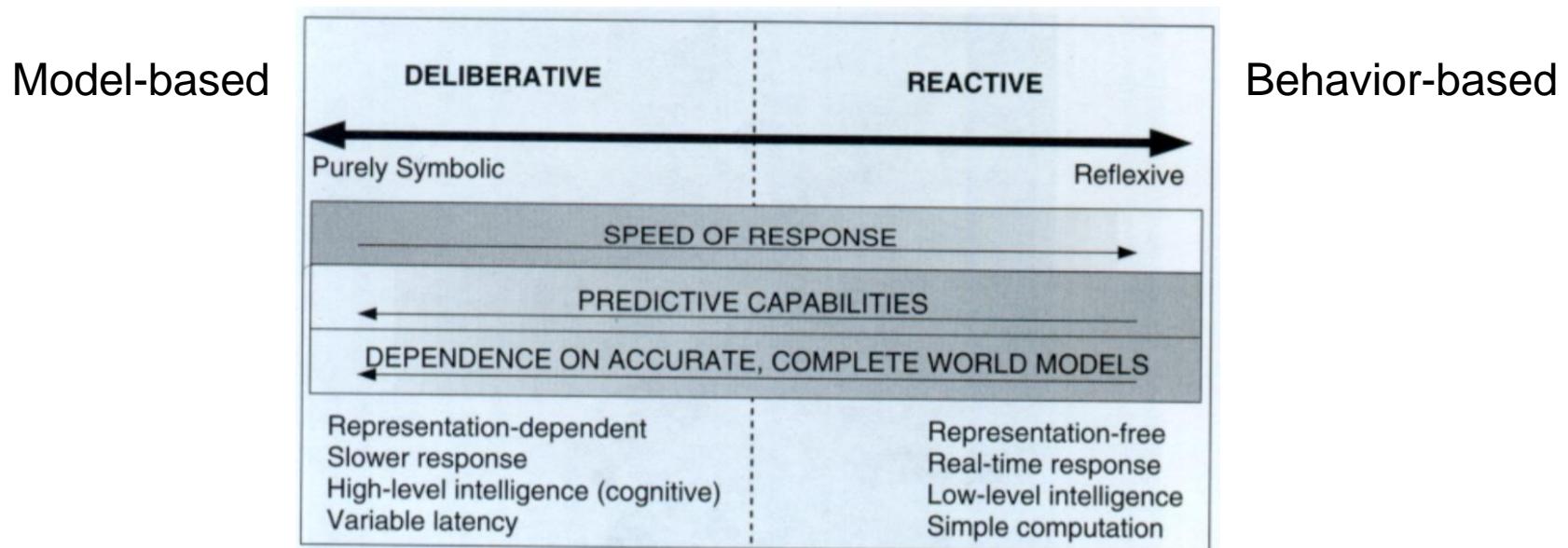
- An **architecture** provides a **principled way of organizing a control system**. However, in addition to **providing structure**, it **imposes constraints** on the way the control problem can be solved
[Mataric]
- An architecture is a description of how a **system is constructed from basic components** and how those **components fit together** to form the whole
[Albus]
- Robotic architecture usually refers to **software**, rather than hardware
[Arkin, 1998]
- How **the job** of generating actions from percepts **is organized**
[Russel and Norvig, 2002]

Issues in Robotic Architectures

- **Representation**
 - unified, heterogeneous, multiple or no representation
- **Control and coordination**
 - centralized or distributed control
- **Learning**
 - architecture should organize structures to facilitate learning
- **Timely performance**
 - deal with real-time constraints
- **Biological and psychological inspiration**
 - parallelism, distributed control, reflex loops, etc
- **Evaluation**

Spectrum of Robot Control Architectures

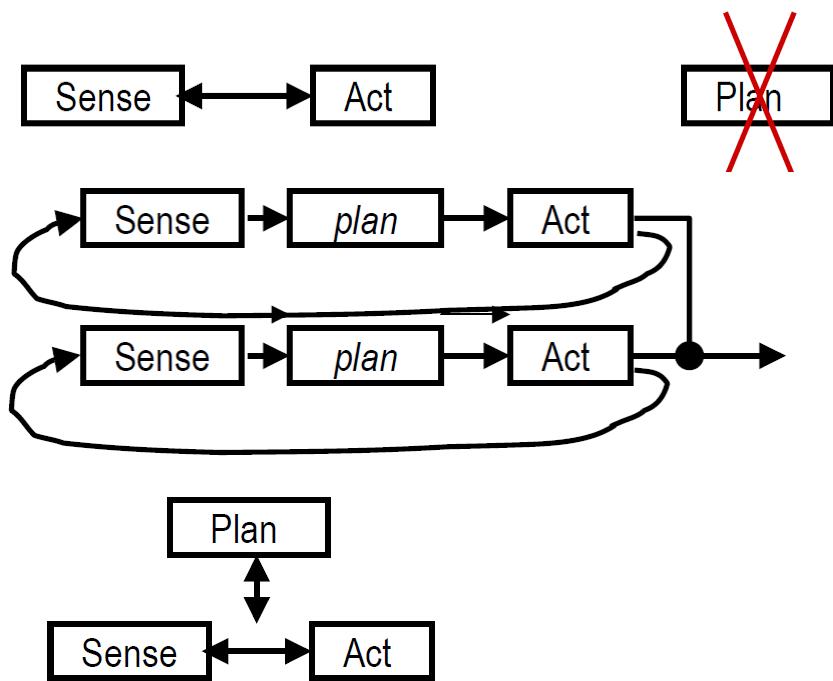
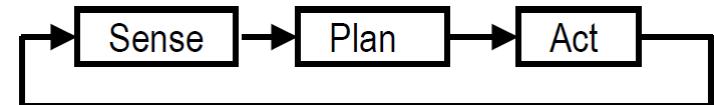
- Deliberative control: “think hard, then act”
- Reactive control: “don't think, (re)act”
- Hybrid control: “think and act in parallel”



Adapted from Arkin, Behavior-based Robotics (MIT Press, 1998)

Typical Organizations

- Typical organizations:
 - Hierarchical / Deliberative
 - Reactive
 - Behavior-based
 - Hybrid

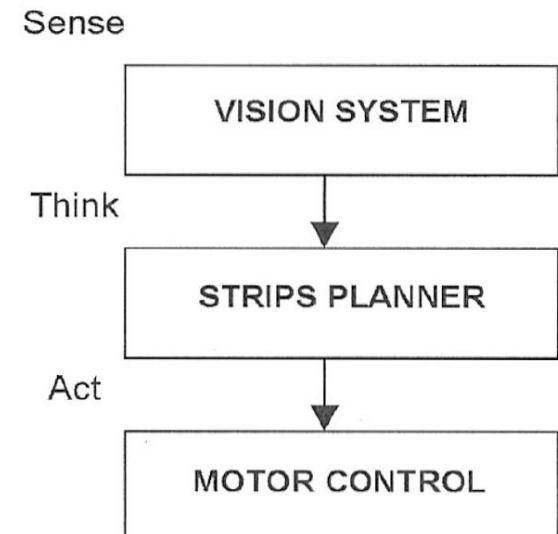


Typical Organizations

- Deliberative
 - Making maps
 - Selecting behaviors
 - Monitor performance
 - Planning
 - Hybrid deliberative/reactive paradigm
- Reactive
 - Cheap low memory processing
 - No world model
- Behavior-Based
 - Combination of simple behaviors
 - No centralized world model
 - Each behavior may store own representation
- Hybrid
 - Combine Reactive and Deliberative approaches

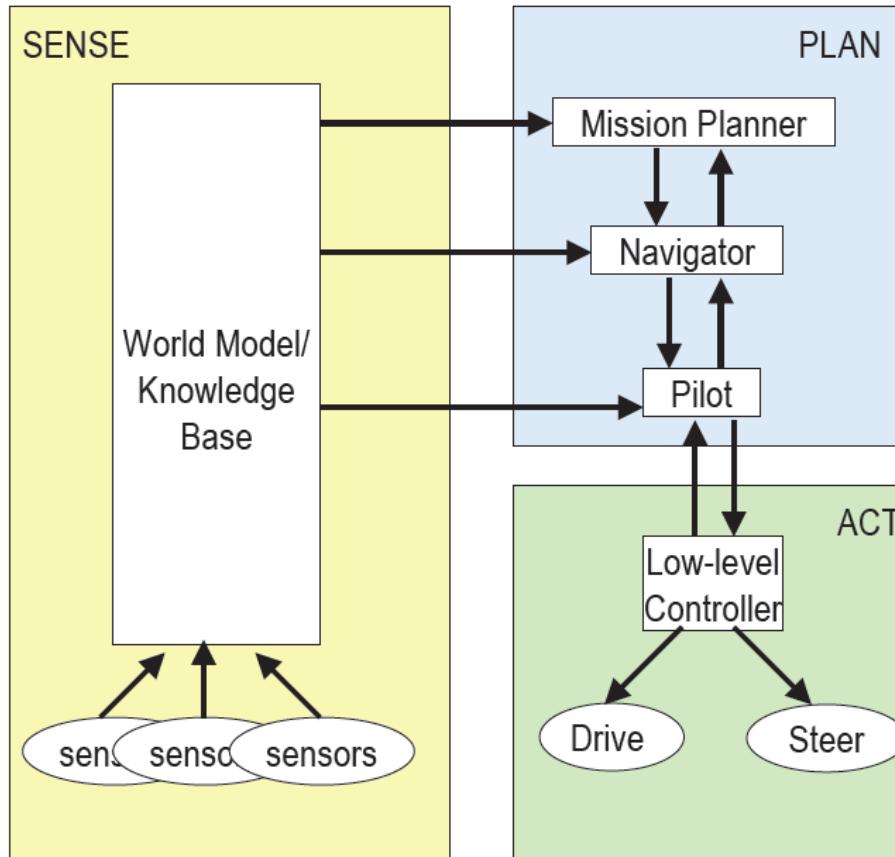
Model-based - Deliberative

- Sense-plan-act paradigm: dominant view in the AI community was that a control system for an autonomous mobile robot should be decomposed into three functional elements [Nilsson, 1980]:
 - a sensing system (translate raw sensor input into a world model)
 - a planning system (take the world model and a goal and generate a plan to achieve the goal)
 - and an execution system (take the plan and generate the actions it prescribes)
- Perception is the establishment and maintenance of correspondence between the internal world model and the external real world [Albus 1991].
- Action results from reasoning over the world model.
- **Perception is not directly tied to action.**



Deliberative Architectures

- Nested Hierarchical Controller



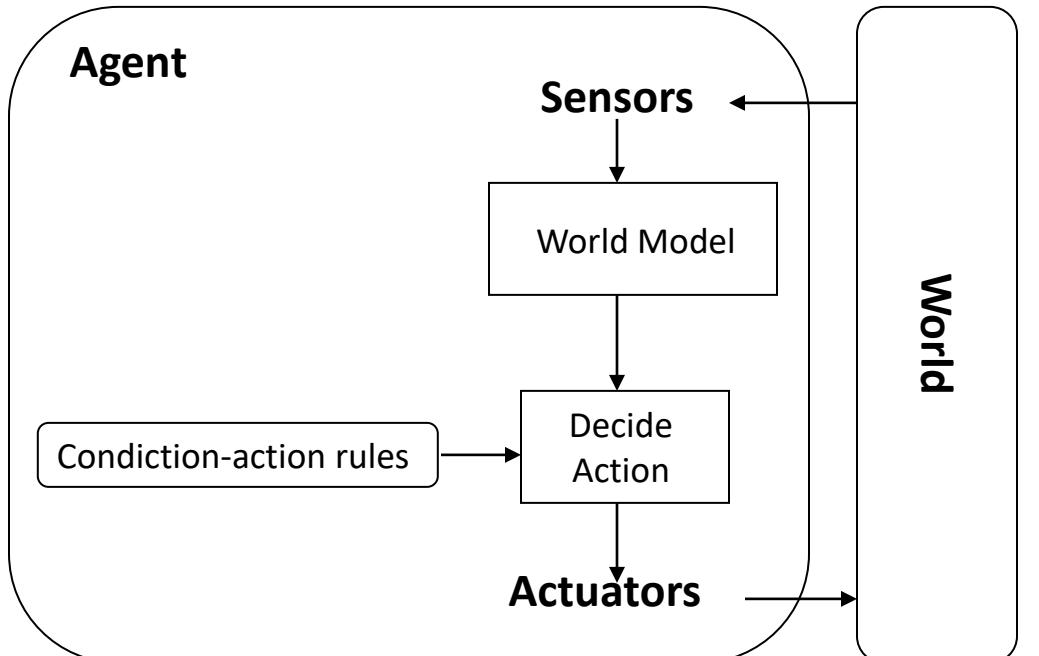
Meystel, A., "Knowledge Based Nested Hierarchical Control", 1990

Reactive Agents

- General assumptions:
 - The environment lacks temporal consistency and stability
 - The robot's immediate sensing is adequate for the task at hand
 - It is difficult to localize a robot relative to a world model
 - Symbolic representational world knowledge is of little or no value

“Planning is Just a Way of Avoiding Figuring Out What To Do Next”, Brooks 1987

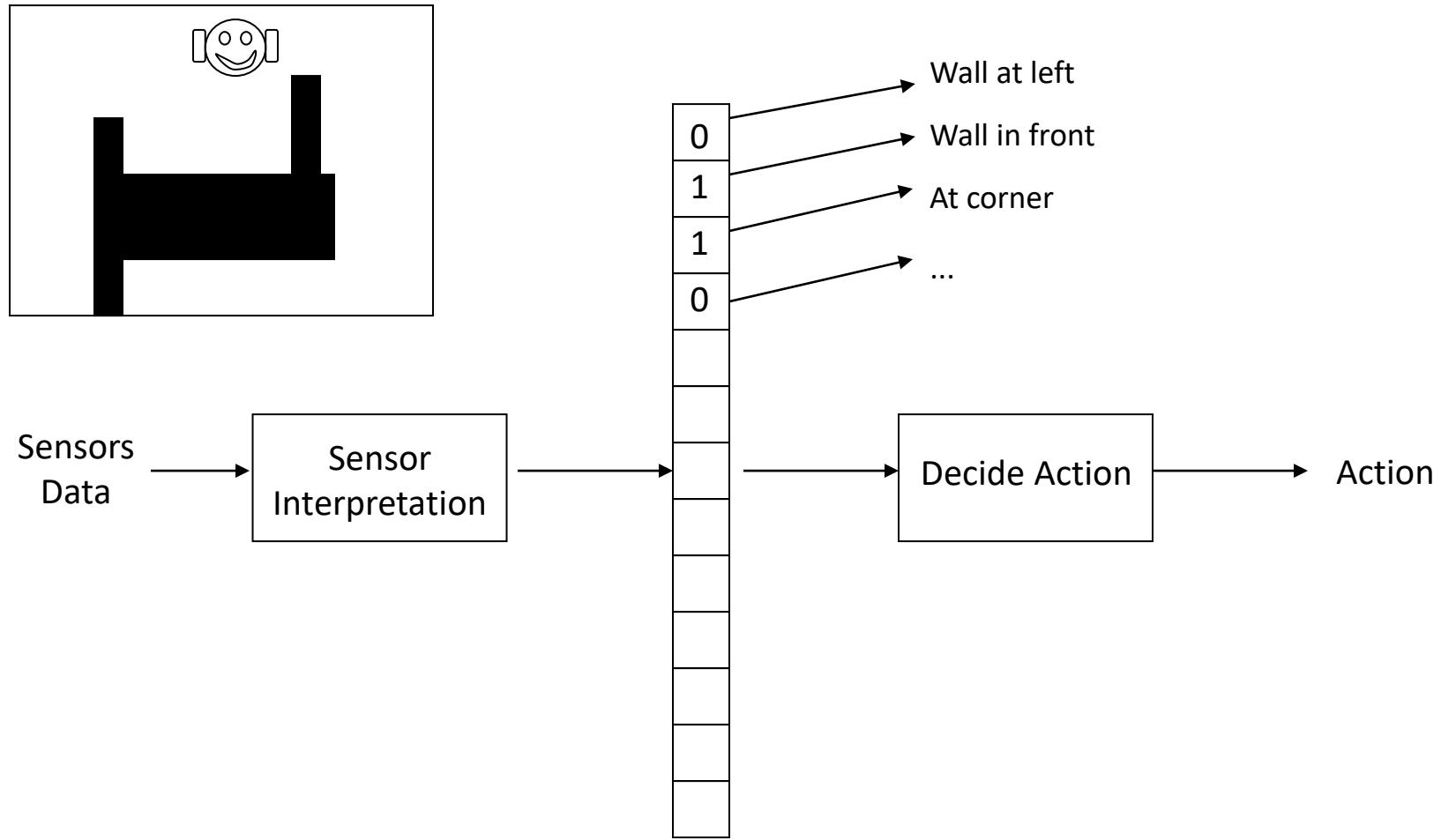
Simple Reactive Agent



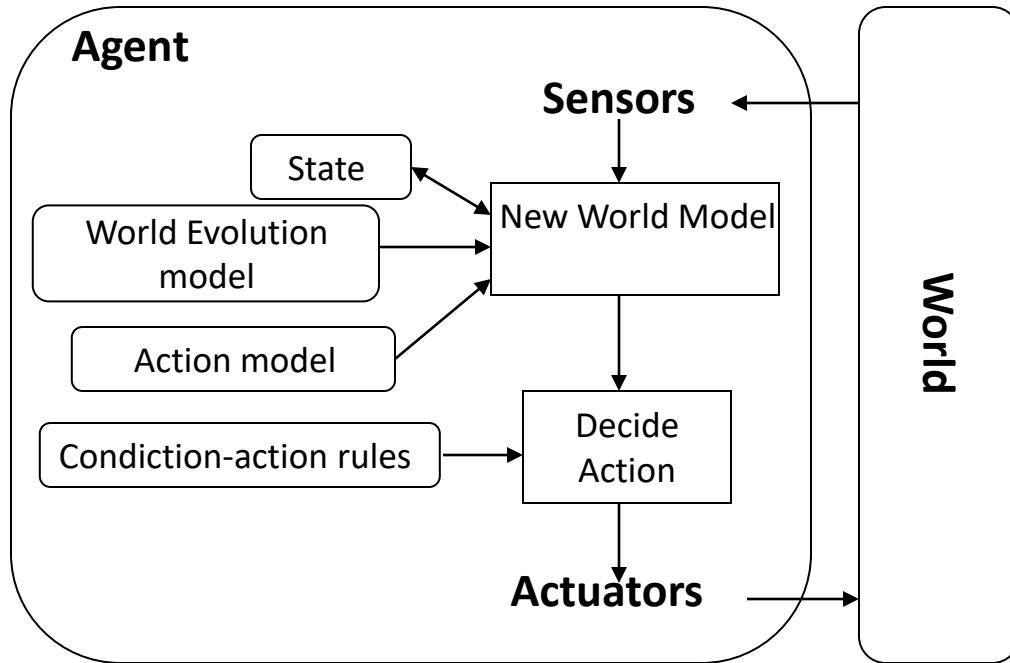
Russel and Norvig, AI: Modern Approach

Simple Reactive Agent

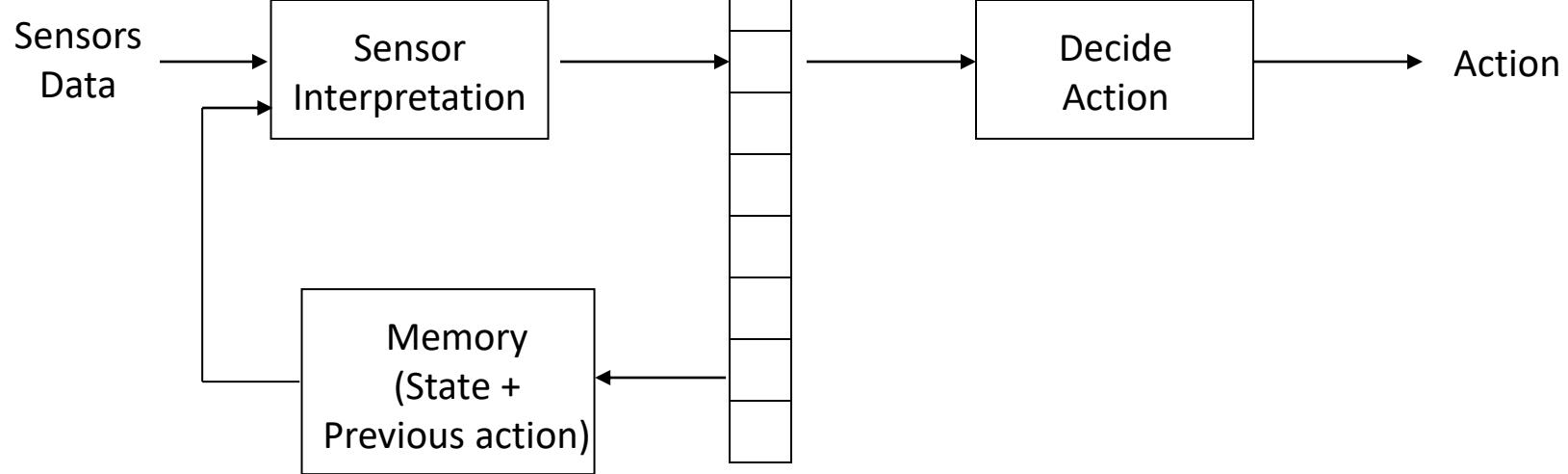
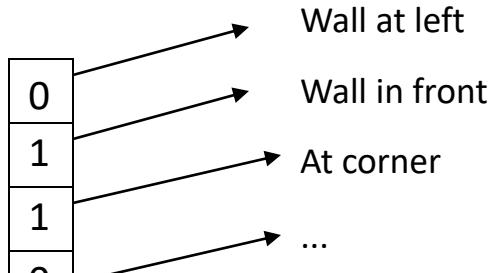
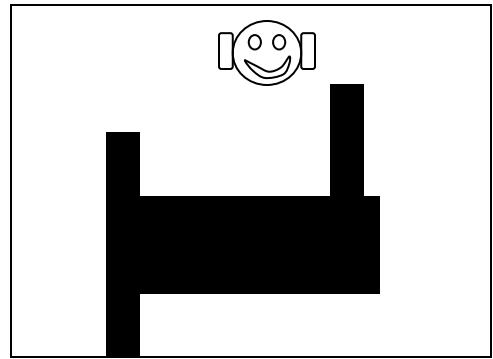
- Perception represented by a feature vector



Reactive Agent with Internal State



Reactive Agent with Internal State



Deliberative vs Reactive

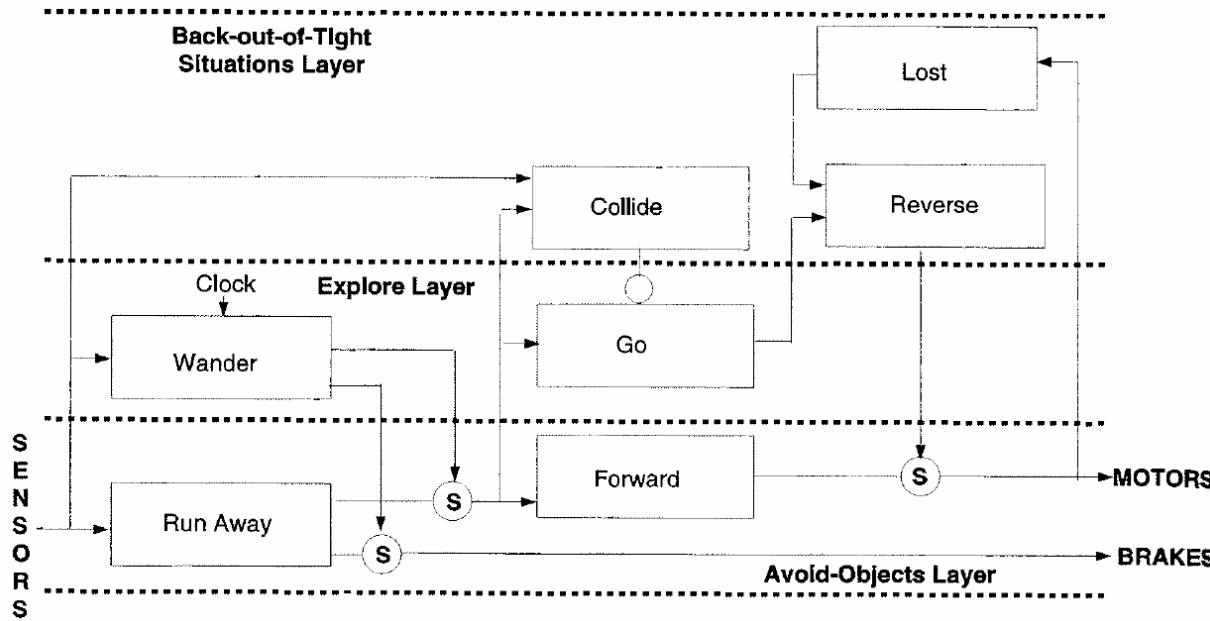
- No single approach is "the best" for all robots; each has its strengths and weaknesses
- Control requires some unavoidable trade-offs because:
 - Thinking is slow
 - Reaction must be fast
 - Thinking allows looking ahead (planning) to avoid bad actions
 - Thinking too long can be dangerous (e.g., falling off a cliff)
 - To think, the robot needs (a lot of) accurate information
 - The world keeps changing as the robot is thinking, so the slower it thinks, the more inaccurate its solutions
- As a result of these trade-offs, some robots don't think at all, while others mostly think and act very little.
 - **It all depends on the robot's task and its environment!**

Behavior-Based Architectures

- Behaviors implemented as control laws (in software or hardware)
- Each behavior receives inputs from the robot's sensors and/or from other modules and sends outputs to the robot's effectors and/or to other modules.
- Many different behaviors may receive input from the same sensors and output commands to the same actuators.
- Behaviors are encoded to be relatively simple and are added to the system incrementally.
- Behaviors (or subsets) are executed concurrently

Subsumption Architecture [Brooks 1986]

- Behaviors are Augmented Finite State Machines (AFSM)
- Stimulus or response signals can be suppressed or inhibited by other active behaviors; a reset input returns the behavior to its start conditions
- Each behavior is responsible for its own perception of the world
- Arrangement in layers: lower layers have no awareness of higher layers

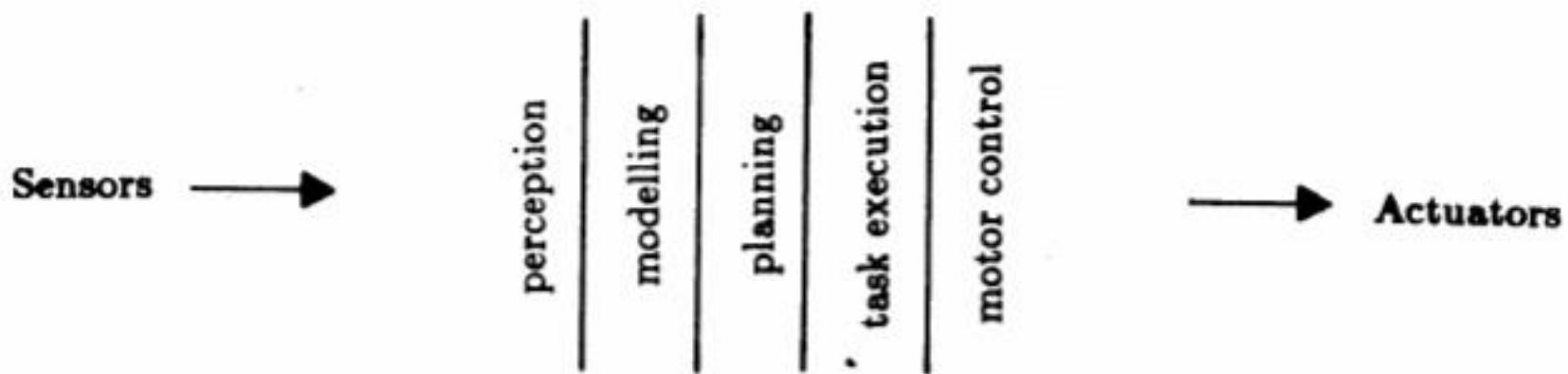


Brooks – Behavior languages

Brooks has put forward three theses:

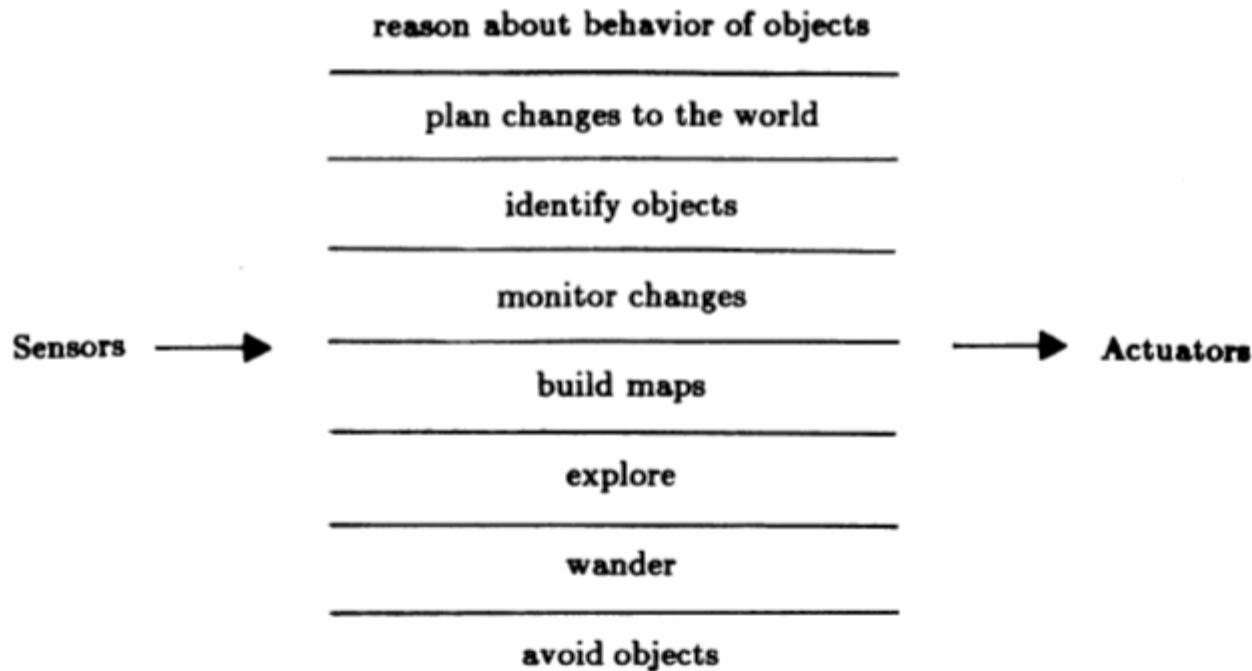
1. **Intelligent behavior** can be generated ***without explicit representations*** of the kind that symbolic AI proposes
2. **Intelligent behavior** can be generated ***without explicit abstract reasoning*** of the kind that symbolic AI proposes
3. **Intelligence is an *emergent property*** of certain complex systems

A Traditional Decomposition of a Mobile Robot Control System into Functional Modules



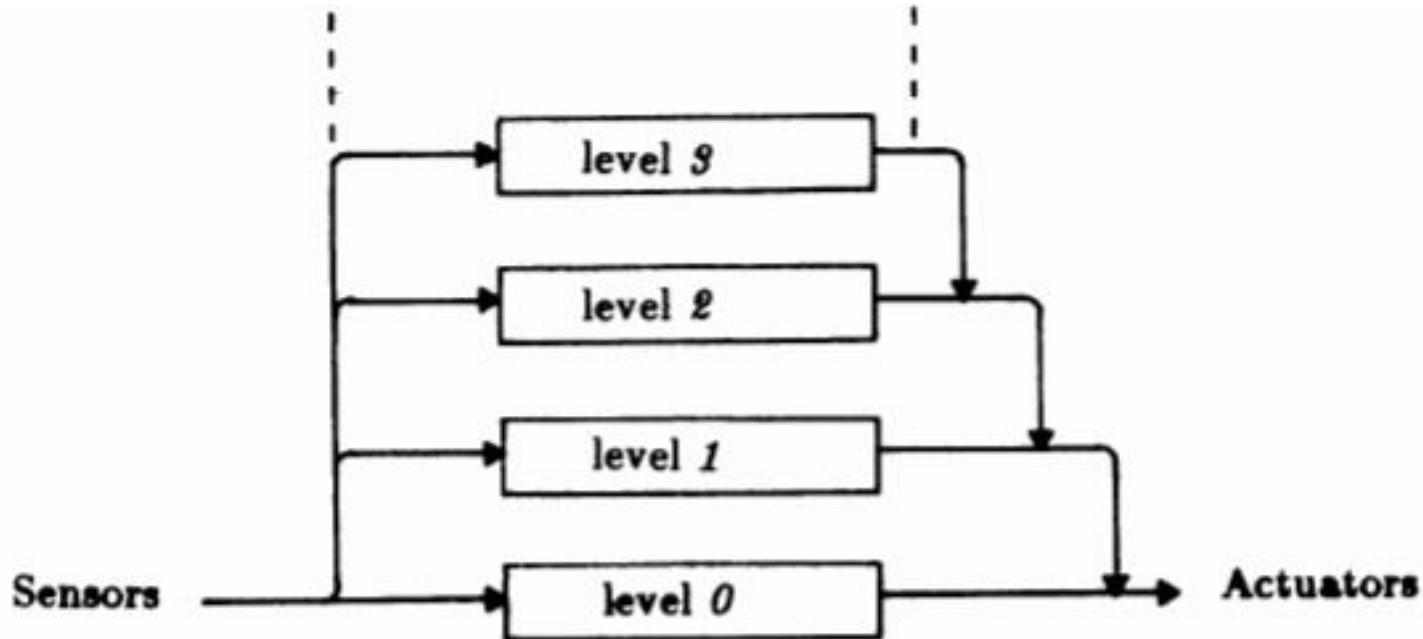
From Brooks, “A Robust Layered Control System for a Mobile Robot”, 1985

A Decomposition of a Mobile Robot Control System Based on Task Achieving Behaviors



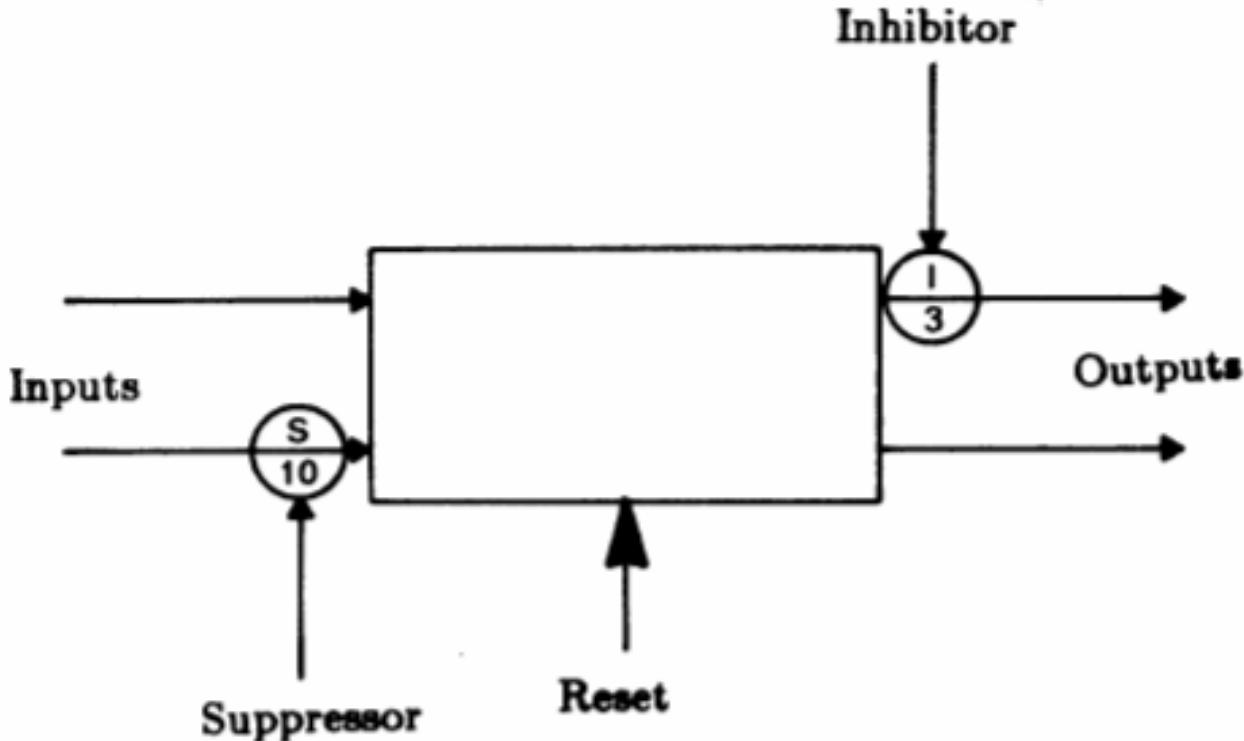
From Brooks, "A Robust Layered Control System for a Mobile Robot", 1985

Layered Control in the Subsumption Architecture



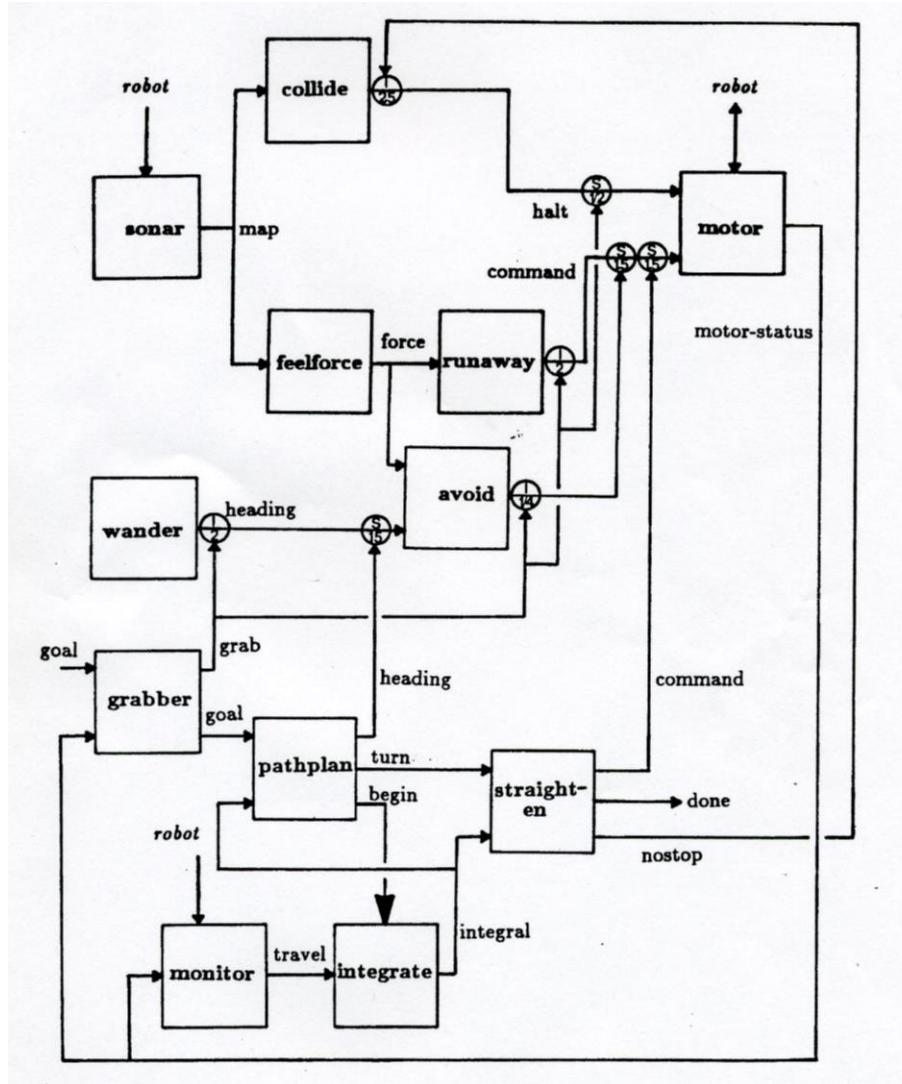
From Brooks, "A Robust Layered Control System for a Mobile Robot", 1985

Schematic of a Module



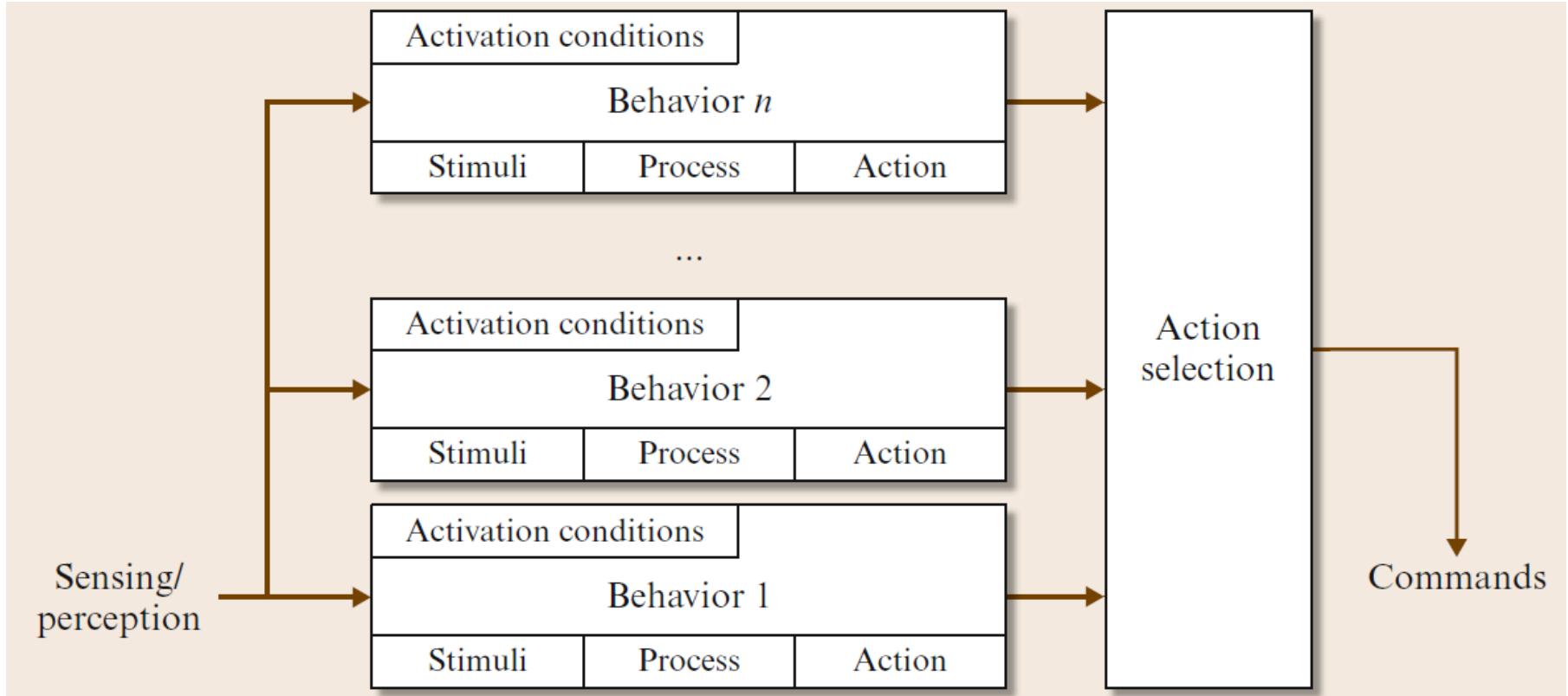
From Brooks, "A Robust Layered Control System for a Mobile Robot", 1985

Levels 0, 1, and 2 Control



From Brooks, "A Robust Layered Control System for a Mobile Robot", 1985

Behavior-Based Architectures



From Siciliano et al., "Springer Handbook of Robotics", Springer, 2008

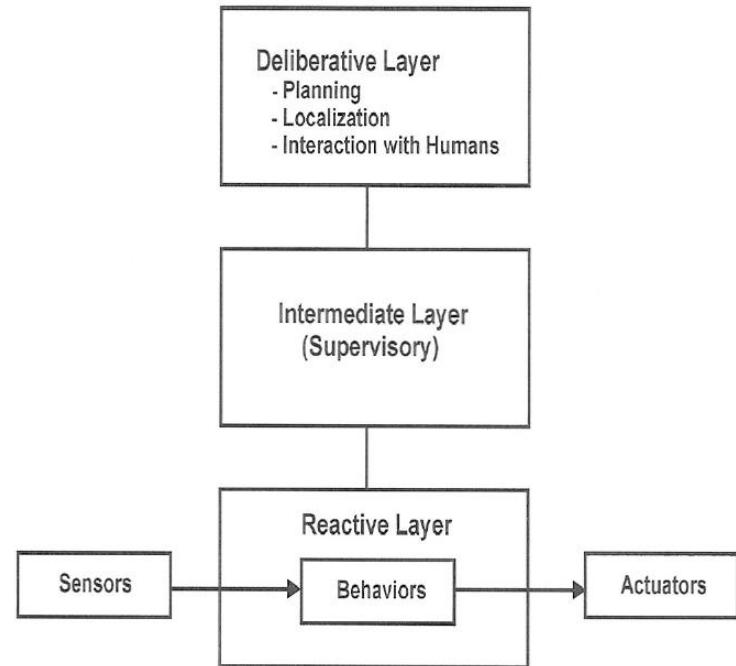
Hybrid Architectures

- In **Hybrid Control**, the goal is to **combine the best of both Reactive and Deliberative** control. In it, one part of the robot's "brain" plans, while another deals with immediate reaction, such as avoiding obstacles and staying on the road.
- The **challenge** of this approach is bringing the **two parts** of the brain **together**, and allowing them to **talk** to each other, and **resolve conflicts** between the two.
- This requires a "third" part of the robot brain, and as a result these systems are often called "three-layer systems"

Adapted from <http://www-robotics.usc.edu/~maja/robot-control.html>

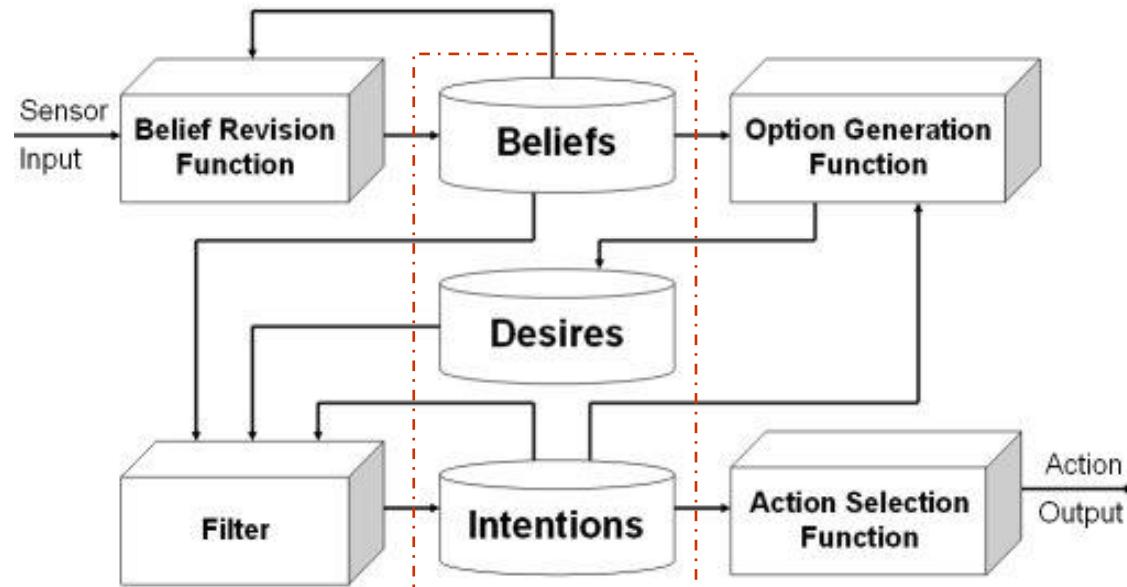
Hybrid Architectures

- Combine the responsiveness, robustness, and flexibility of purely reactive systems with more traditional symbolic/deliberative methods
- Reason: purely reactive systems lack the ability to take into account a priori knowledge (e.g. about the world) and to keep track of the history (memory)
- Typical three-layer (3T) hybrid architecture
 - Bottom layer is the reactive/behavior-based layer, in which sensors/actuators are closely coupled
 - Upper layer provides the deliberative component (e.g., planning, localization)
 - The intermediate between the two is sometimes called supervisory layer
- Examples of coupling between planning and reactive layers:
 - Planning to guide reaction: planning sets reactive system parameters.
 - Coupled: planning and reacting are concurrent activities, each guiding the other

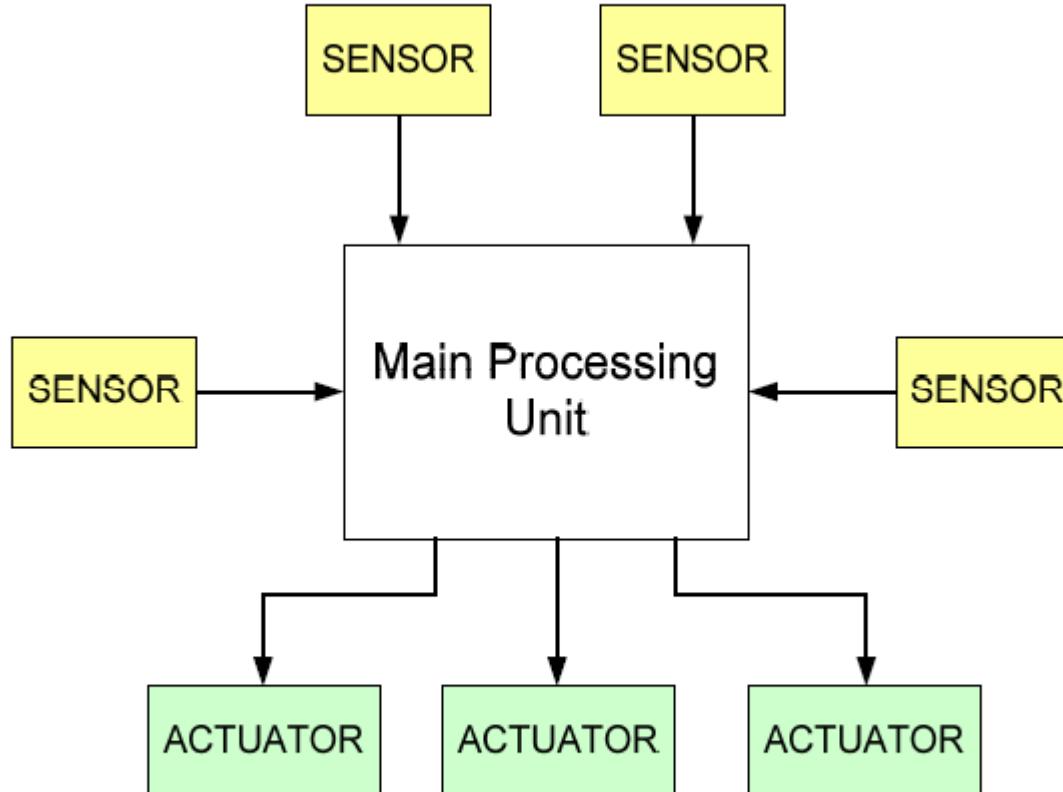


The BDI Model

- Three “mental attitudes”
 - (B)eliefs** are information the agent has about the world – information
 - (D)esires** are all the possible states of affairs that the agent might like to accomplish – motivation
 - (I)ntentions** are the states of affairs that the agent has decided to work towards – deliberation



Fully Centralized Computing Architecture

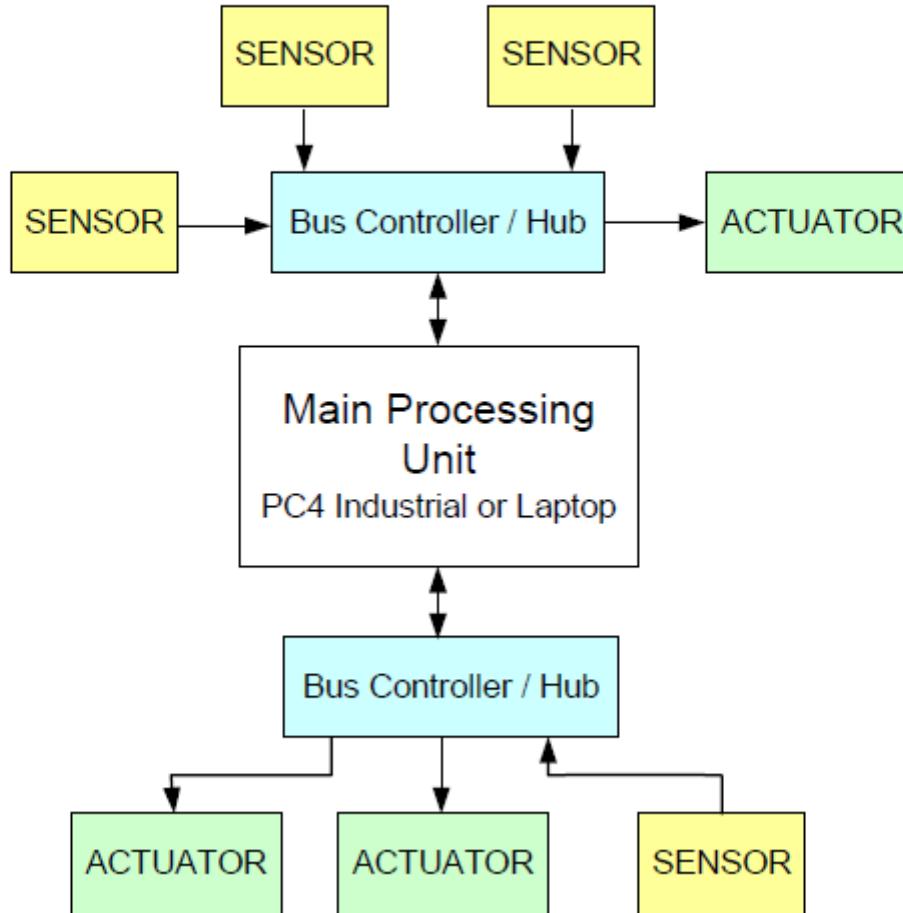


NOTES:

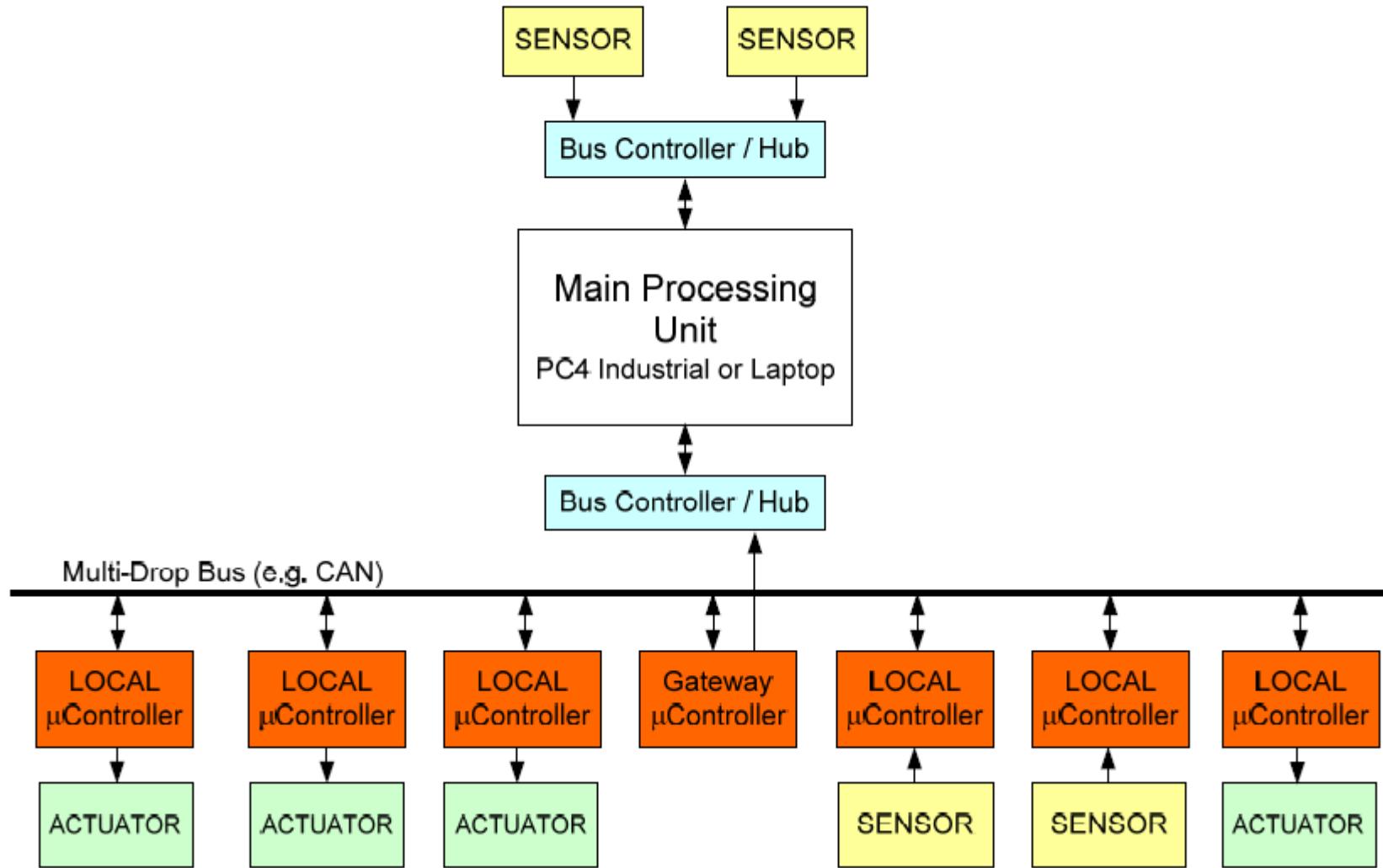
Sensors may include conditioning electronics and A/D converters

Actuators may include driving electronics and/OR D/A converters

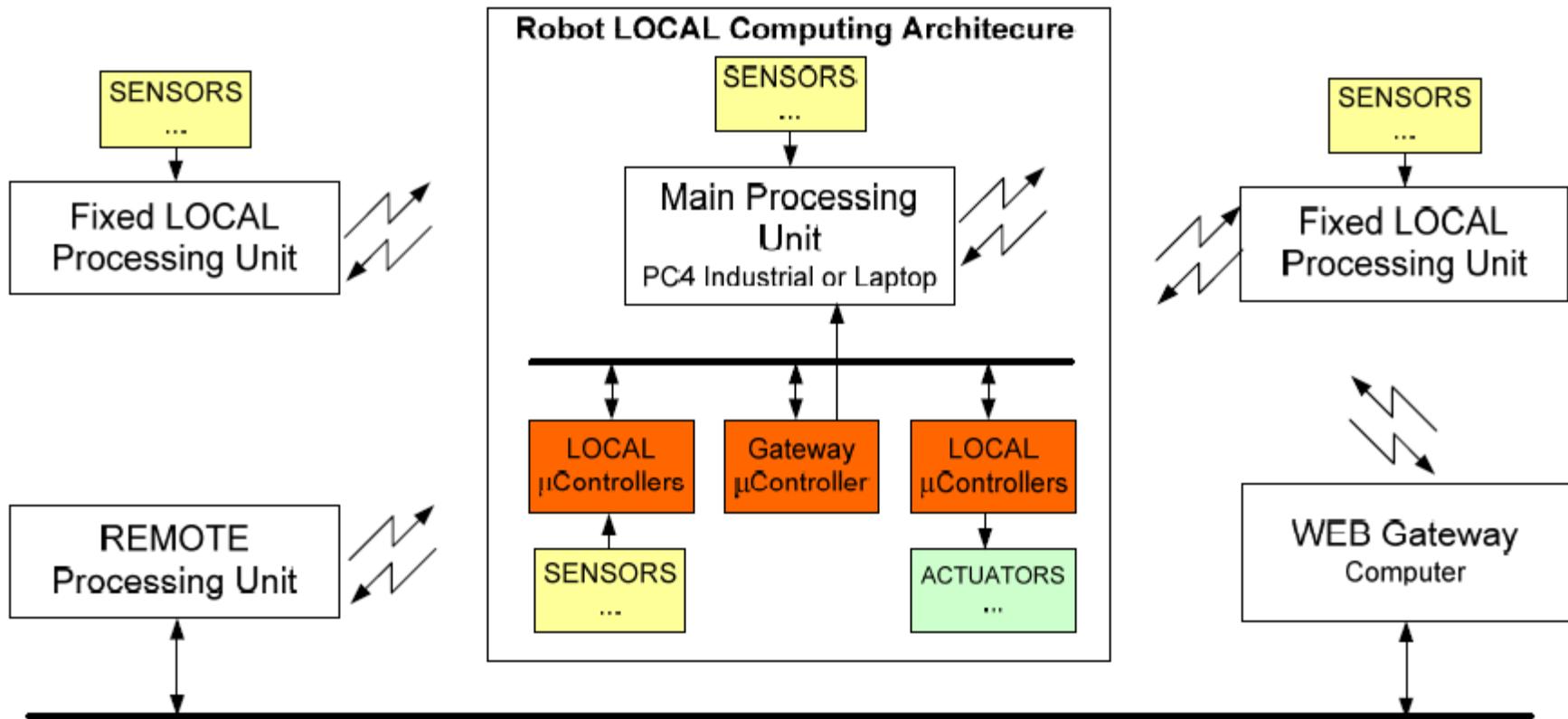
Star Network Computing Architecture



Hierarchical Distributed Computing Architecture



Fully Distributed Computing Architecture (FDCA)



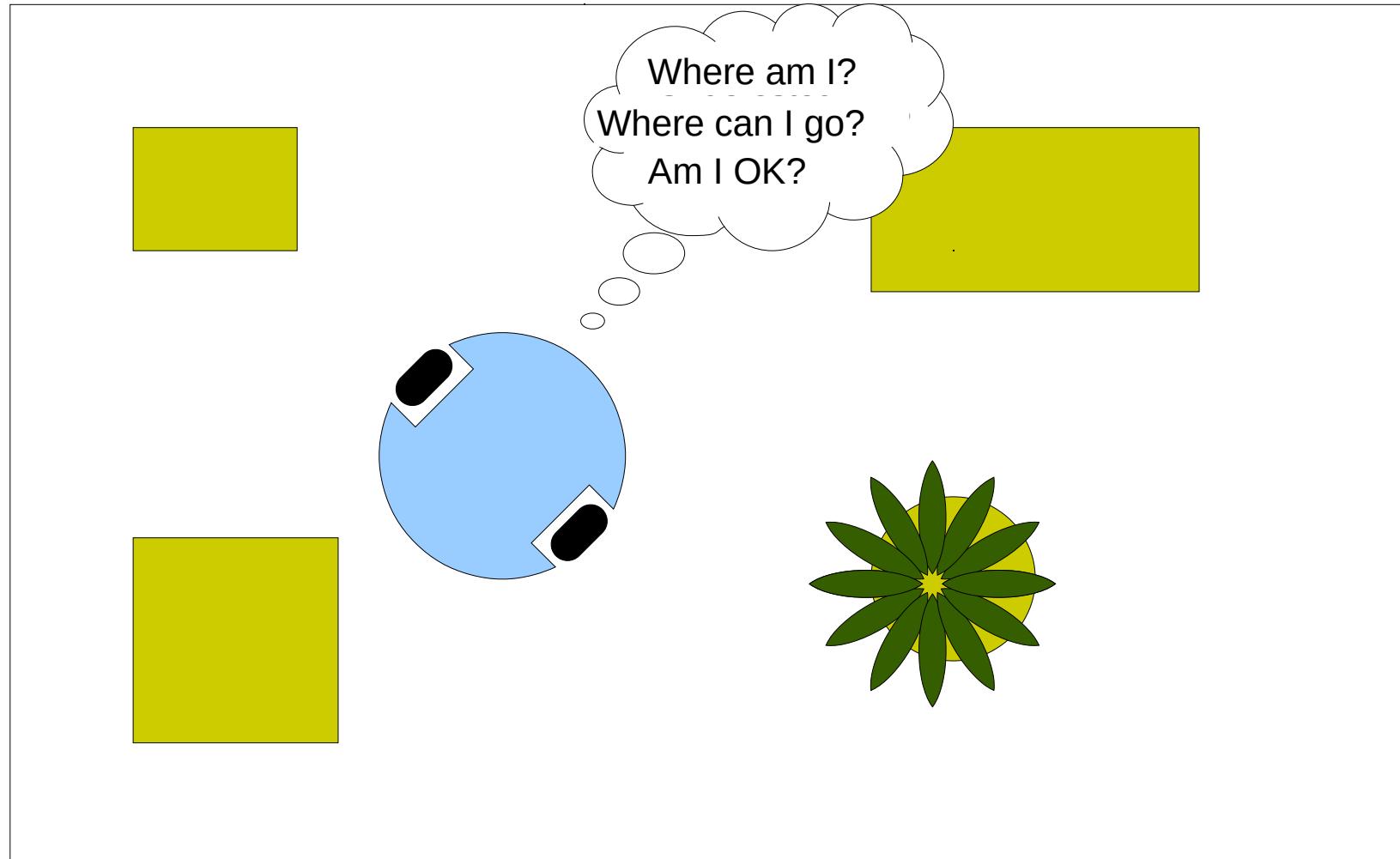
Robótica Móvel e Inteligente / Mobile and Intelligent Robotics
Mestrado Integrado em Engenharia de Computadores e Telemática

Academic year 2022/23

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

CC-BY-SA The IRIS team, 2023

existential problems in a robot's life



- Self perception (“How am I doing?”)
 - Posture
 - Batteries, ...
- Location (“Where am I?”)
 - Position
 - Orientation
- Environment perception (“Where can I go to?”)
 - obstacles
 - maps: constructing and location
 - targets (application level)

Challenges in mobile robotics

Navigate :: follow fixed path

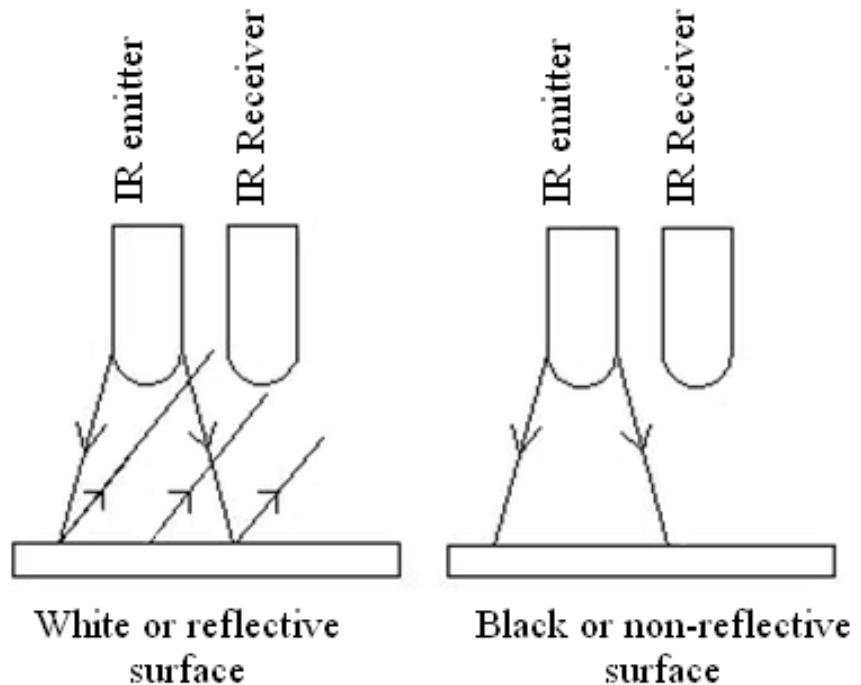
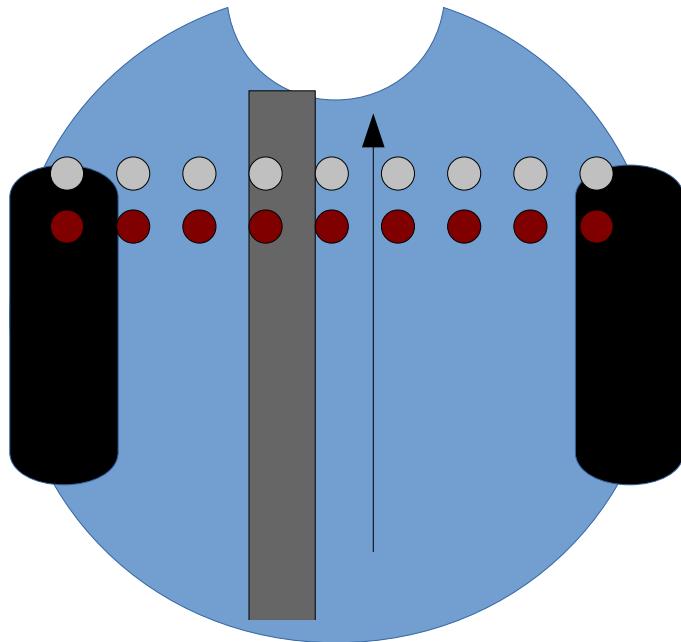
Line sensors: magnetic



<https://www.roboteq.com/all-products/magnetic-guide-sensors>

Line sensors :: optic

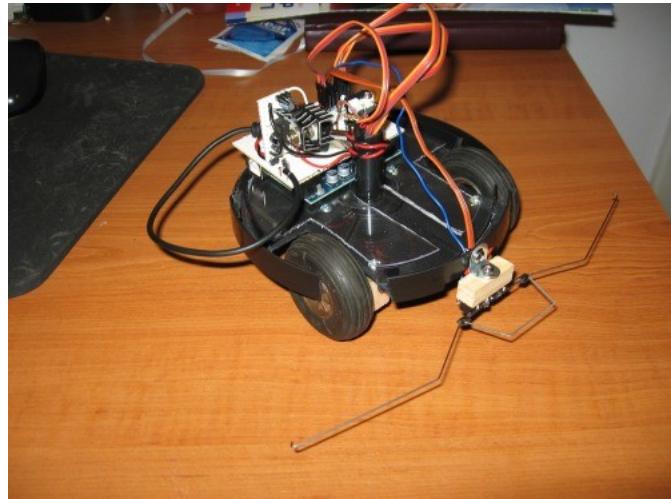
- Reflection on a object
- Detection depends on the object colour



Navigate :: react to surroundings

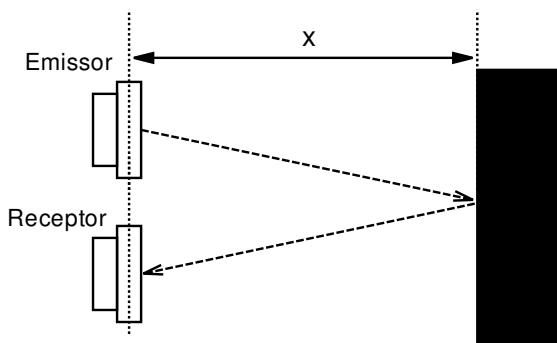
contact sensors

- **mechanically actuated switch**
 - whisker
 - bumper



ultra-sound

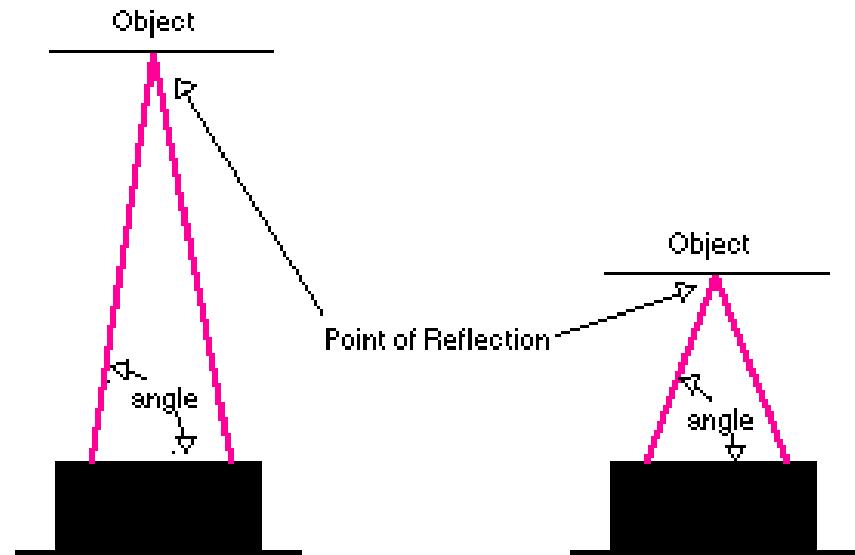
- Detection based on the reflection by an object



$$x = \frac{1}{2} \cdot c \cdot t_{\text{echo}}$$



- Sharp sensor
 - Distance information

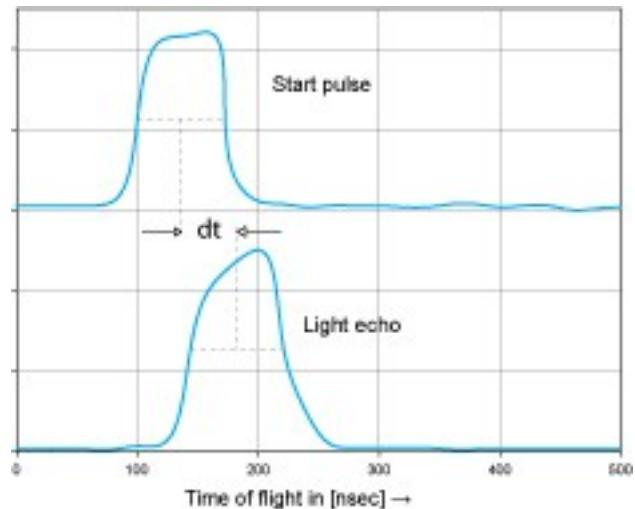
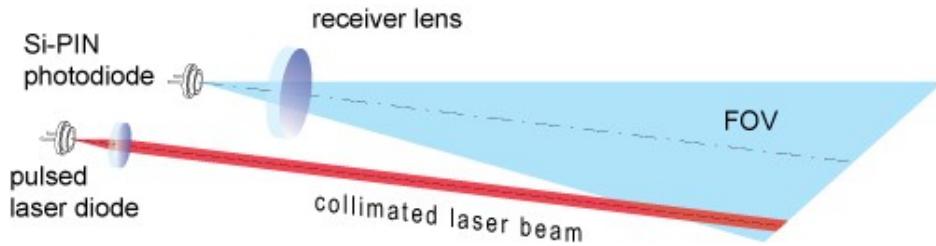


Robotic challenges

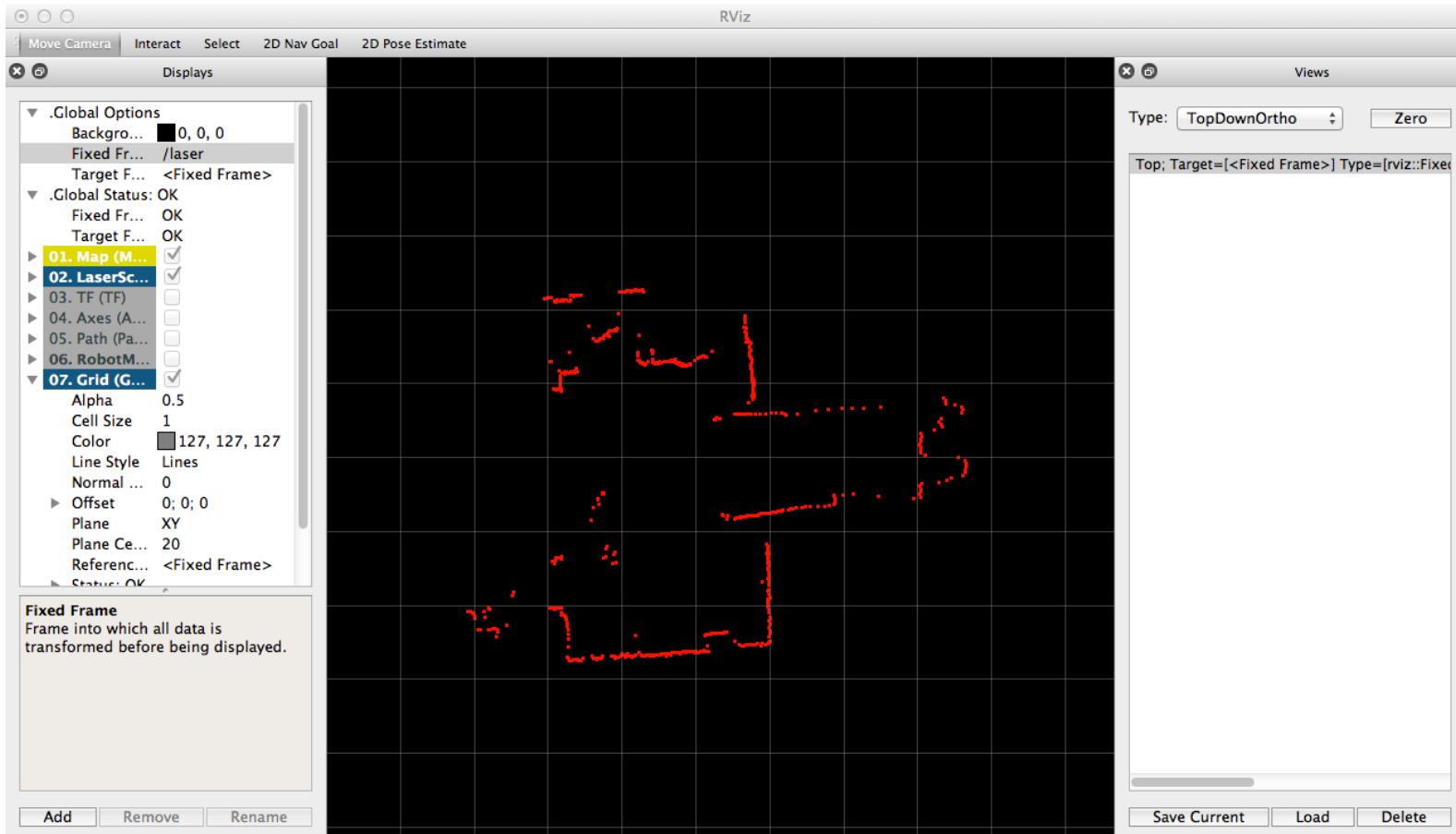
Navigate :: follow a path in a map

Laser range finder

- Laser scans the space ahead or around the robot
- Measuring obstacle distance
 - Limited to the beam working plan
 - Sometimes used with beam oscillation (\perp to the plan of beam)



Laser range finder



- Image of Sick's LRF software application

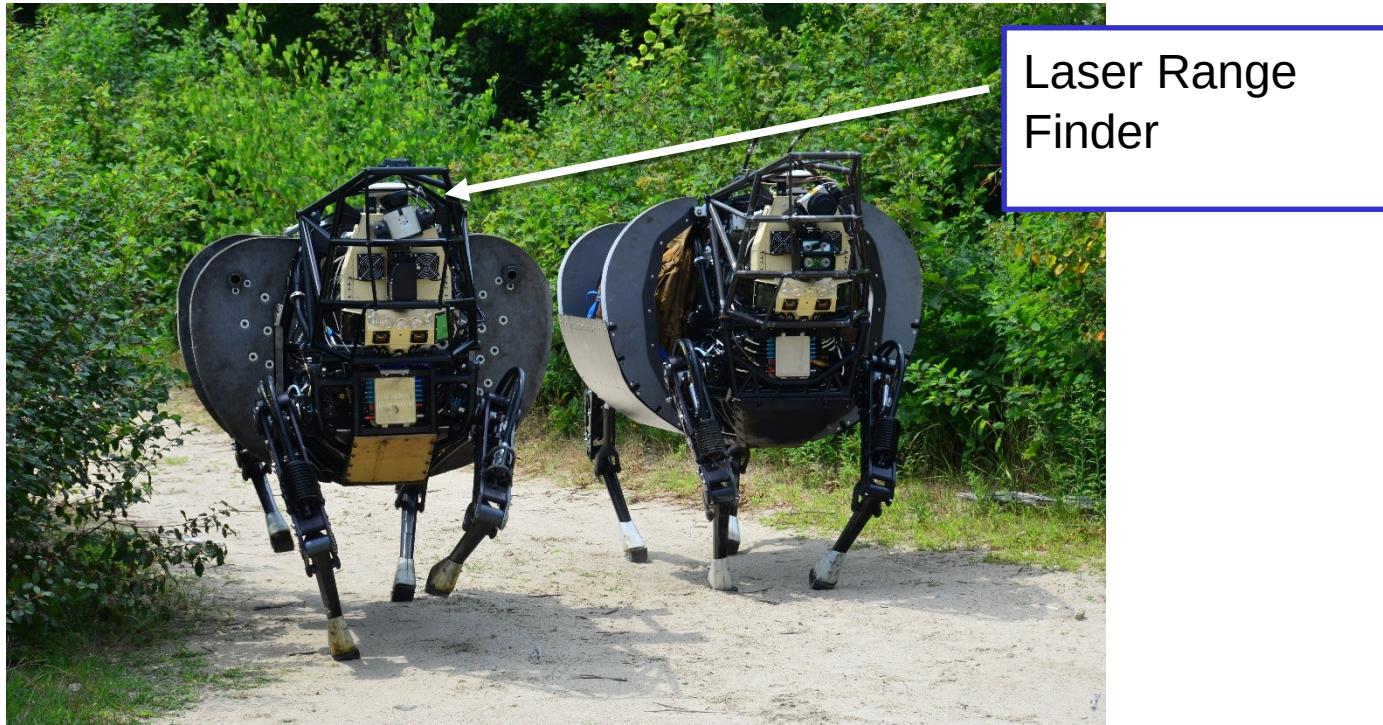


Mapa of IEETA's building level 0,
obtained by LRF scanning

LS3 – Legged Squad Support System

- The LS3 (Legged Squad Support System) is an example of a robot using LRF.
 - For example, in the YouTube video, at 1:17 you can clearly see the LFR oscillating to create a 3D perception of space ahead.

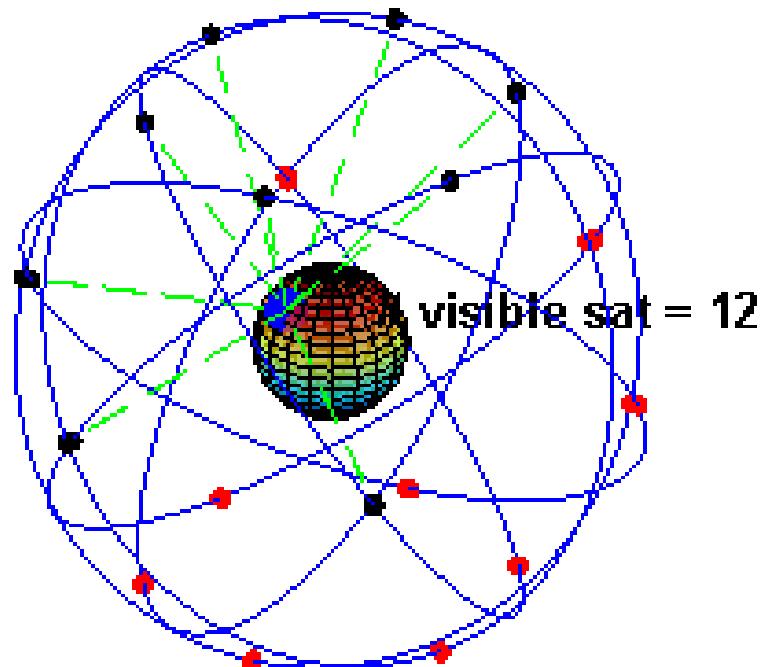
<http://youtu.be/R7ezXBEBE6U?t=1m17s>



position

GPS

- Absolute positioning (error in the order of m)
- Relative positioning (error in the order of cm for short time intervals)
- Requires line of sight to a minimum number of satellites → outdoor use
- Start time



error sources in GPS

Ionospheric effects	± 5 meters
Shifts in the satellite orbits	± 2.5 meter
Clock errors of the satellites' clocks	± 2 meter
Multipath effect	± 1 meter
Tropospheric effects	± 0.5 meter
Calculation and rounding errors	± 1 meter

<http://www.kowoma.de/en/gps/errors.htm>

mobile phone location

- Based on mobile phone location services
 - e.g.: multilateration with cell tower signals

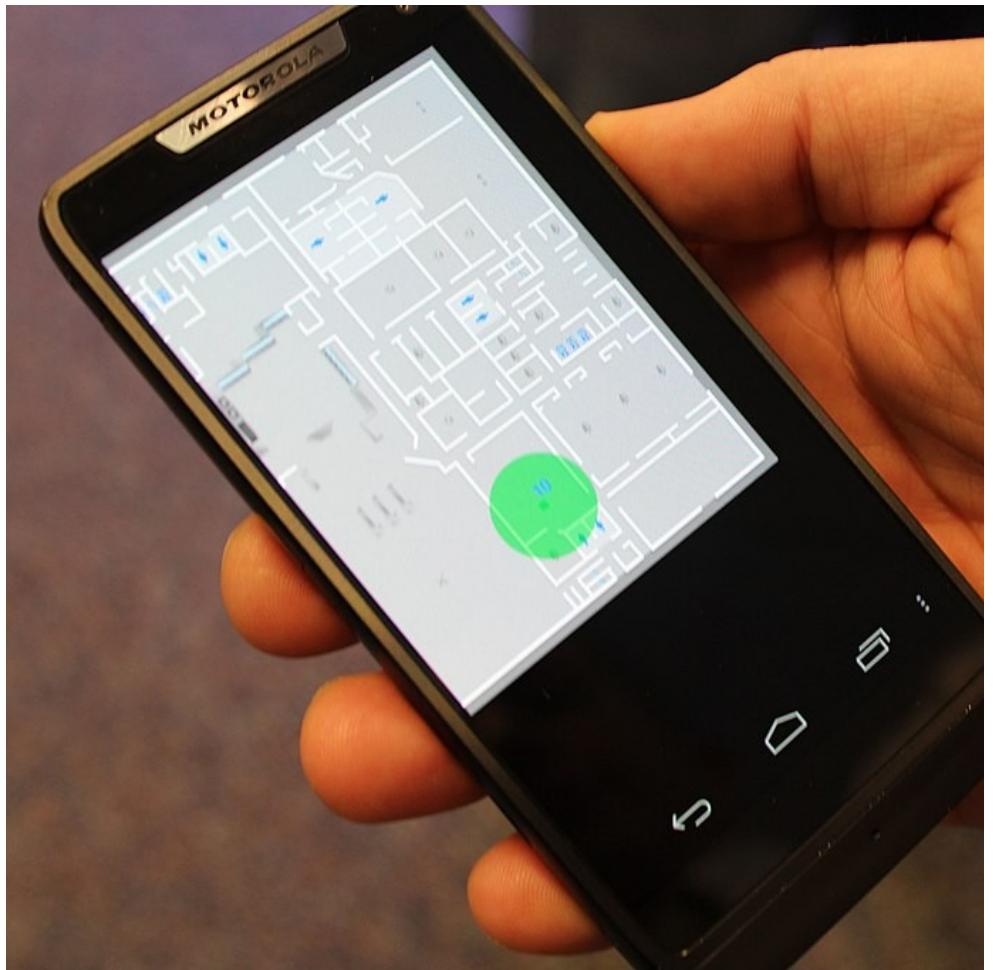


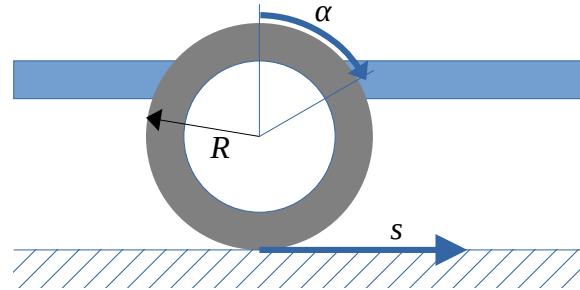
Photo by Intel Free Press - Indoor location services on mobile phone, CC BY-SA 2.0,
<https://commons.wikimedia.org/w/index.php?curid=71130241>

Navigate :: sense myself

Position

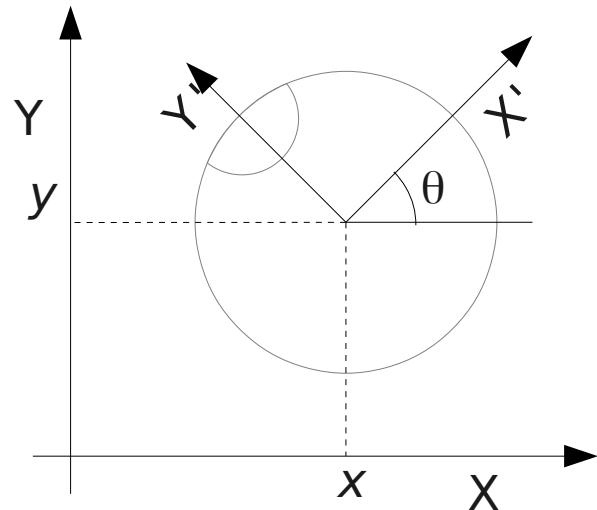
- Odometry

- def.: use of data from motion sensors to estimate change in position over time
- relative position
- Sources of error:
 - limited resolution (encoder)
 - model inaccuracies:
 - systematic → accumulates (error in slope)
 - error in wheel diameter
 - slippage
 - random, can reach significant values
 - Errors accumulate → unbounded !!



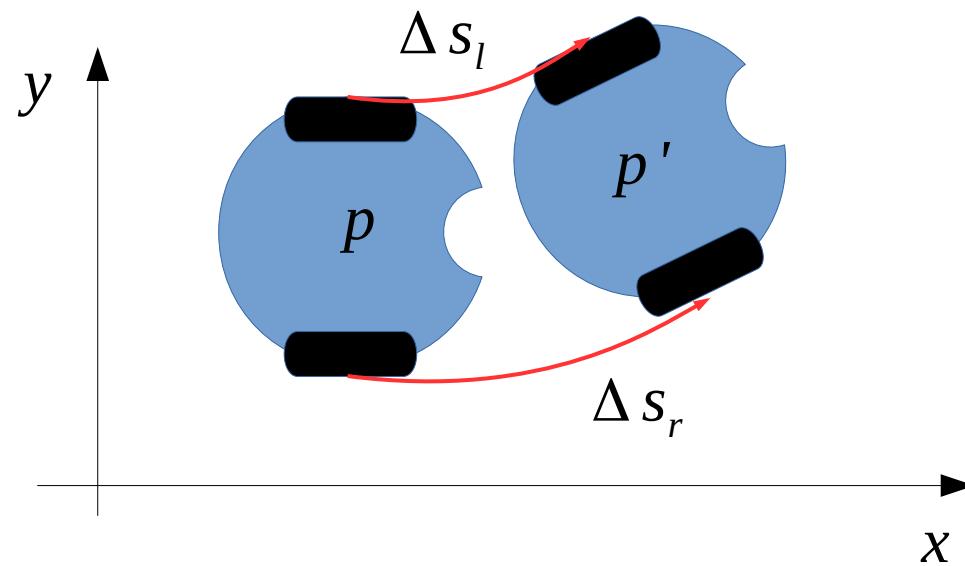
$$s = \alpha \times R$$

s



Robot location in the plane
Pose

$$p = \begin{bmatrix} x \\ y \\ \Theta \end{bmatrix}$$

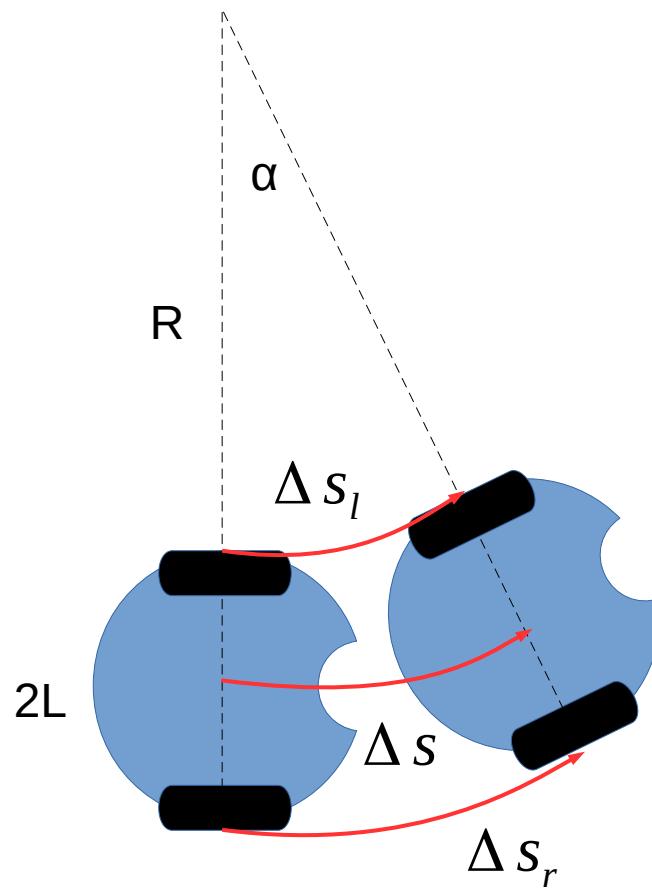


$$\Delta s_l = R \alpha$$

$$\Delta s_r = (R + 2L) \alpha$$

$$\Delta s = (R + L) \alpha$$

$$\Delta s = \frac{\Delta s_l + \Delta s_r}{2}$$

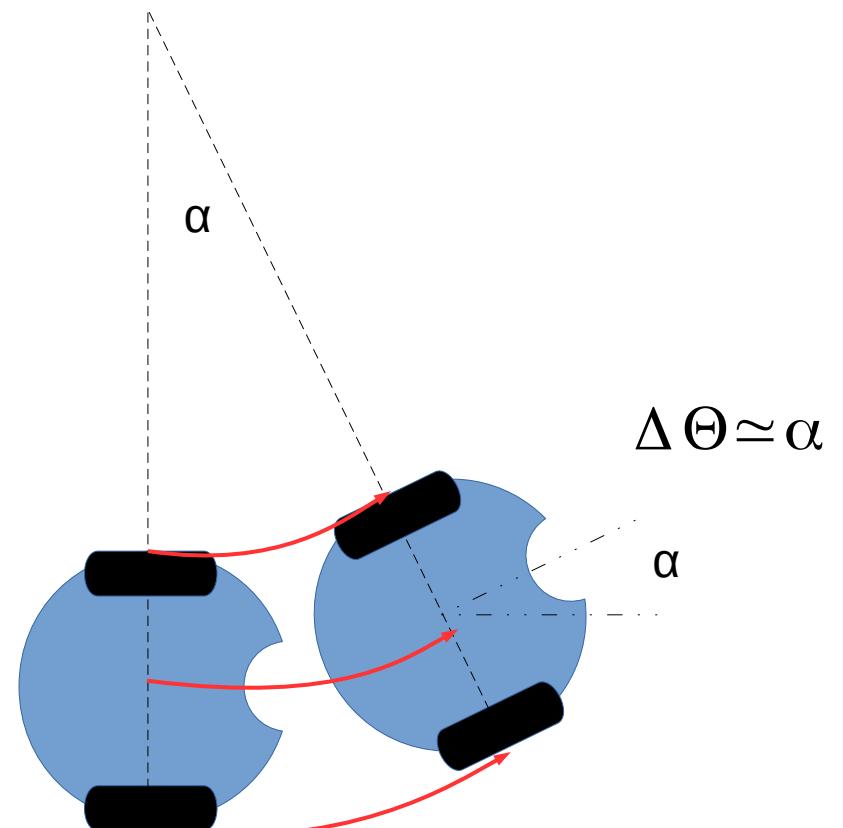


$$\Delta s_l = R \alpha$$

$$\Delta s_r = (R + 2L) \alpha$$

$$\frac{\Delta s_l}{R} = \frac{\Delta s_r}{R + 2L}$$

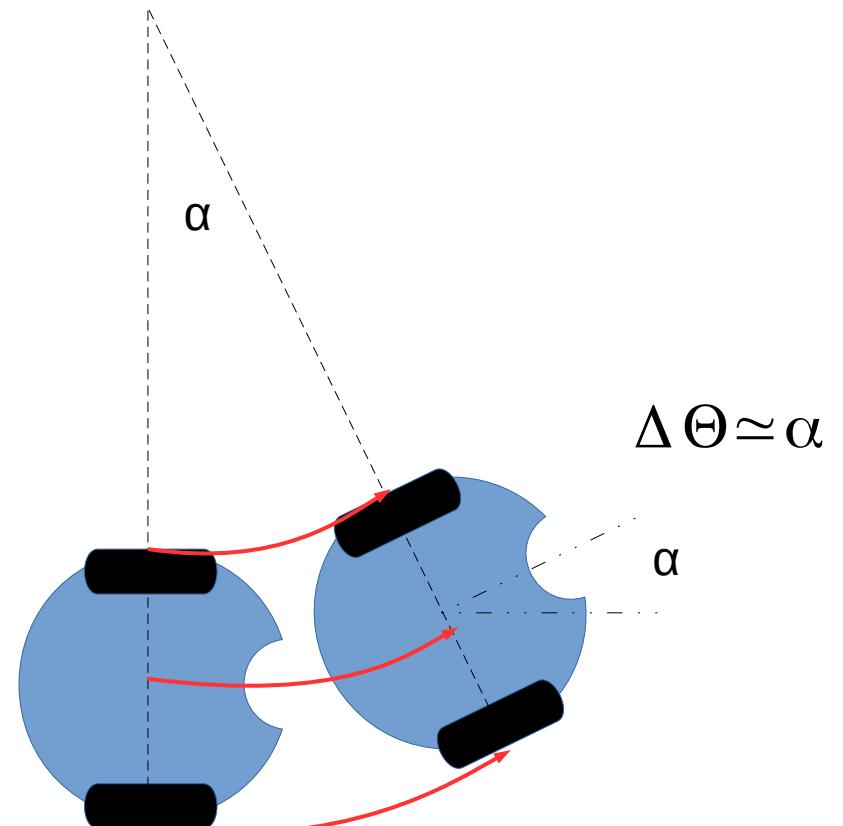
$$R = \frac{2L \Delta s_l}{\Delta s_r - \Delta s_l}$$



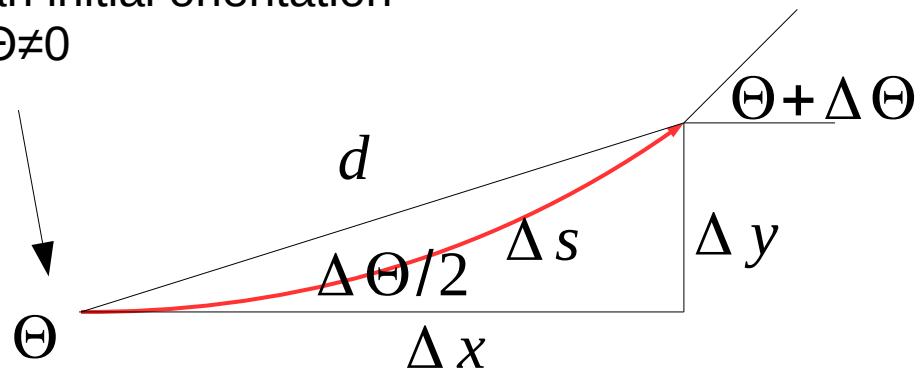
$$\Delta S_l = R \alpha$$

$$R = \frac{2L \Delta S_l}{\Delta S_r - \Delta S_l}$$

$$\begin{aligned} \alpha &= \Delta S_l / R \\ &= \frac{\Delta S_r - \Delta S_l}{2L} \\ \Delta \Theta &\simeq \frac{\Delta S_r - \Delta S_l}{2L} \end{aligned}$$



Robot may have
an initial orientation
 $\Theta \neq 0$



Projection of d on the x and y axis

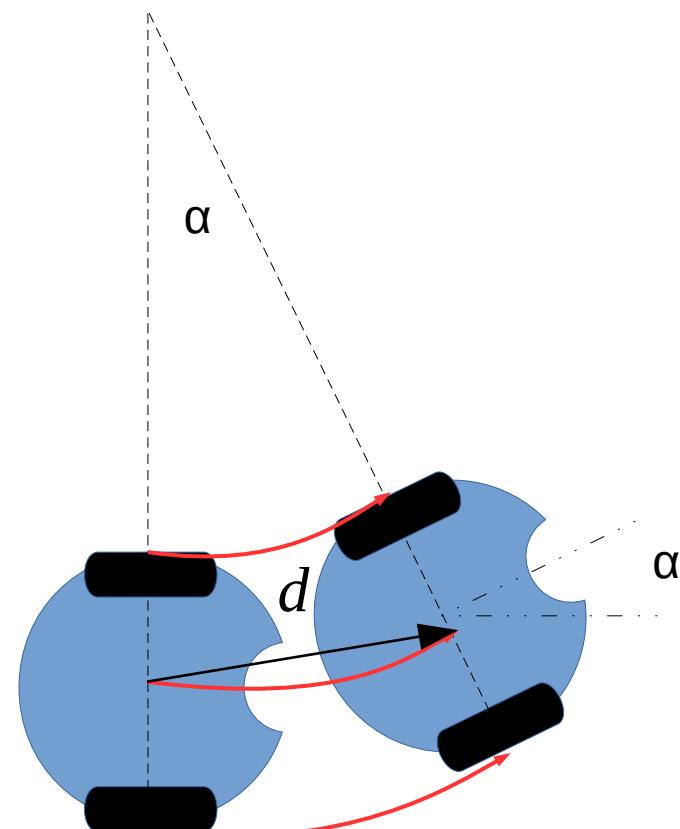
$$\Delta x = \Delta d \cos(\Theta + \Delta\Theta/2)$$

$$\Delta y = \Delta d \sin(\Theta + \Delta\Theta/2)$$

If Δs very small:

$$\Delta x = \Delta s \cos(\Theta + \Delta\Theta/2)$$

$$\Delta y = \Delta s \sin(\Theta + \Delta\Theta/2)$$



Initial pose:

$$p = \begin{bmatrix} x \\ y \\ \Theta \end{bmatrix}$$

$$\Delta x = \Delta s \cos(\Theta + \Delta\Theta/2)$$

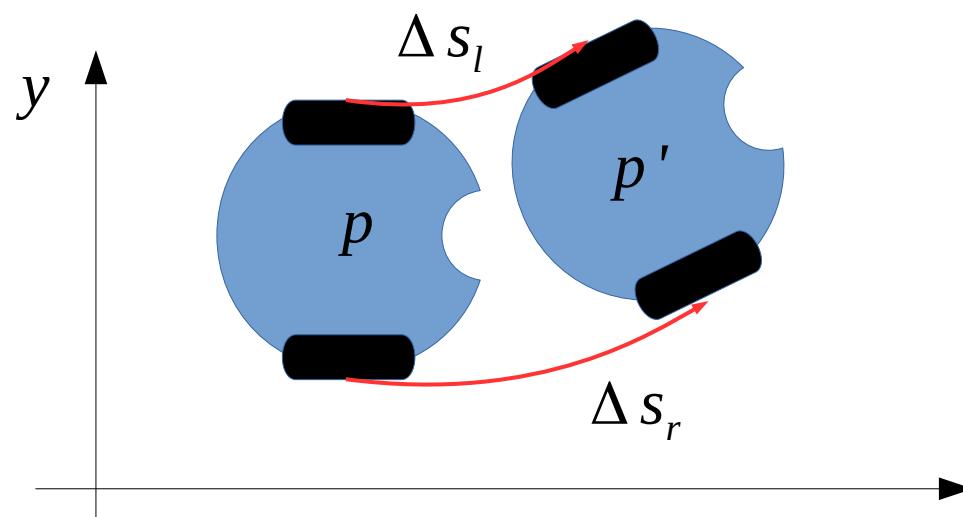
$$\Delta y = \Delta s \sin(\Theta + \Delta\Theta/2)$$

$$\Delta\Theta \approx \frac{\Delta s_r - \Delta s_l}{2L}$$

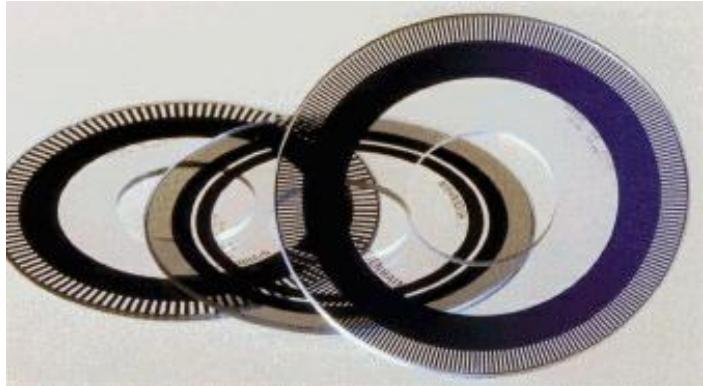
$$\Delta s = \frac{\Delta s_l + \Delta s_r}{2}$$

Final pose (after displacement)

$$p' = \begin{bmatrix} x \\ y \\ \Theta \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_l + \Delta s_r}{2} \cos(\Theta + \Delta\Theta/2) \\ \frac{\Delta s_l + \Delta s_r}{2} \sin(\Theta + \Delta\Theta/2) \\ \frac{\Delta s_r - \Delta s_l}{2L} \end{bmatrix}$$



optical encoder

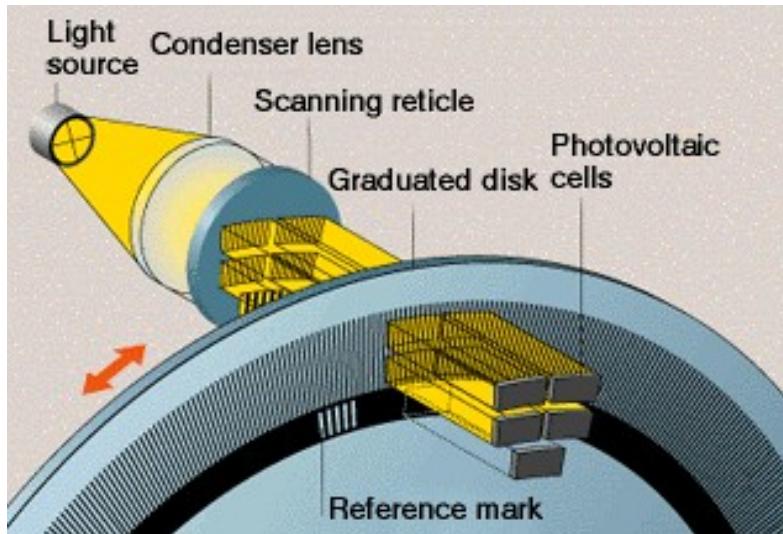
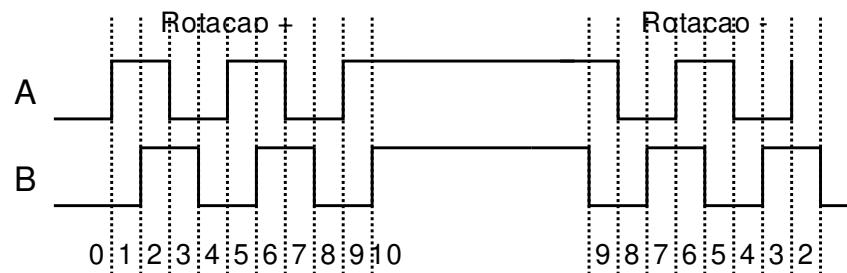


- Pulses generated by the interference of two patterns of stripes
- encoder characterized by p.p.r. (pulses per revolution)
- # of pulses proportional to displacement

$$\alpha = \frac{\text{count}}{\text{p.p.r.}} \cdot 2\pi$$

optical encoder

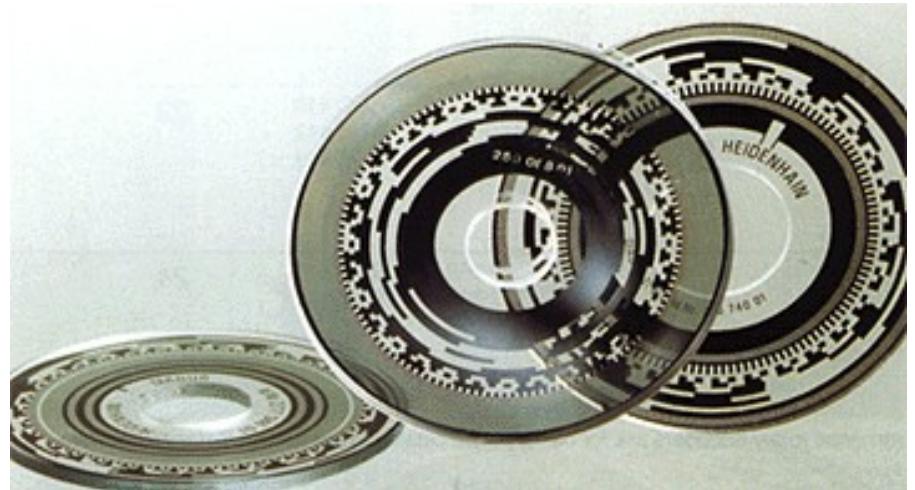
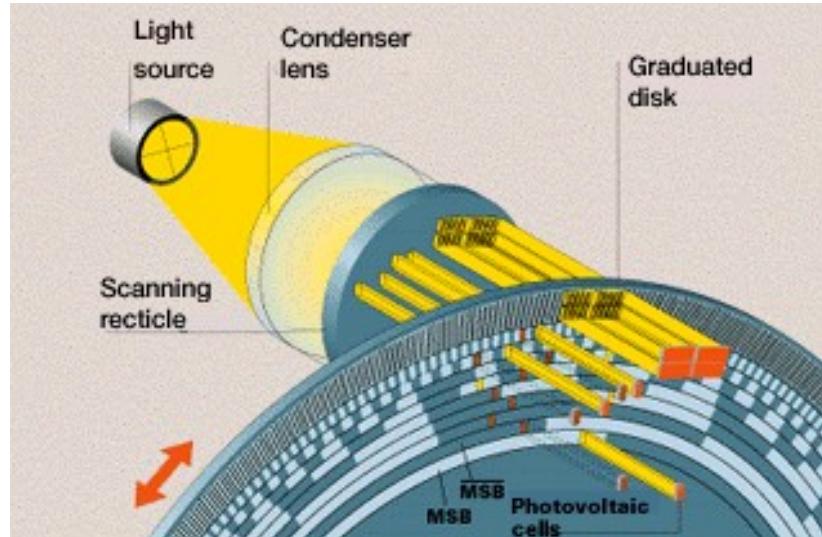
- Interference generates a varying signal with displacement
- This signal is converted to digital



- **Quadrature allows:**
 - to detect the direction of movement
 - multiply encoder resolution by 4
 - 1 impulse = 4 counts

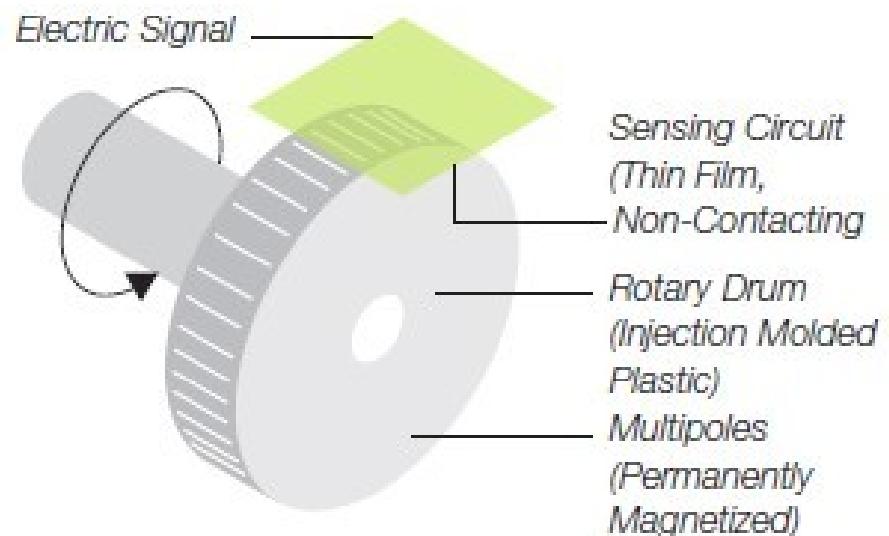
absolute encoder

- optical disk with Gray code
- output is shaft position (angle) in binary code



magnetic encoder

- principle similar to optical incremental encoder
- small permanent magnets in the shaft drum
- sensing circuit detects passing magnets and measures rotation
- Pro: unaffected by dust, moisture, and extreme temperatures, and shock.



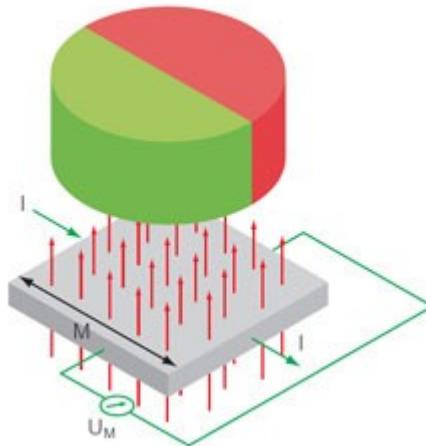
- bicycle computers work in a similar fashion



<https://www.bicycle-guides.com/cycling-advice/best-bike-computers-buyers-guide/>

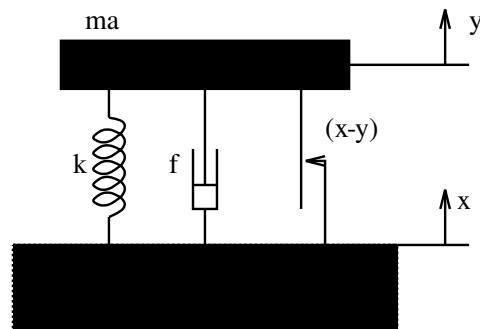
hall effect encoder

- Based on **Hall effect**
 - production of a voltage difference (the Hall voltage) across an electrical conductor, transverse to an electric current in the conductor and to an applied magnetic field perpendicular to the current
- A permanent magnet is attached to the shaft
- Orientation of magnetic field is detected by an array sensor



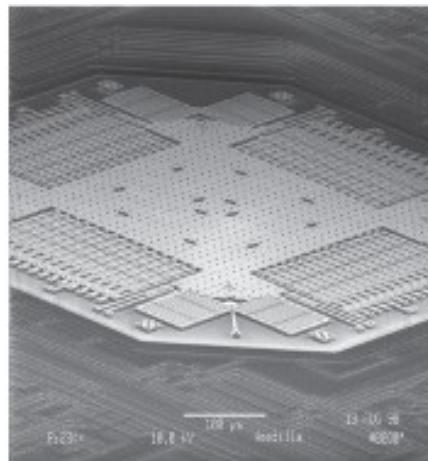
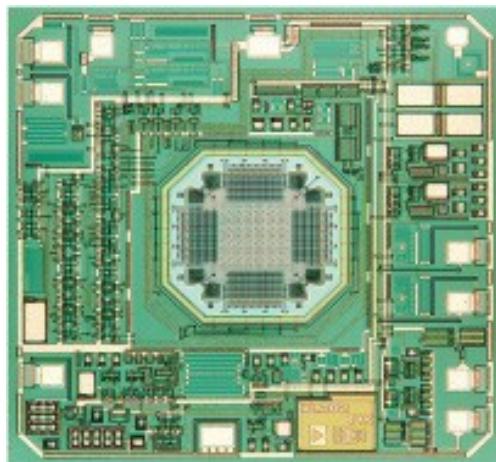
https://www.dynapar.com/Technology/Encoder_Basics/Magnetic_Encoder/

accelerometers



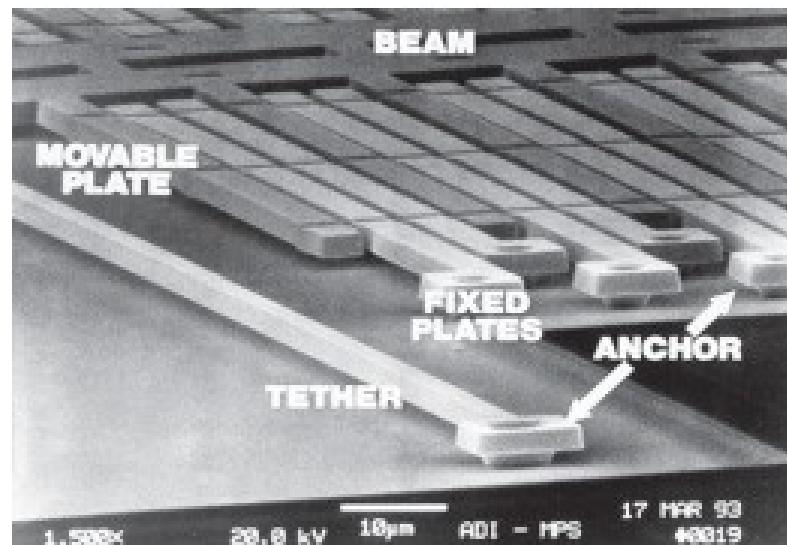
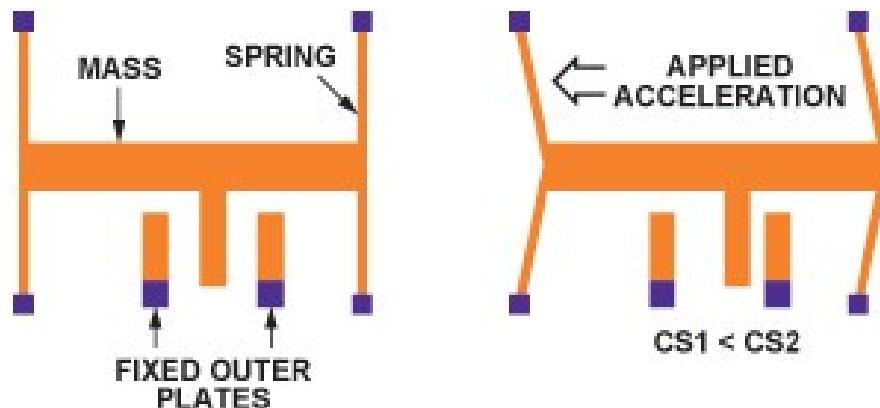
$$k(x-y) + f(\dot{x} - \dot{y}) - m_a \ddot{y} = 0$$

- inertial mass principle
- Ex.: ADXL... from Analog Devices
- MEMS: Micro-electromechanical Systems



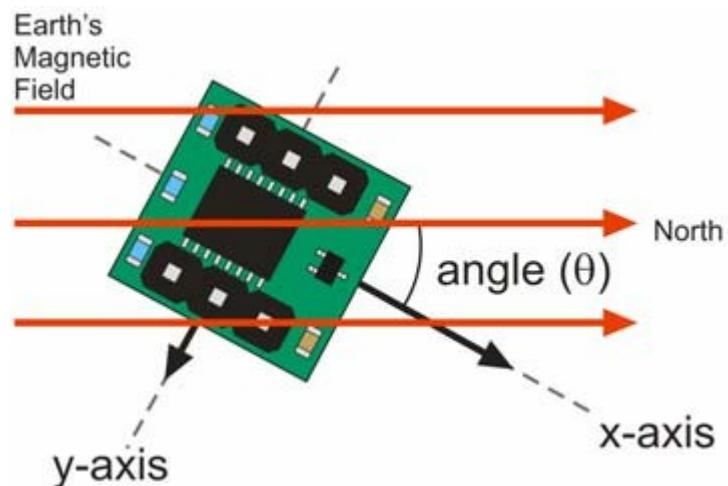
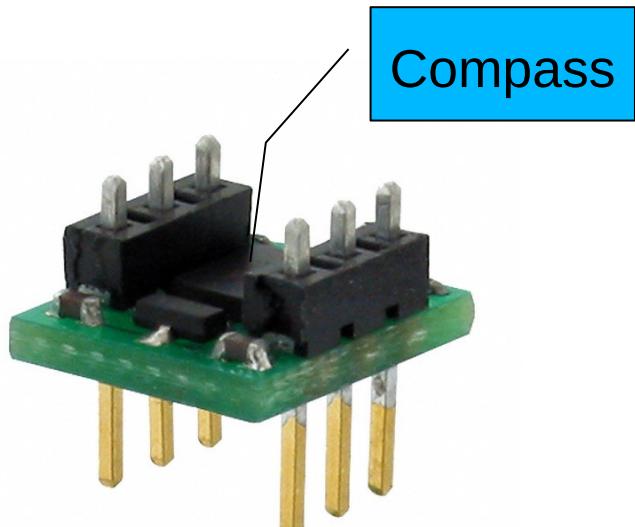
accelerometers

- Detection by changes on:
 - capacity
 - resistance



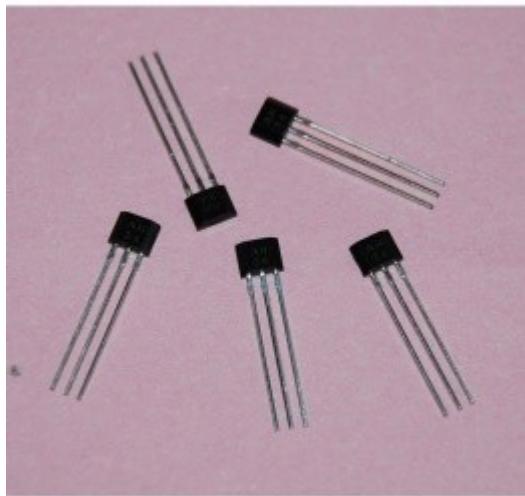
compass

- Ex.: HM55B (Hitachi)
 - Magnetic field detection in 2 axes, x e y
 - Trigonometry is used to compute angle with magnetic N



working principle

- **Hall effect**
 - voltage as a function of magnetic field



Gyroscopes

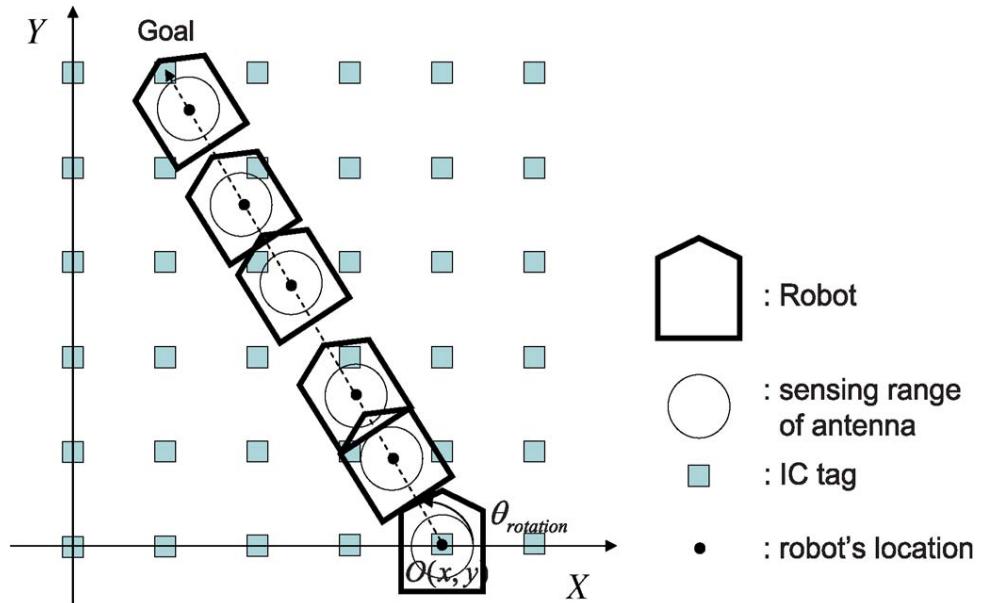
- Based on inertia principle
 - rotating disk
- Electronic devices based on mechanical oscillation
 - Foucault pendulum



Navigate :: use external references

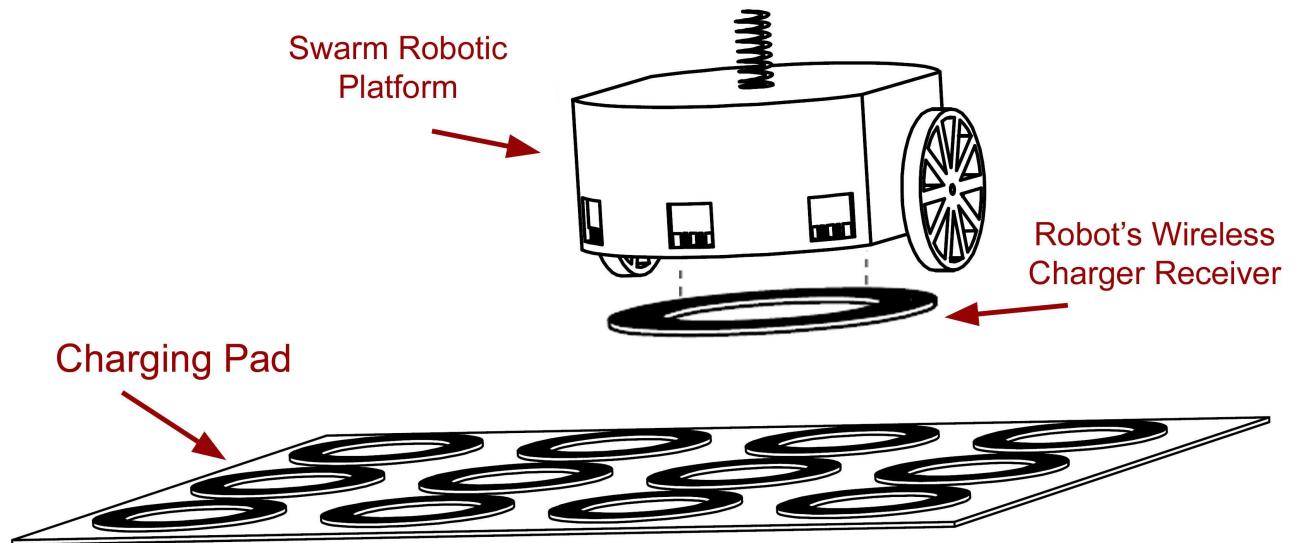
RFID location

- set of RFID tags installed in the floor
 - identifying the tag allows knowledge of position.



[1] Sunhong Park e Shuji Hashimoto, «Autonomous Mobile Robot Navigation Using Passive RFID in Indoor Environment», IEEE Transactions on Industrial Electronics, vol. 56, n. 7, Jul 2009.

Charging and location



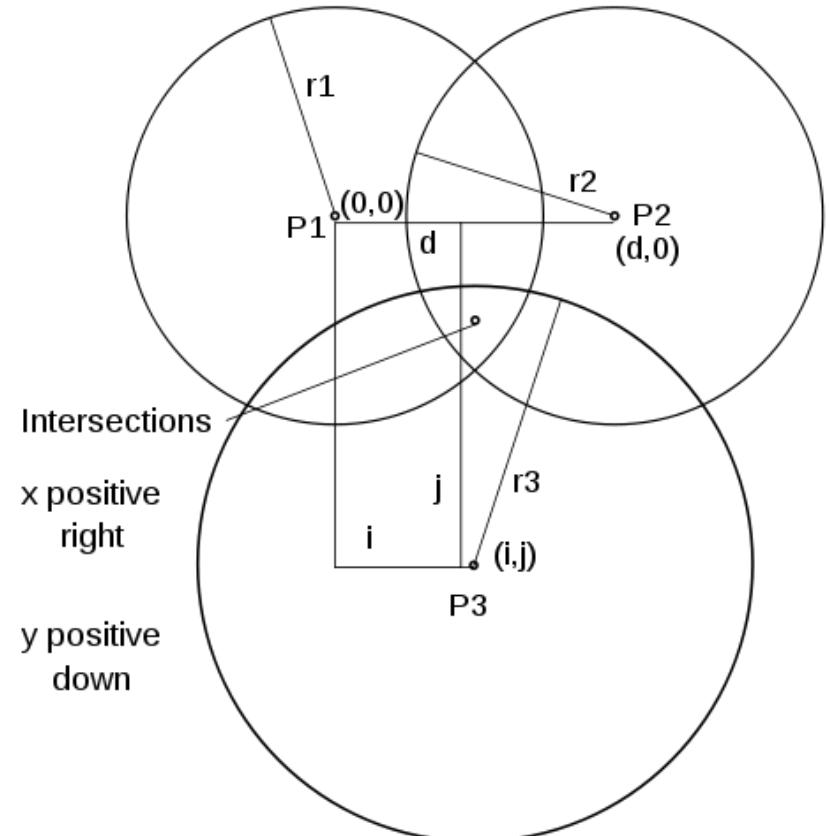
Coils in the floor can work for both:

- Charging (Dynamic Wireless Charging)
- Location

Li, Y., Zhong, L., & Lin, F. (2021). Predicting-Scheduling-Tracking: Charging Nodes With Non-Deterministic Mobility. *Ieee Access*, 9, 2213–2228. <https://doi.org/10.1109/ACCESS.2020.3046857>

trilateration

- computing position by measuring distance to 3 reference points
- TOF: ultra-sounds
- RSSI: radio signal
 - D. Hahnel, W. Burgard, D. Fox, K. Fishkin, e M. Philipose, «Mapping and localization with RFID technology», 2004, pp. 1015-1020 Vol.1.



$$\begin{aligned}
 d_1^2 &= (x - x_1)^2 + (y - y_1)^2 \\
 &= x^2 - 2x x_1 + x_1^2 + y^2 - 2y y_1 + y_1^2 \\
 d_2^2 &= x^2 - 2x x_2 + x_2^2 + y^2 - 2y y_2 + y_2^2
 \end{aligned}$$

...

$$d_n^2 = x^2 - 2x x_n + x_n^2 + y^2 - 2y y_n + y_n^2$$

Subtracting the first equation from equations 2 to n:

$$d_2^2 - d_1^2 = 2x(x_1 - x_2) + x_2^2 - x_1^2 + 2y(y_1 - y_2) + y_2^2 - y_1^2$$

$$d_3^2 - d_1^2 = 2x(x_1 - x_3) + x_3^2 - x_1^2 + 2y(y_1 - y_3) + y_3^2 - y_1^2$$

...

$$d_n^2 - d_1^2 = 2x(x_1 - x_n) + x_n^2 - x_1^2 + 2y(y_1 - y_n) + y_n^2 - y_1^2$$

$$2(x_1 - x_2)x + 2(y_1 - y_2)y = d_2^2 - d_1^2 + x_1^2 - x_2^2 + y_1^2 - y_2^2$$

$$2(x_1 - x_3)x + 2(y_1 - y_3)y = d_3^2 - d_1^2 + x_1^2 - x_3^2 + y_1^2 - y_3^2$$

...

$$2(x_1 - x_n)x + 2(y_1 - y_n)y = d_n^2 - d_1^2 + x_1^2 - x_n^2 + y_1^2 - y_n^2$$

$$\begin{bmatrix} 2(x_1 - x_2) & 2(y_1 - y_2) \\ 2(x_1 - x_3) & 2(y_1 - y_3) \\ \dots & \dots \\ 2(x_1 - x_n) & 2(y_1 - y_n) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} d_2^2 - d_1^2 + x_1^2 - x_2^2 + y_1^2 - y_2^2 \\ d_3^2 - d_1^2 + x_1^2 - x_3^2 + y_1^2 - y_3^2 \\ \dots \\ d_n^2 - d_1^2 + x_1^2 - x_n^2 + y_1^2 - y_n^2 \end{bmatrix}$$

$$A X = B$$

$$A = \begin{bmatrix} 2(x_1 - x_2) & 2(y_1 - y_2) \\ 2(x_1 - x_3) & 2(y_1 - y_3) \\ \dots & \dots \\ 2(x_1 - x_n) & 2(y_1 - y_n) \end{bmatrix} \quad X = \begin{bmatrix} x \\ y \end{bmatrix}$$

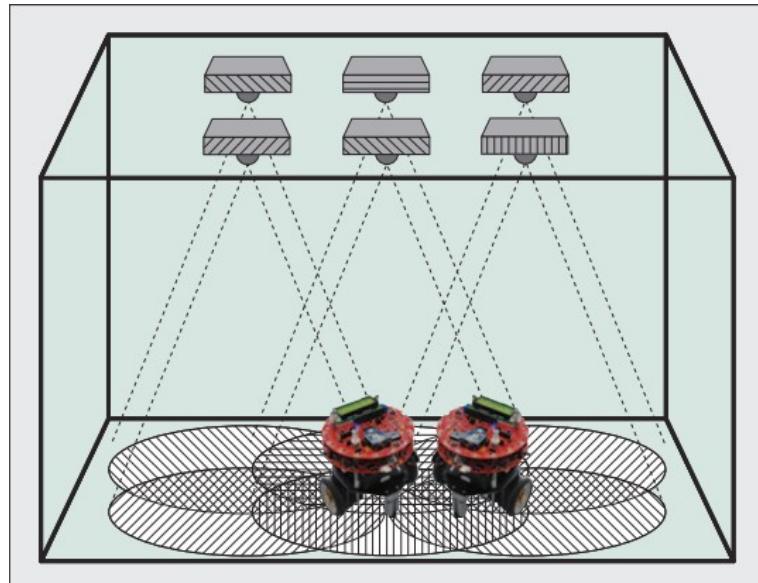
A least-square approximation can be found by using the Moore-Penrose pseudo-inverse:

$$X = (A^T \cdot A)^{-1} A^T B$$

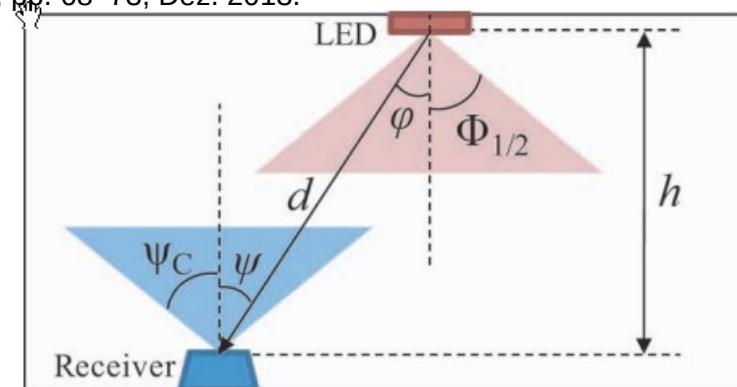
$$B = \begin{bmatrix} d_2^2 - d_1^2 + x_1^2 - x_2^2 + y_1^2 - y_2^2 \\ d_3^2 - d_1^2 + x_1^2 - x_3^2 + y_1^2 - y_3^2 \\ \dots \\ d_n^2 - d_1^2 + x_1^2 - x_n^2 + y_1^2 - y_n^2 \end{bmatrix}$$

visible light positioning (VLP)

- Conditions
 - widespread use of LED for illumination
 - LEDs allow light modulation → VLC (visible light communication)
- VLC + AOA (Angle of Arrival) = VLP
 - AOA is not the only method



J. Armstrong, Y. Sekercioğlu, e A. Neild, «Visible light positioning: a roadmap for international standardization», *Communications Magazine, IEEE*, vol. 51, n. 12, pp. 68–73, Dez. 2013.



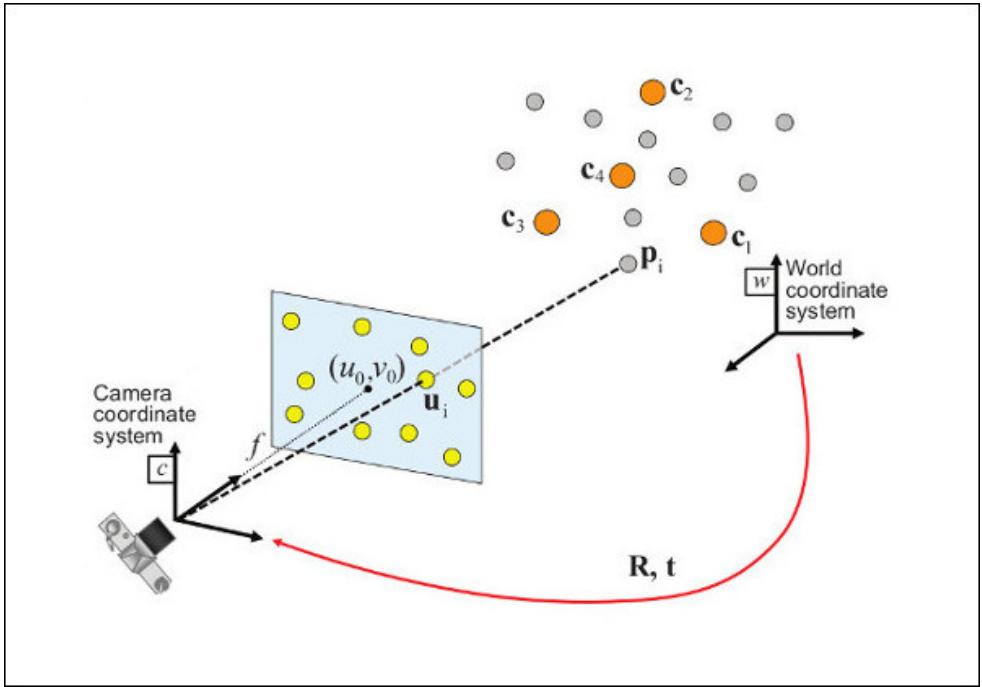
S.-Y. Jung, S. Hann, S. Park, e C.-S. Park, «Optical wireless indoor positioning system using light emitting diode ceiling lights», *Microwave and Optical Technology Letters*, vol. 54, n. 7, pp. 1622–1626, 2012.

Visible Light Positioning

- Perspective-N-Point

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{A} \Pi^c \mathbf{T}_w \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

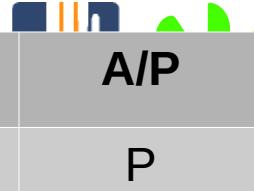


This matrix contains the camera (~robot) position coordinates in the world

visible light positioning (VLP)

sensor classification

- Proprioceptive / exteroceptive
 - Proprioceptive information internal to the robot.
 - Ex: motor speed, battery voltage, ...
 - Exteroceptive: external information to the robot
 - Ex.: distance to objects, light intensity
- Passive / Active
 - Passive: have no explicit source of energy; energy required comes from the measurement process itself
 - Ex temperature probes ..
 - Active: have internal power source, necessary for the measurement process; uses that energy to interact with the environment
 - Ex.: laser range finder



un
de

Classification	Sensor	PC/EC	A/P
Tactile Sensors	Switches	EC	P
	Optical barrier	EC	A
	Contactless proximity sensors	EC	A
Wheels and motors	Potentiometers	PC	P
	Synchros and resolvers	PC	A
	Optical encoders	PC	A
Orientation	Compass	EC	P
	Gyroscope	PC	P
	Inclinometer	EC	A/P
Localization	GPS	EC	A
	RF beacons	EC	A
	Reflected beams	EC	A
Ranging	Sensores ultrassons	EC	A
	Laser Range Finder	EC	A
	Optical triangulation	EC	A
Vision	CCD/CMOS cameras	EC	P

R. Siegwart e I. R. Nourbakhsh, *Introduction to autonomous mobile robots*. Cambridge Mass.: MIT Press, 2004.

- Think of a mobile robot mission
- Devise the sensor structure required for that mission
 - What sensors will the robot need?

Robótica Móvel e Inteligente / Mobile and Intelligent Robotics

Academic year 2022-23

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

CC-BY-SA The IRIS team, 2022

Control systems

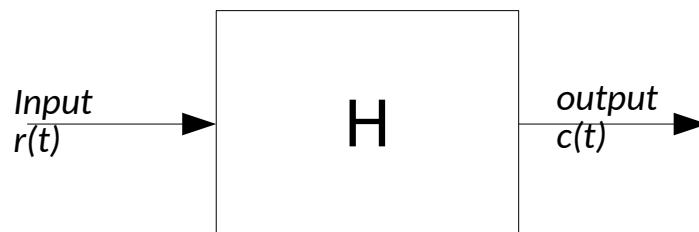
- **Objective:** to impose a given value of some physical quantity in a system by acting on some other physical quantity



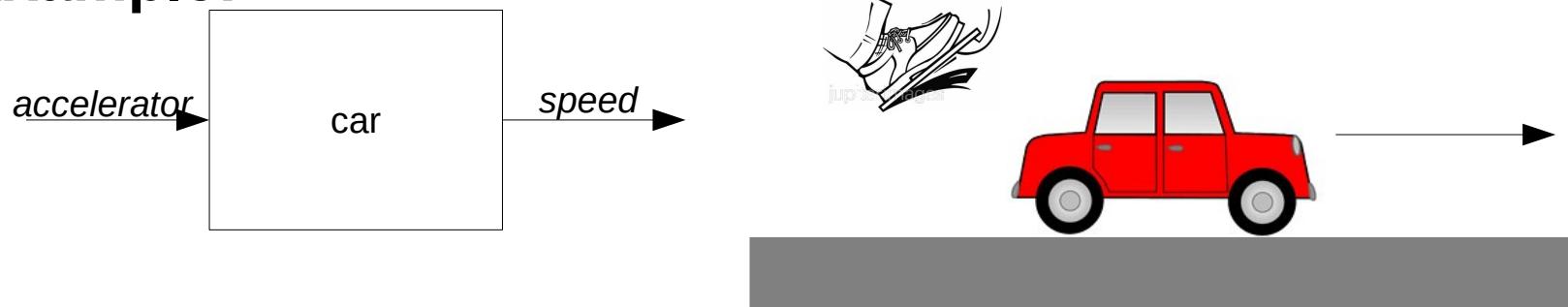
<http://blog.caranddriver.com/nissan-develops-fully-electric-steer-by-wire-system-will-go-on-sale-next-year/>

basic concepts

- **Systems approach:**
 - Input signal
 - Output signal
 - Process, transforming input into output
- **Objective: to impose a given value at a system's output, by acting in its input**

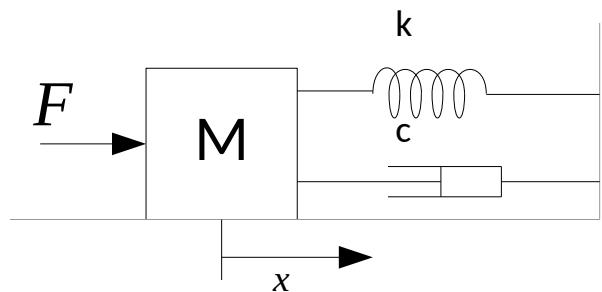


- **Example:**



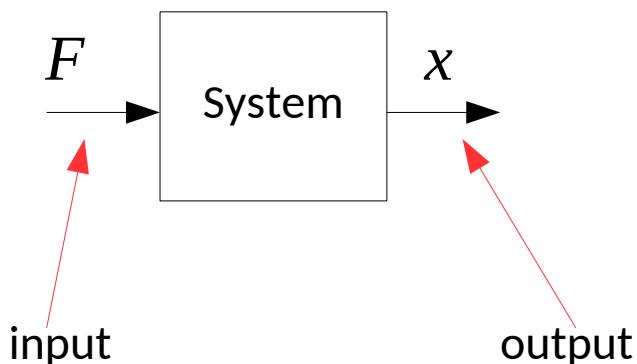
Input / output relationship

- **General case:**
 - input $r(t)$ and output $c(t)$ are related by differential equations
 - this is the “default” in physical systems...



Mathematical relation
between input and output

$$F(t) = M a(t) + c v(t) + k x(t)$$

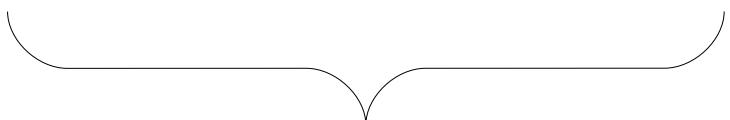


$$F(t) = M \frac{d^2 x(t)}{dt^2} + c \frac{dx(t)}{dt} + k x(t)$$

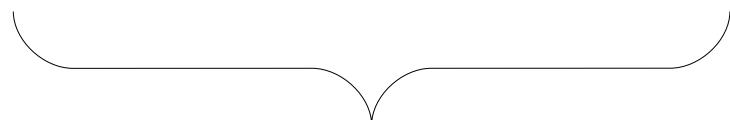
Input / output relationship

- **General case:**
 - input $r(t)$ and output $c(t)$ are related by differential equations

$$a_n \frac{d^n c(t)}{dt^n} + a_{n-1} \frac{d^{n-1} c(t)}{dt^{n-1}} + \dots + a_0 c(t) = b_m \frac{d^m r(t)}{dt^m} + b_{m-1} \frac{d^{m-1} r(t)}{dt^{m-1}} + \dots + b_0 r(t)$$

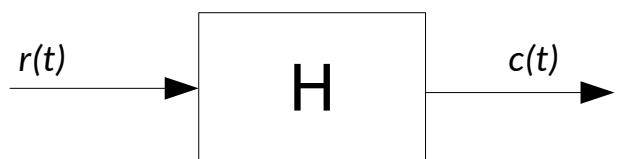
A curly brace grouping the terms $a_n \frac{d^n c(t)}{dt^n}, a_{n-1} \frac{d^{n-1} c(t)}{dt^{n-1}}, \dots, a_0 c(t)$.

Combination of $c(t)$ and
its derivatives

A curly brace grouping the terms $b_m \frac{d^m r(t)}{dt^m}, b_{m-1} \frac{d^{m-1} r(t)}{dt^{m-1}}, \dots, b_0 r(t)$.

Combination of $r(t)$ and
its derivatives

Difficult to solve and convert to a systems perspective

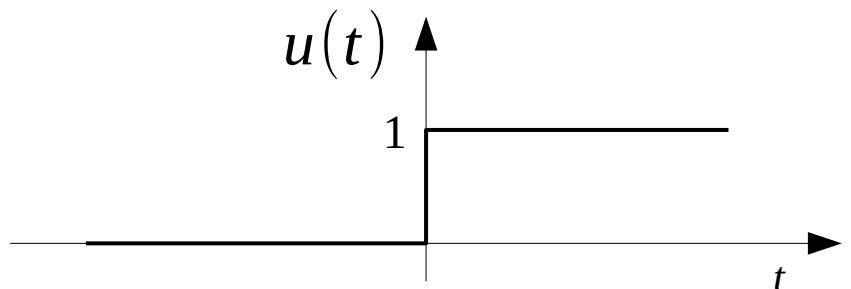


Laplace transform

- Differential equations are simplified by the use of Laplace transforms.

$$L\{f(t)\} = F(s) = \int_0^{+\infty} e^{-st} f(t) dt$$

$$L^{-1}\{F(s)\} = \frac{1}{2\pi j} \int_{\sigma-j\infty}^{\sigma+j\infty} e^{st} F(s) dt = f(t) \cdot u(t)$$



$u(t)$ is the unit step function. We only consider the function $f(t)$ to have non-null values for $t > 0$.

Computing the Laplace transform



- Example for $u(t)$

$$\begin{aligned} L\{u(t)\} &= \int_0^{+\infty} e^{-st} u(t) dt \\ &= \int_0^{+\infty} e^{-st} dt \\ &= \left[-\frac{1}{s} e^{-st} \right]_0^{+\infty} \\ &= 0 - \left(-\frac{1}{s} \right) \\ &= \frac{1}{s} \end{aligned}$$

Laplace transform

TABLE 2.1 Laplace transform table

Item no.	$f(t)$	$F(s)$
1.	$\delta(t)$	1
2.	$u(t)$	$\frac{1}{s}$
3.	$tu(t)$	$\frac{1}{s^2}$
4.	$t^n u(t)$	$\frac{n!}{s^n + 1}$
5.	$e^{-at} u(t)$	$\frac{1}{s + a}$
6.	$\sin \omega t u(t)$	$\frac{\omega}{s^2 + \omega^2}$
7.	$\cos \omega t u(t)$	$\frac{s}{s^2 + \omega^2}$

Laplace transform

Laplace transform theorems

Theorem	Name
$\mathcal{L}[f(t)] = F(s) = \int_{0-}^{\infty} f(t)e^{-st}dt$	Definition
$\mathcal{L}[kf(t)] = kF(s)$	Linearity theorem
$\mathcal{L}[f_1(t) + f_2(t)] = F_1(s) + F_2(s)$	Linearity theorem
$\mathcal{L}[e^{-at}f(t)] = F(s+a)$	Frequency shift theorem
$\mathcal{L}[f(t-T)] = e^{-sT}F(s)$	Time shift theorem
$\mathcal{L}[f(at)] = \frac{1}{a}F\left(\frac{s}{a}\right)$	Scaling theorem
$\mathcal{L}\left[\frac{df}{dt}\right] = sF(s) - f(0-)$	Differentiation theorem
$\mathcal{L}\left[\frac{d^2f}{dt^2}\right] = s^2F(s) - sf(0-) - f'(0-)$	Differentiation theorem
$\mathcal{L}\left[\frac{d^n f}{dt^n}\right] = s^n F(s) - \sum_{k=1}^n s^{n-k} f^{k-1}(0-)$	Differentiation theorem
$\mathcal{L}\left[\int_{0-}^t f(\tau)d\tau\right] = \frac{F(s)}{s}$	Integration theorem
$f(\infty) = \lim_{s \rightarrow 0} sF(s)$	Final value theorem ¹
$f(0+) = \lim_{s \rightarrow \infty} sF(s)$	Initial value theorem ²

Laplace transform

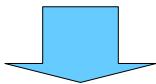
$$a_n \frac{d^n c(t)}{dt^n} + a_{n-1} \frac{d^{n-1} c(t)}{dt^{n-1}} + \dots + a_0 c(t) = b_m \frac{d^m r(t)}{dt^m} + b_{m-1} \frac{d^{m-1} r(t)}{dt^{m-1}} + \dots + b_0 r(t)$$

Computing the
Laplace transform

$$\begin{aligned}
 \rightarrow L\left\{a_n \frac{d^n c(t)}{dt^n}\right\} &= a_n L\left\{\frac{d^n c(t)}{dt^n}\right\} && \text{Linearity} \\
 &= a_n s^n L\{c(t)\} && \text{Differentiation theorem} \\
 &= a_n s^n C(s) && \text{Definition of Laplace transform}
 \end{aligned}$$

Laplace transform

$$a_n \frac{d^n c(t)}{dt^n} + a_{n-1} \frac{d^{n-1} c(t)}{dt^{n-1}} + \dots + a_0 c(t) = b_m \frac{d^m r(t)}{dt^m} + b_{m-1} \frac{d^{m-1} r(t)}{dt^{m-1}} + \dots + b_0 r(t)$$



$$a_n s^n C(s) + a_{n-1} s^{n-1} C(s) + \dots + a_0 C(s) = b_m s^m R(s) + b_{m-1} s^{m-1} R(s) + \dots + b_0 R(s)$$

$$(a_n s^n + a_{n-1} s^{n-1} + \dots + a_0) C(s) = (b_m s^m + b_{m-1} s^{m-1} + \dots + b_0) R(s)$$

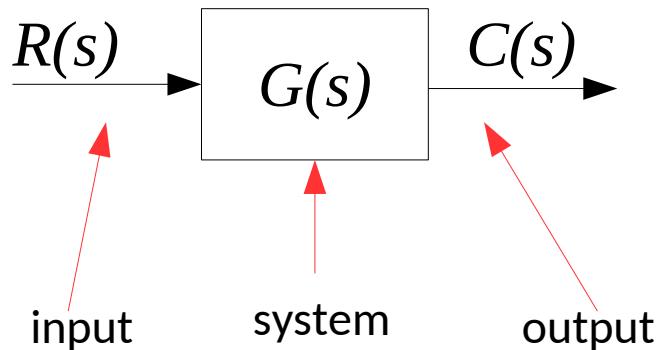
$$\frac{C(s)}{R(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_0} = G(s)$$

Transfer function

$$\frac{C(s)}{R(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_0} = G(s)$$

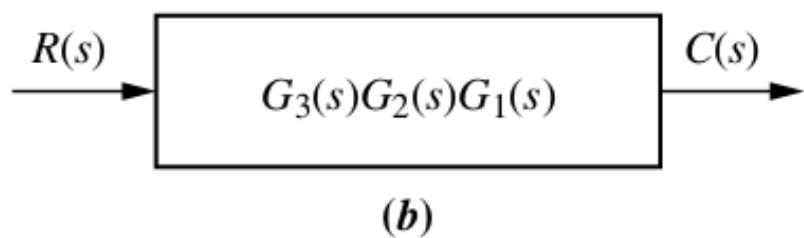
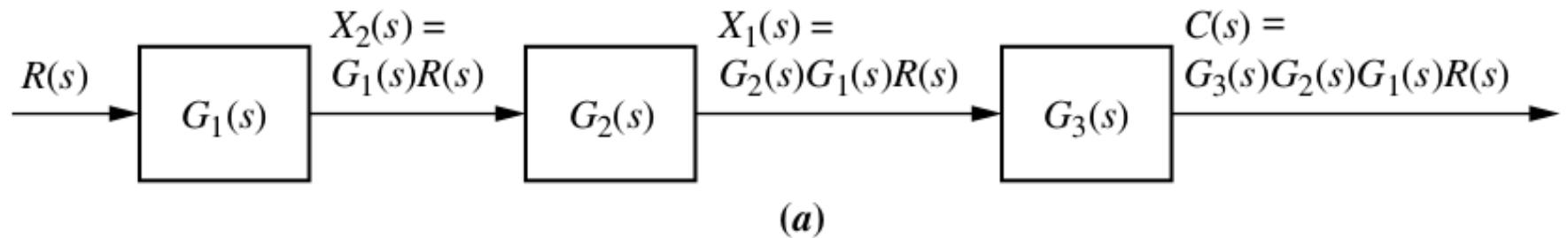
Transfer
function

$$C(s) = G(s) \cdot R(s)$$

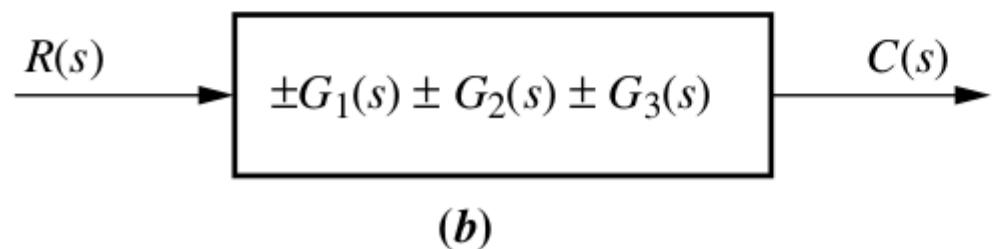
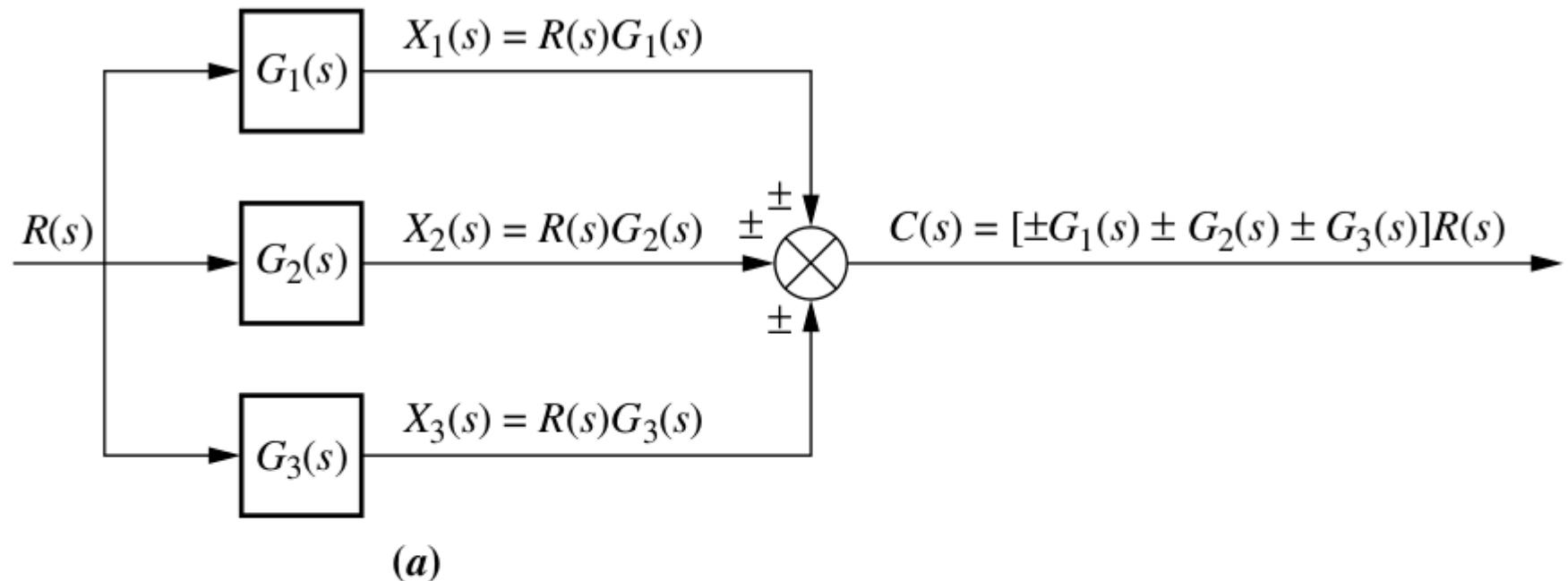


- A relation expressed originally in terms of a differential equation is expressed as a product
- the physical nature of input/output relationship is irrelevant; only mathematical relationship matters --> **abstraction**

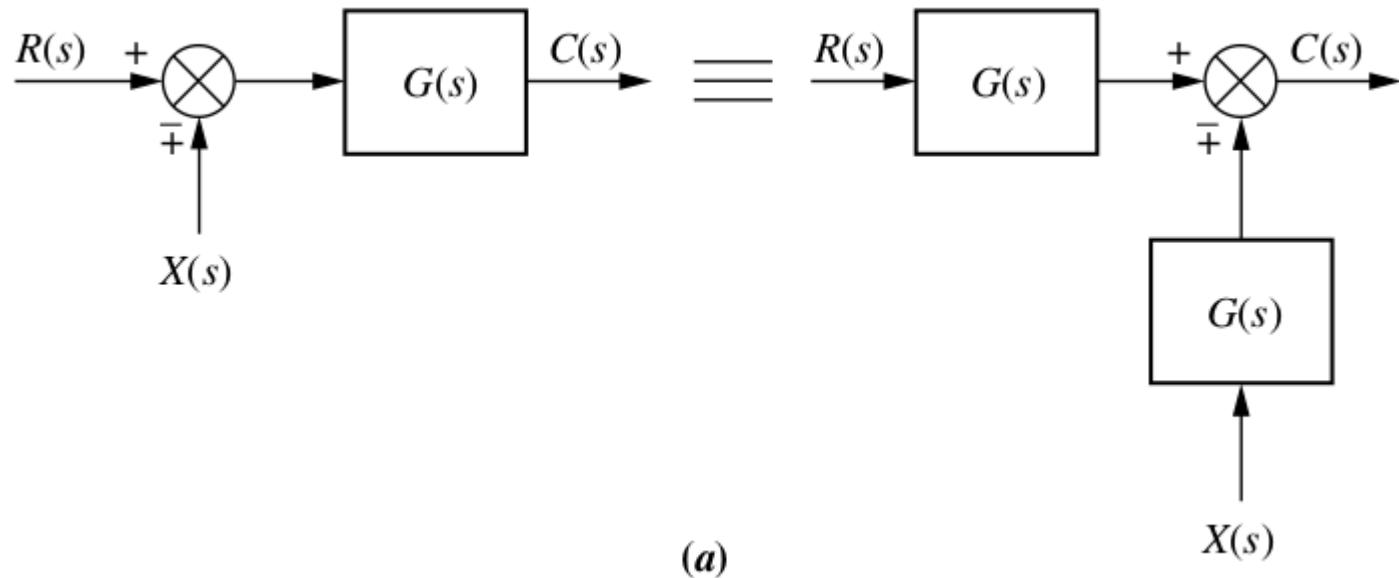
Block diagram algebra



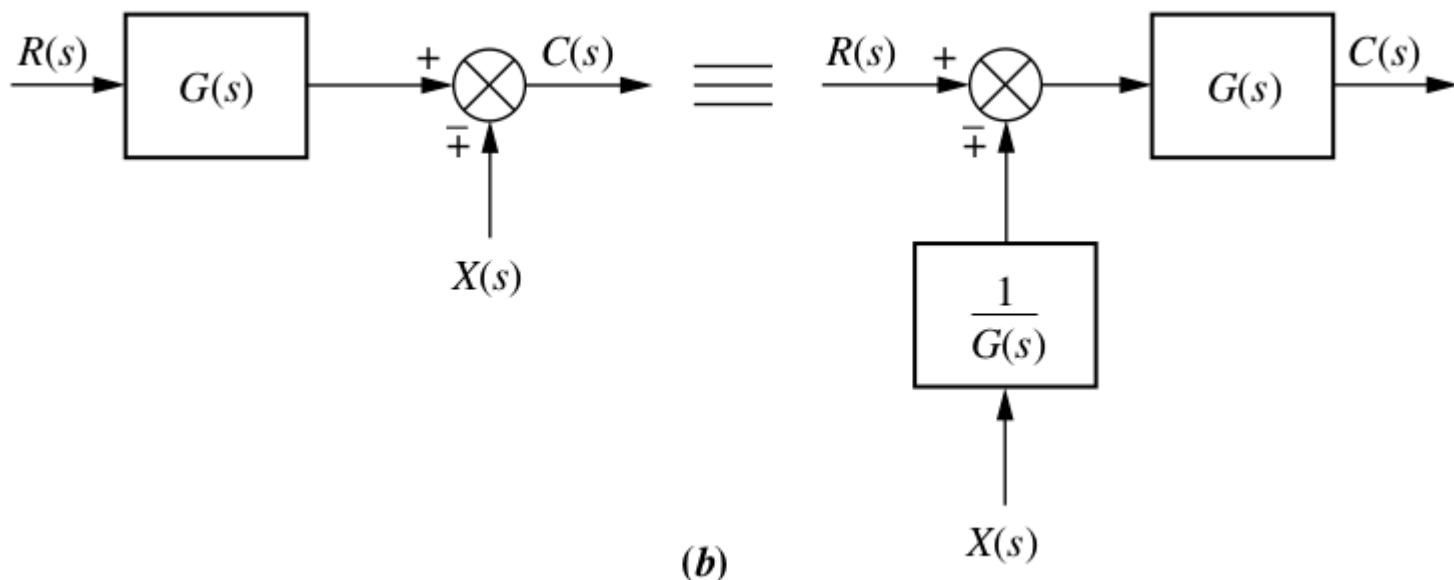
Block diagram algebra



Block diagram algebra

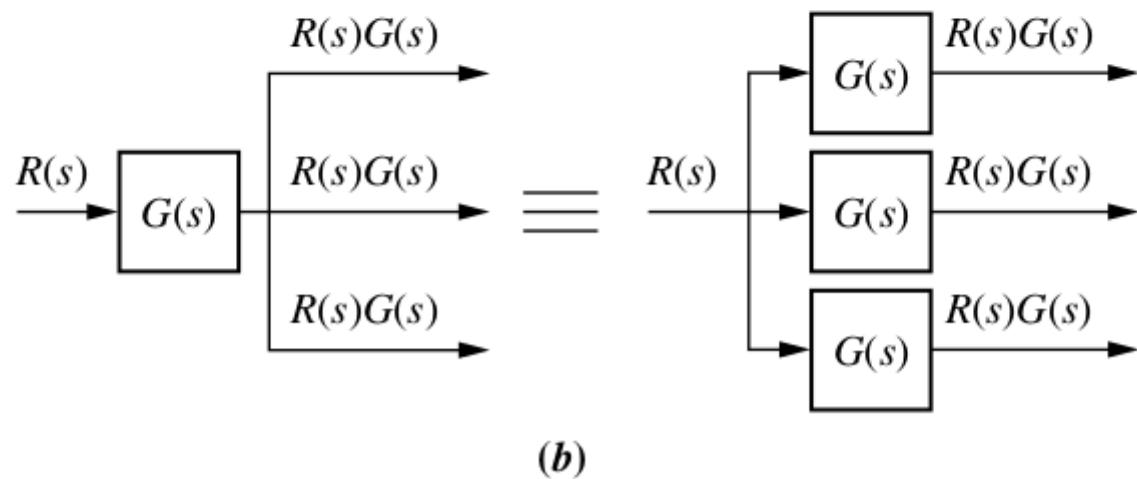
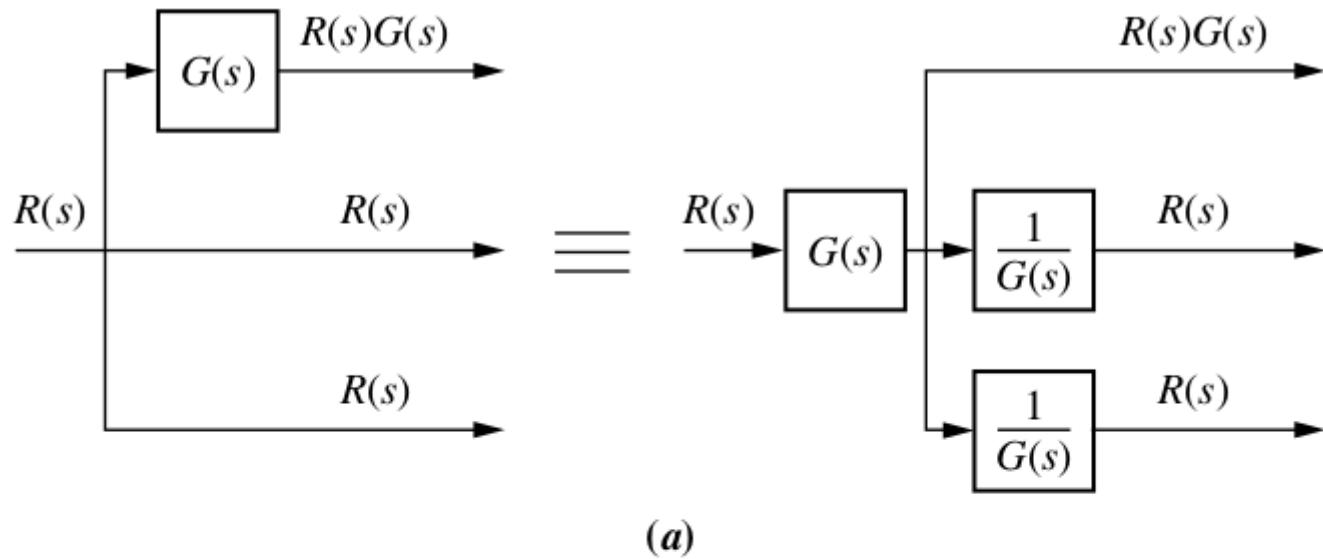


(a)



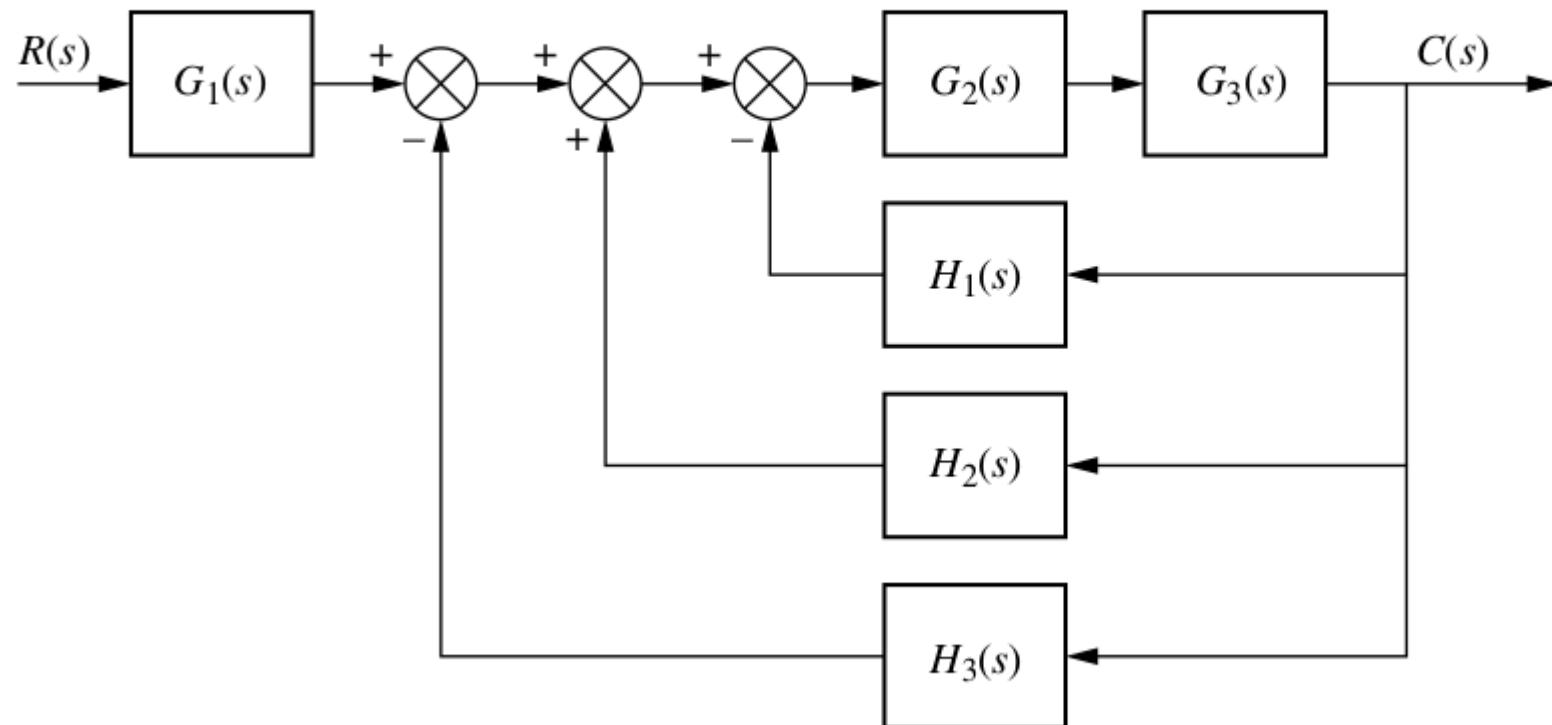
(b)

Block diagram algebra

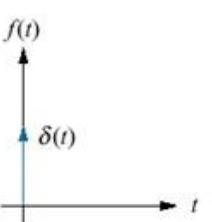
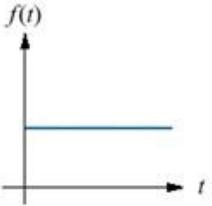
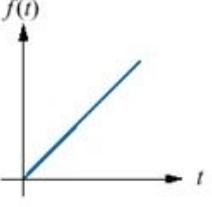
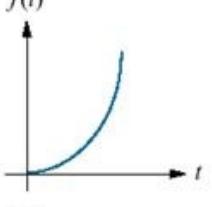
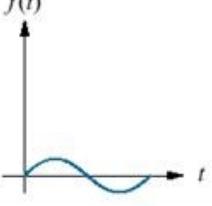


Block diagram algebra

PROBLEM: Reduce the block diagram shown in Figure 5.9 to a single transfer function.



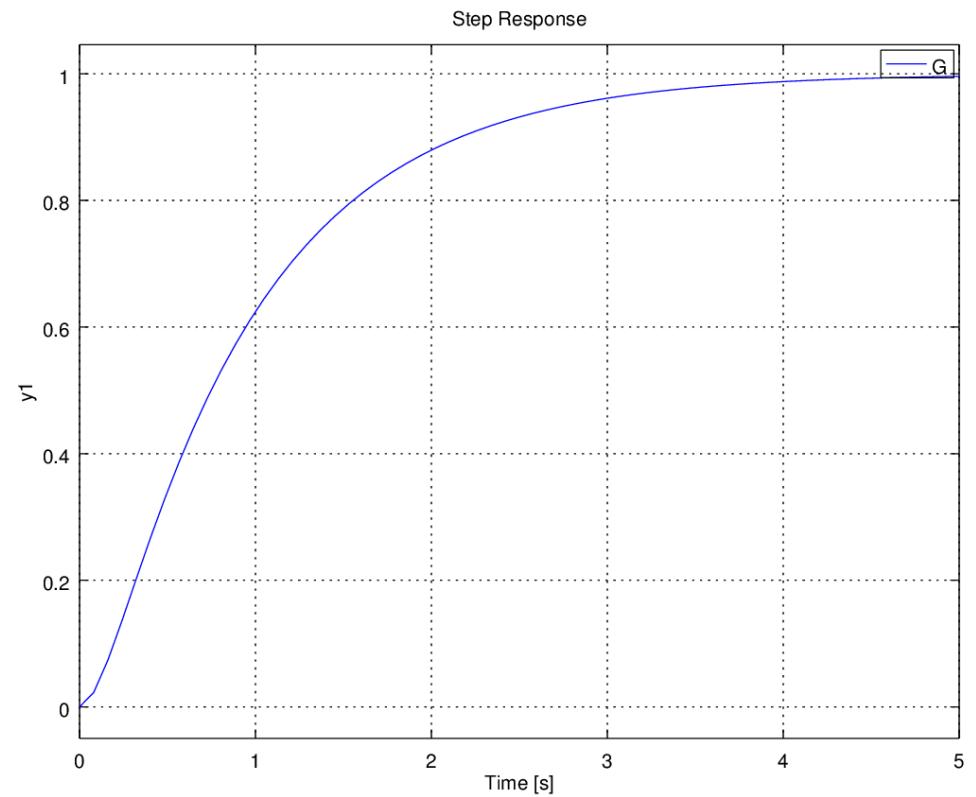
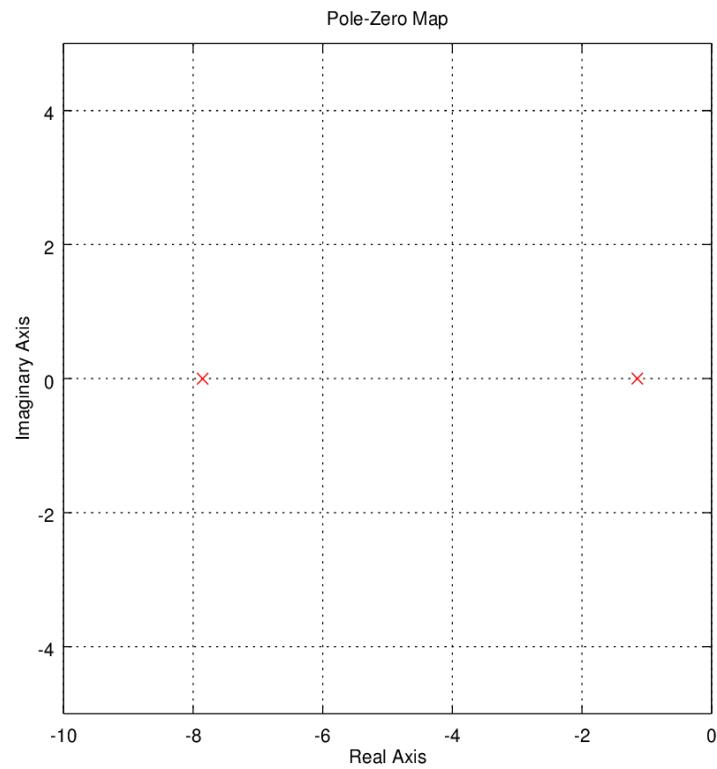
Test waveforms

Input	Function	Description	Sketch	Use
Impulse	$\delta(t)$	$\delta(t) = \infty \text{ for } 0- < t < 0+$ $= 0 \text{ elsewhere}$ $\int_{0-}^{0+} \delta(t) dt = 1$		Transient response Modeling
Step	$u(t)$	$u(t) = 1 \text{ for } t > 0$ $= 0 \text{ for } t < 0$		Transient response Steady-state error
Ramp	$tu(t)$	$tu(t) = t \text{ for } t \geq 0$ $= 0 \text{ elsewhere}$		Steady-state error
Parabola	$\frac{1}{2}t^2 u(t)$	$\frac{1}{2}t^2 u(t) = \frac{1}{2}t^2 \text{ for } t \geq 0$ $= 0 \text{ elsewhere}$		Steady-state error
Sinusoid	$\sin \omega t$			Transient response Modeling Steady-state error

Poles and step response

$$r(t) = u(t) \quad G_1(s) = \frac{9}{s^2 + 9s + 9}$$

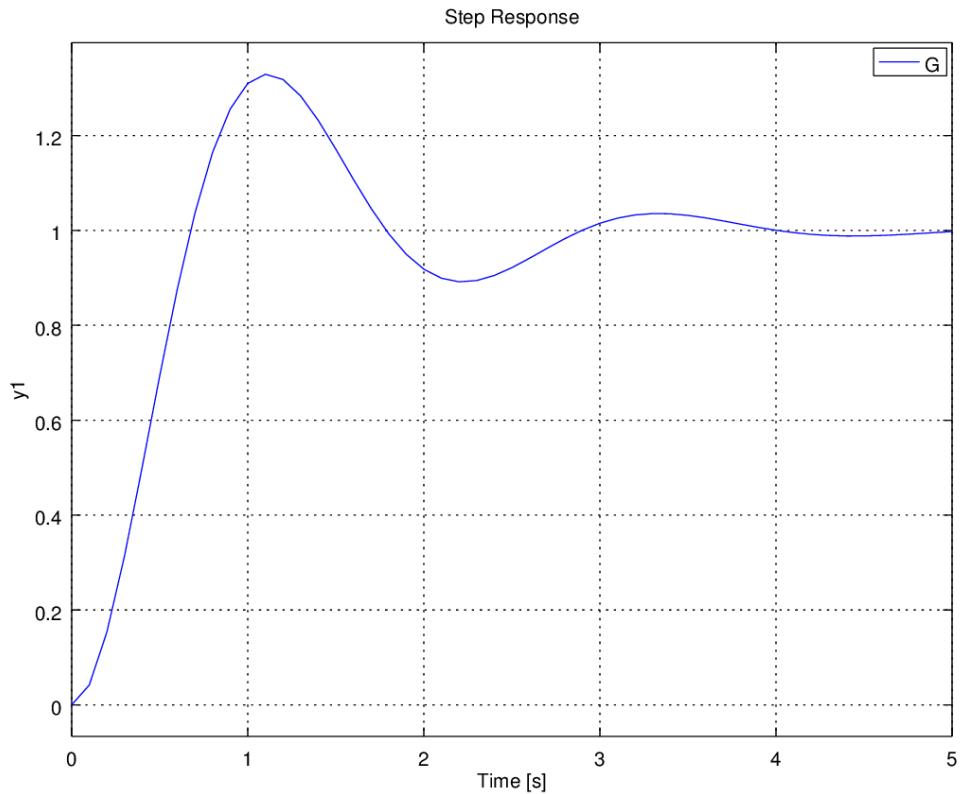
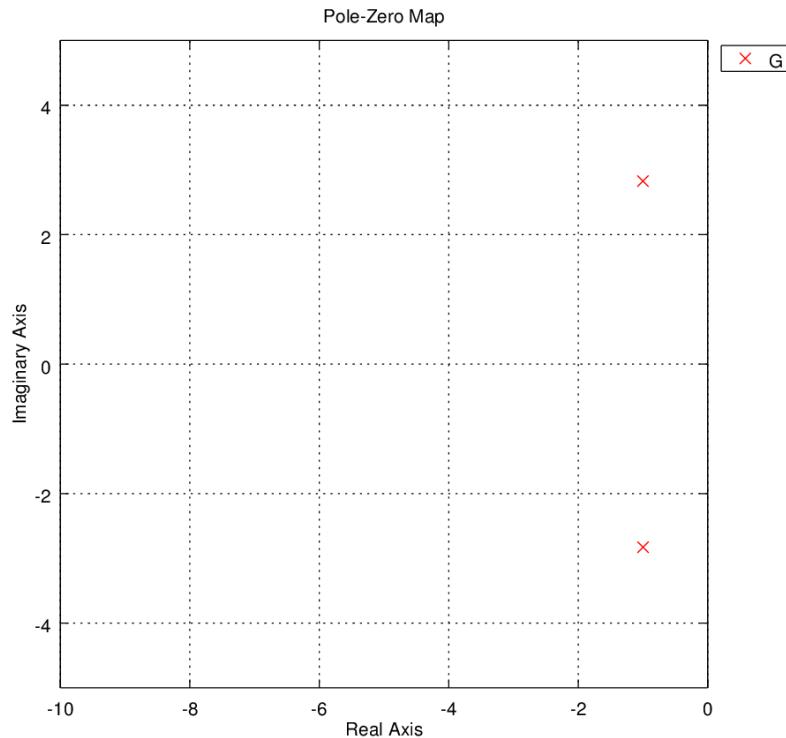
Overdamped



Poles and step response

$$r(t) = u(t) \quad G_2(s) = \frac{9}{s^2 + 2s + 9}$$

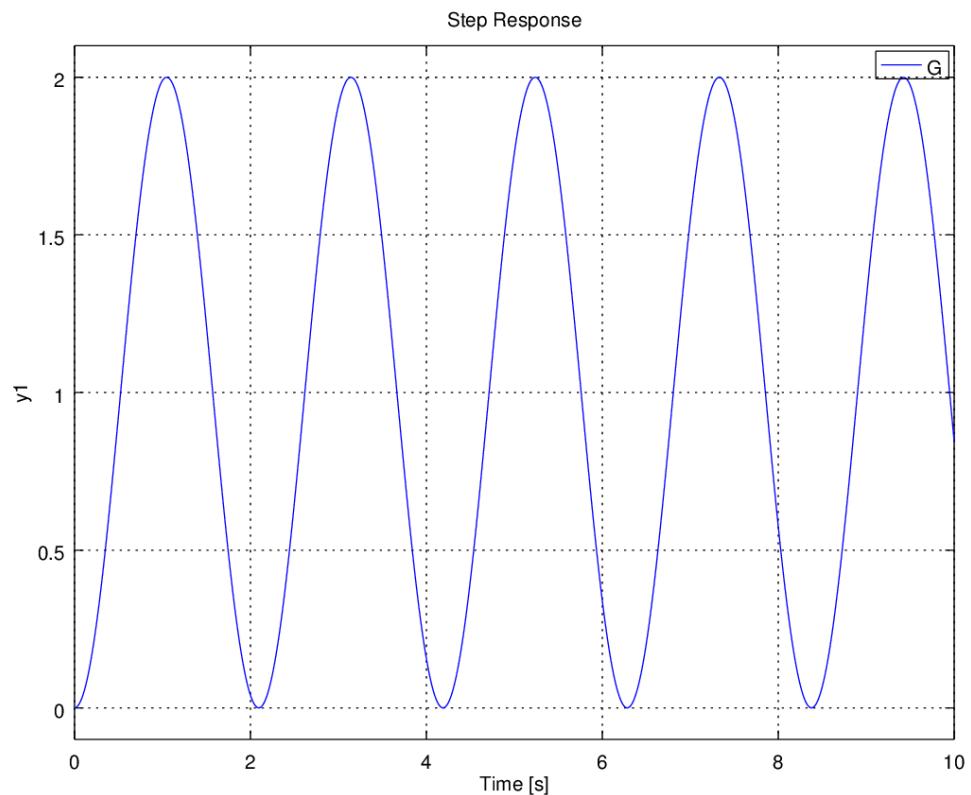
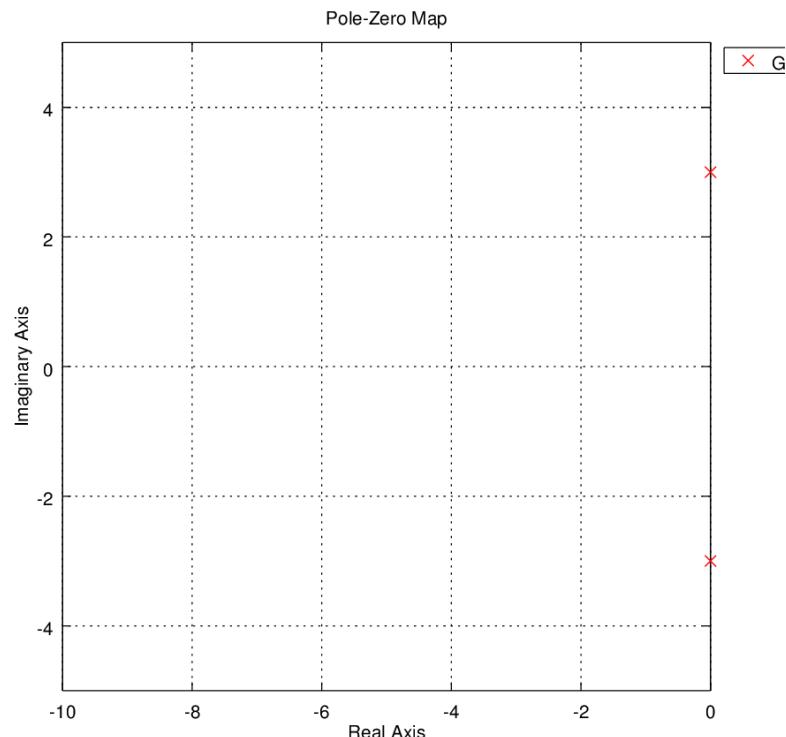
Underdamped



Poles and step response

$$r(t) = u(t) \quad G_3(s) = \frac{9}{s^2 + 9}$$

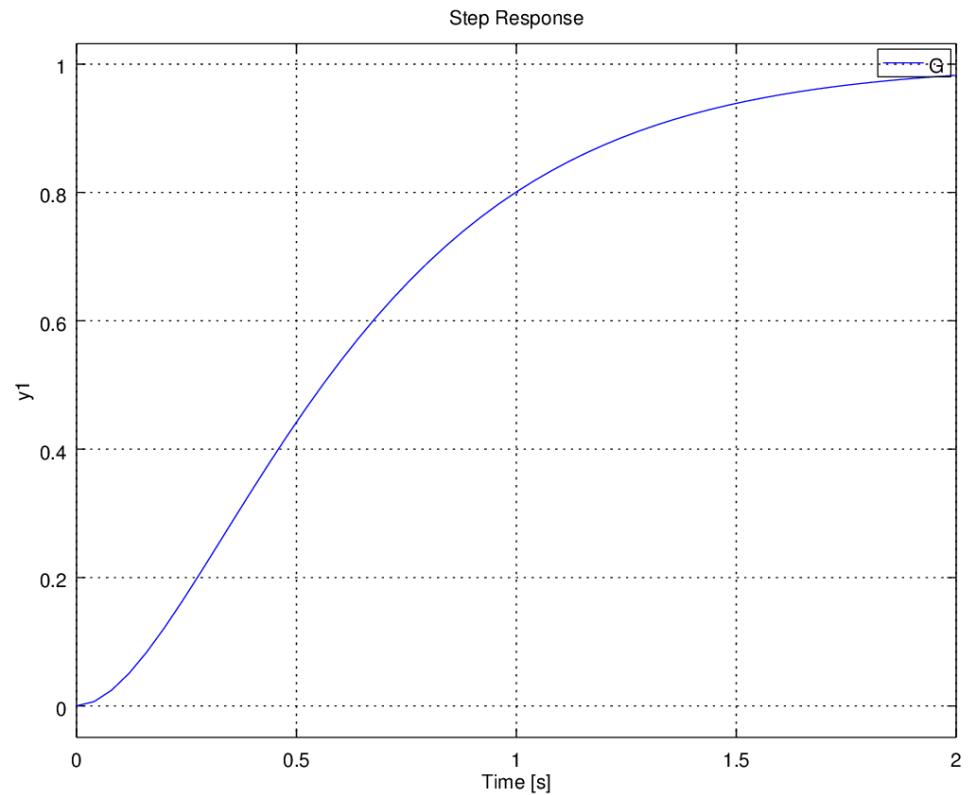
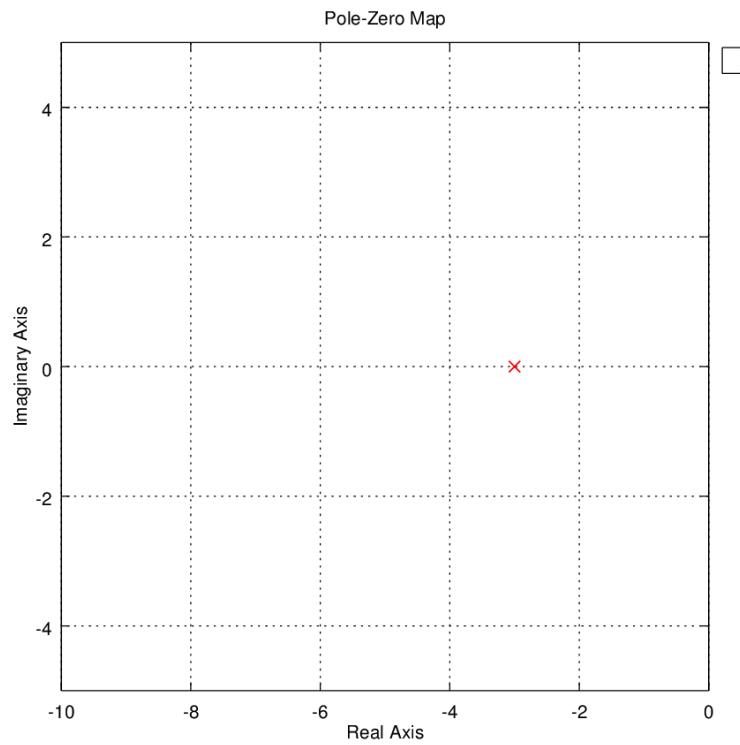
Undamped



Poles and step response

$$r(t) = u(t) \quad G_4(s) = \frac{9}{s^2 + 6s + 9}$$

Critically damped

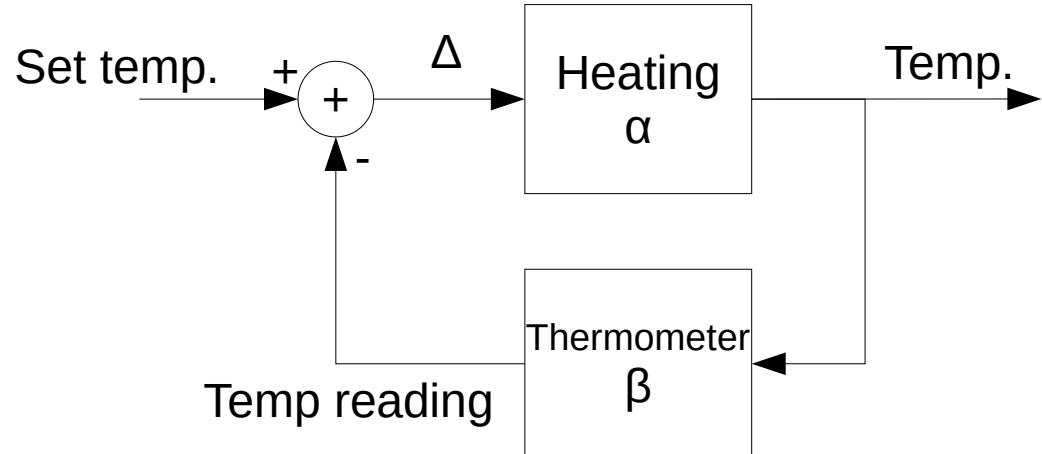
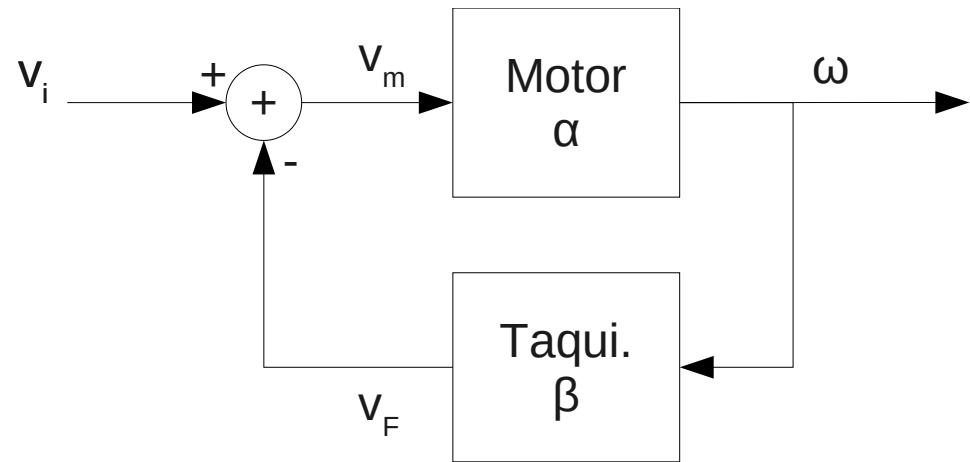


Video

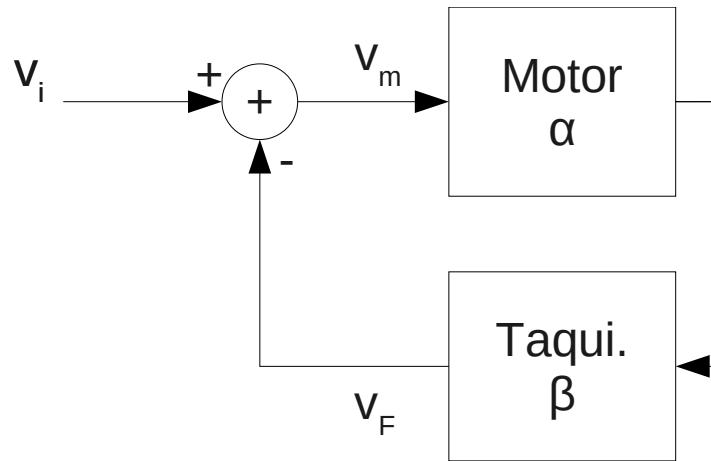


feedback

- a complex system is represented as a collection of interconnected set of simpler systems
 - each simple system has a known transfer function



feedback



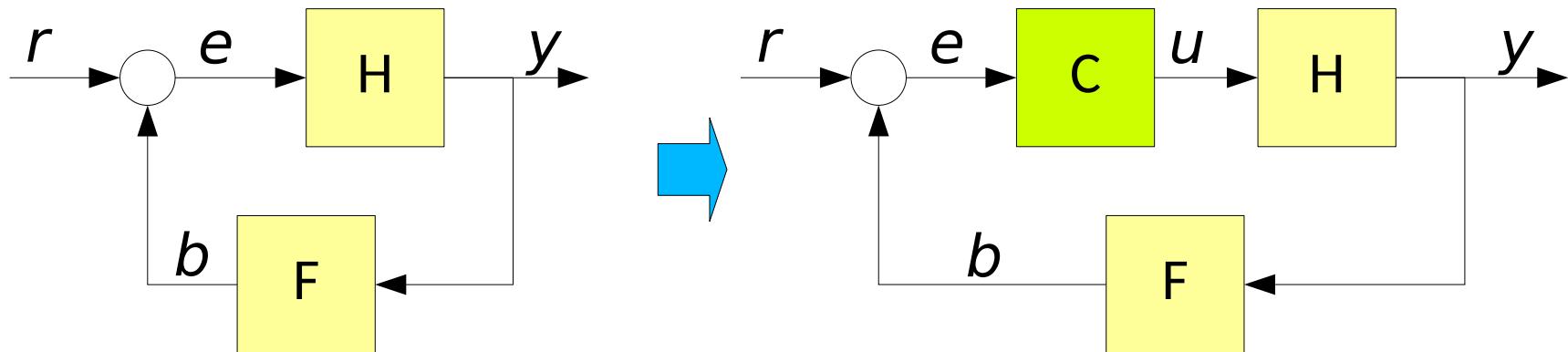
$$\begin{cases} v_m = v_i - v_T \\ \omega = \alpha v_m \\ v_F = \beta \omega \end{cases} \Rightarrow \omega = \frac{\alpha}{1 + \alpha\beta} v_i$$

$$\lim_{\alpha \rightarrow \infty} \frac{\alpha}{1 + \alpha\beta} = \frac{1}{\beta}$$

- for high values of α , the output value will depend mainly on the feedback

controller

- Controller C: included to improve the system response



r : reference (input)

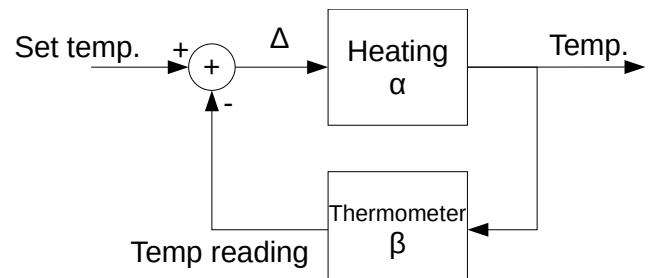
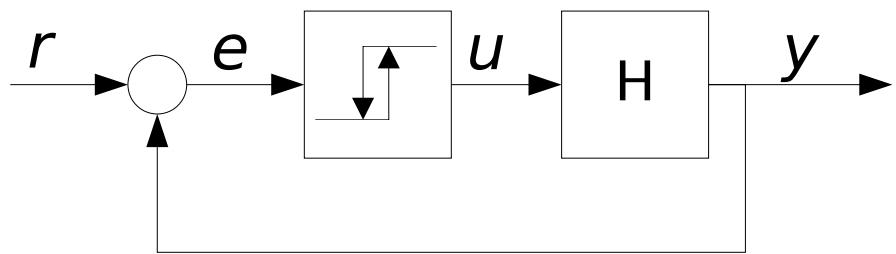
e : error

u : system input / control

y : system output

b : feedback

On-Off (Bang-bang)



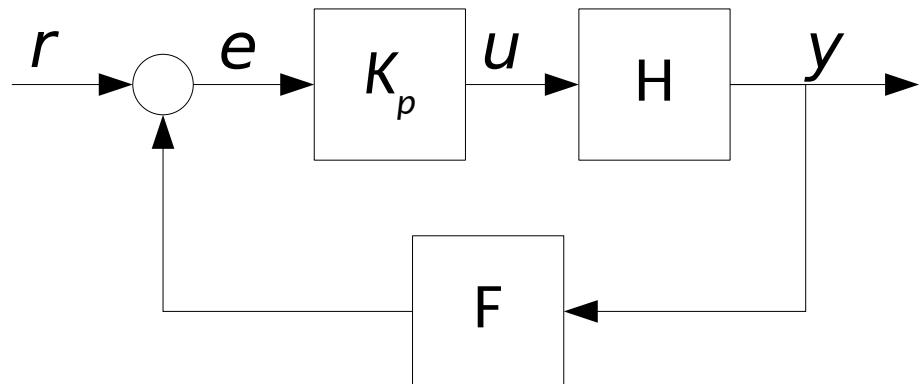
- Acting signal u goes on and off depending on error
- Example:
 - home temperature control
- May have hysteresis
- Usually not so good behaviour, but...
- Easy to implement (switch on and off...)

P controller

- P = “proportional”. Simplest form of linear controller

$$u = K_p e$$

- Gain: K_p



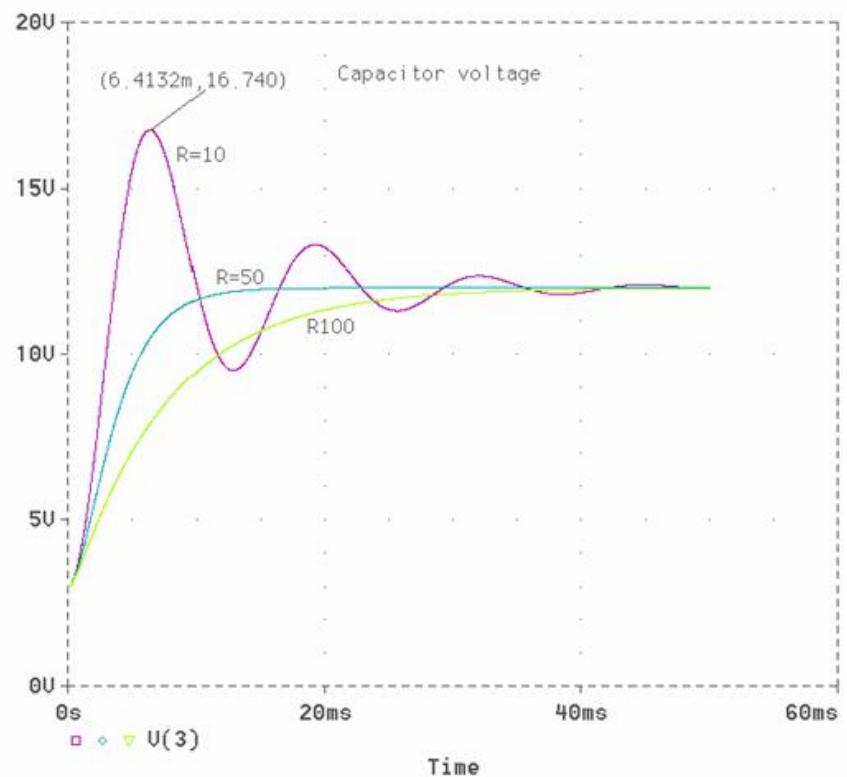
The larger K_p , the smaller e for the same output.

- K_p low: soft system
 - it takes a large value of error for system to react
- K_p high: hard system
 - strong reaction, even with small values of e .

$$u \neq 0 \text{ iff } e \neq 0$$

To reduce error, a high value of K_p is required

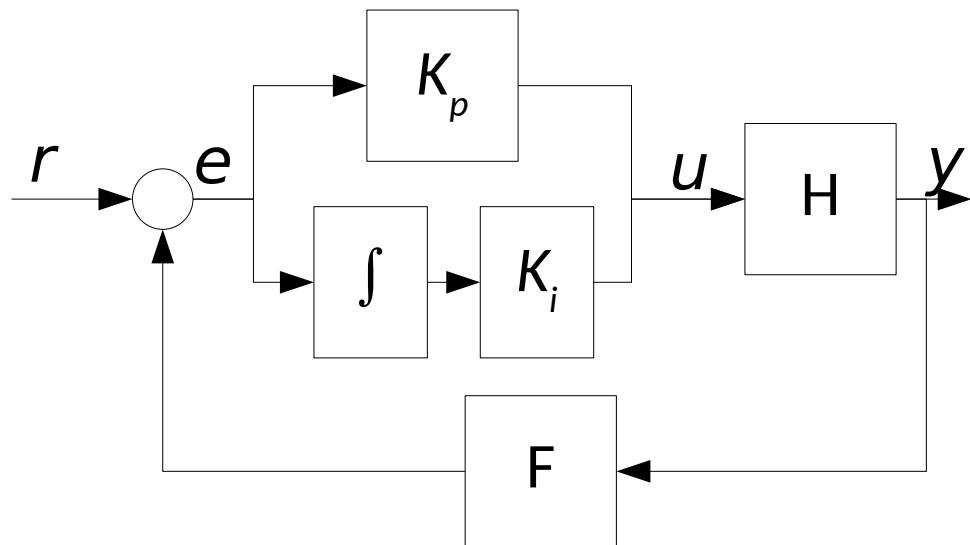
P controller



- Increasing gain K_p reduces error, but...
- High values of gain K_p may cause the system to be unstable

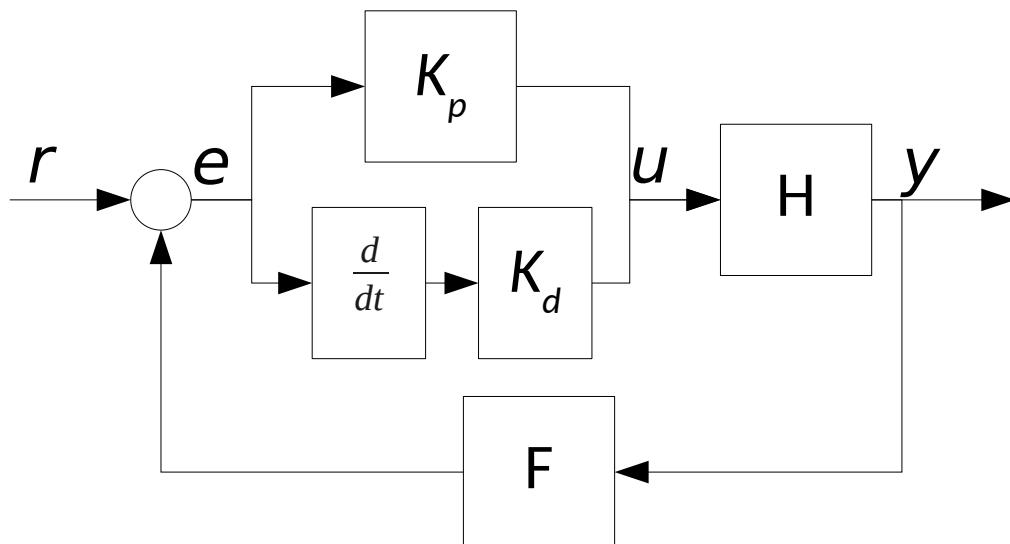
PI controller

- **PI: Proportional + Integral**
 - Adding a term $K_i \int e dt$.
 - PI controller allows for systems with $e=0$
 - Problem: inertia (memory effect)
 - with rapid changes in the input, u may be at a value when the error e would require to be different



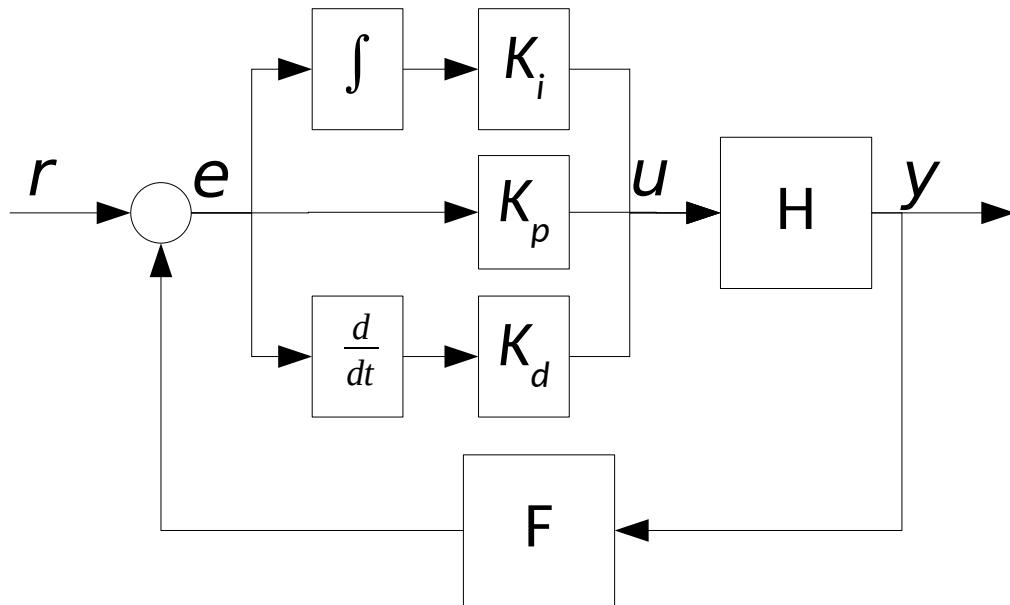
PD controller

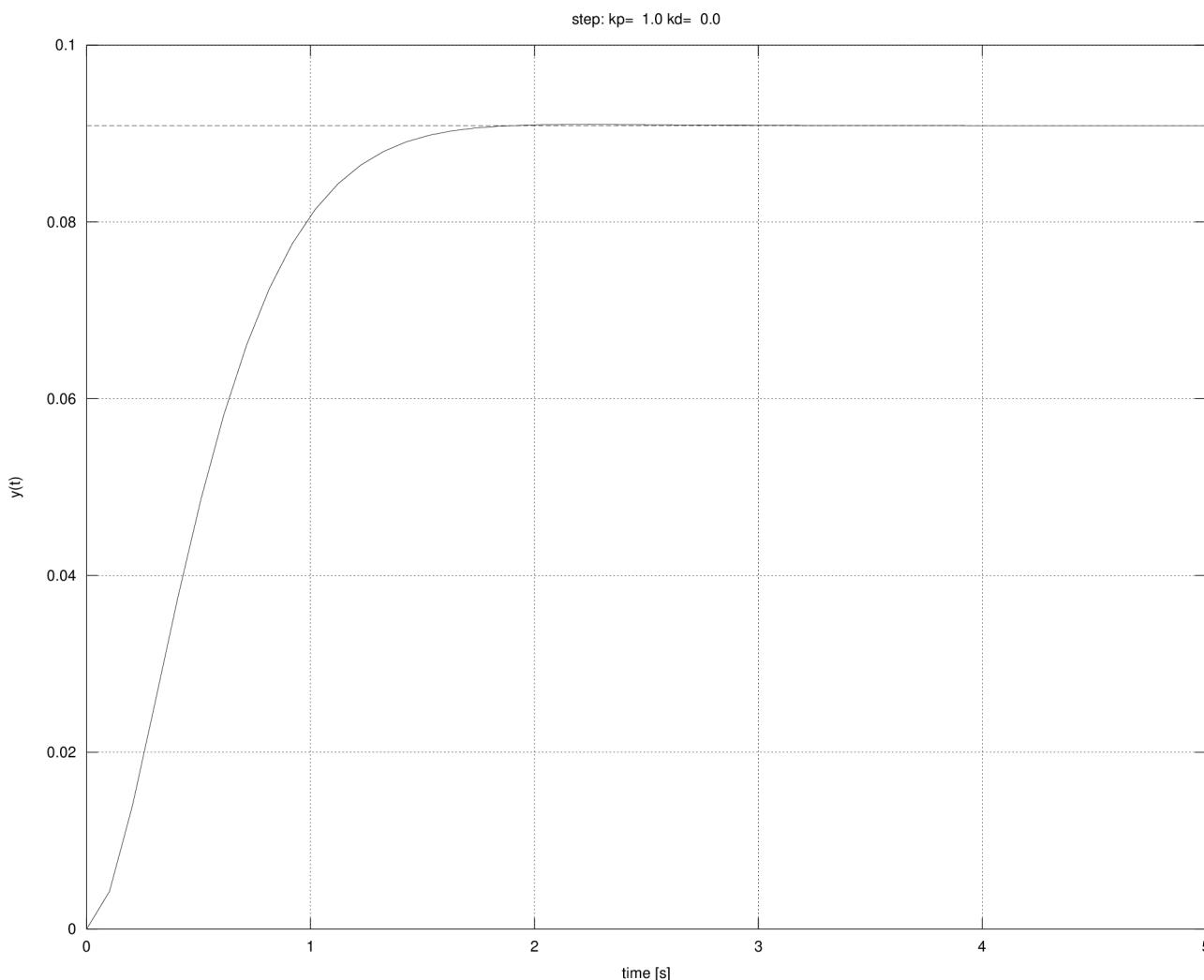
- **PD = Proportional + Derivative**
 - adding a term proportional to the error derivative
 - In general, it has the effect of reducing oscillations (damper)



PID controller

- **PID = Proportional + Integral + Derivative**
 - Reunion of previous controllers
 - One of the most popular controllers



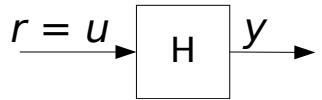


Coding a PID controller

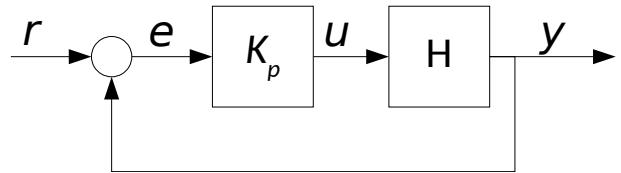


Controller demo

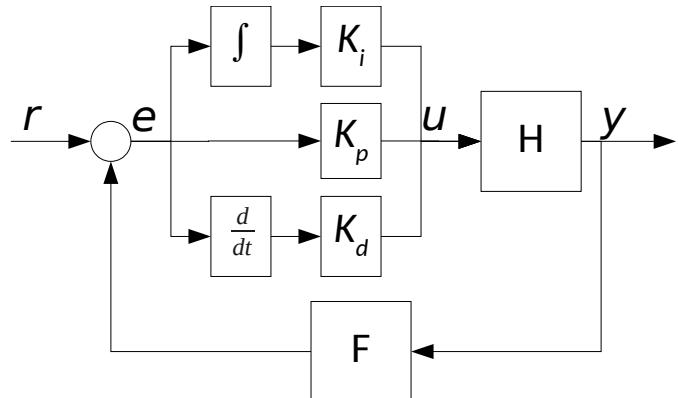
NONE



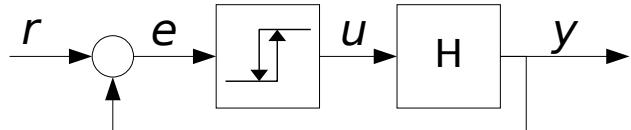
P



PID



BANG,
BANG2,
BANGH



Sensor Fusion

Robótica Móvel e Inteligente

José Luís Azevedo, Bernardo Cunha, Pedro Fonseca,
Nuno Lau e Artur Pereira

Ano Letivo 2022/2023
IRIS/IEETA – DETI – Universidade de Aveiro

Outline

- Introduction to Sensor Fusion
- Kalman Filter
- Particle Filter
- Conclusion

Sensor Fusion

- **Act of combining sensory data** or data derived from sensory data from disparate sources
- **The resulting information is “better”** than it would be possible when the sources were used individually
- “**Better**” is defined according to the context. Can be **more accurate, more complete, a different view**, etc.
- Using a broader definition, we can speak of **Information Fusion**
- Sensor Fusion is usually considered a subset of information fusion, although the terms are often used with the same meaning
- Another very used term for this task is Multi-Sensor Data Fusion

Sensor Fusion

- Usually based on **modeling the sensors** and **modeling the system** being measured
- Most common methodologies are **probability based**
- However, some methodologies try to overcome some limitations of probability models (complexity, inconsistency, precision of models, uncertainty about uncertainty)
- Some of these methodologies are:
 - Interval calculus
 - Fuzzy logic
 - Theory of evidence (Dempster-Shafer methods)

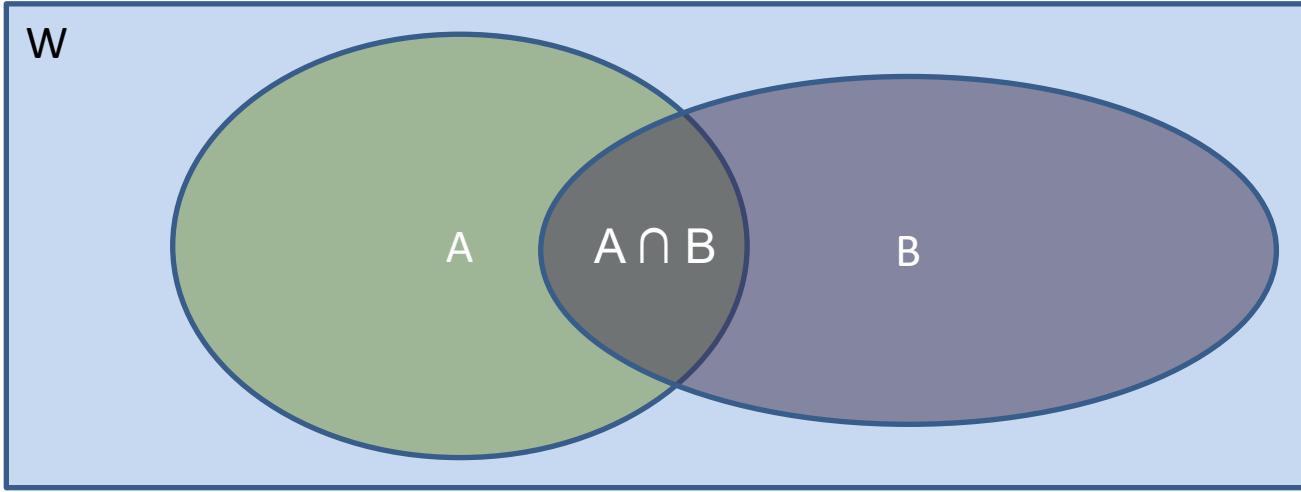
Bayes Rule

- Determine the probability of a state/event, given the result of other states/events that are related.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

- $P(A | B)$ is the probability of A given that B is true, called **conditional probability**
- $P(B | A)$ is the probability of B given that A is true, called conditional probability
- $P(A)$ and $P(B)$ are the **marginal probabilities** of A and B

Bayes Rule



- Consider that the probability of a given event is related to its area in the above diagram, and that $\text{area}(W) = 1$
- $P(A) = \text{area}(A)$ $P(B) = \text{area}(B)$
- $$P(A | B) = \frac{\text{area}(A \cap B)}{\text{area}(B)} = \frac{\frac{\text{area}(A \cap B)}{\text{area}(A)} \cdot \frac{\text{area}(A)}{\text{area}(W)}}{\text{area}(B)} = \frac{P(B | A) \cdot P(A)}{P(B)}$$

Bayes Rule

- Applying Bayes Rule to determine the probability of being sick if the test for the disease is positive (+).
 - Assumptions:
 - Tests are correct 99% (test are positive with 99% probability if the person is sick, and negative with 99% if the person is not sick)
 - Disease is rare, happening only 1 in every 10000 people
 - If a person is tested and result is positive which is the probability of being sick?

<https://math.hmc.edu/funfacts/medical-tests-and-bayes-theorem/>

Bayes Rule

- Applying Bayes Rule to determine the probability of being sick if the test for the disease is positive (+).
 - Assumptions:
 - Tests are correct 99% (test are positive with 99% probability if the person is sick, and negative with 99% if the person is not sick)
 - Disease is rare, happening only 1 in every 10000 people
 - If a person is tested and result is positive which is the probability of being sick?

$$P(\text{sick} | +) = P(+ | \text{sick}) \cdot P(\text{sick}) / P(+)$$

$$P(+ | \text{sick}) = 0.99 \quad P(\text{sick}) = 1/10000$$

$$P(+) = P(+ | \text{sick}) \cdot P(\text{sick}) + P(+ | \text{not sick}) \cdot P(\text{not sick})$$

$$= 0.99 \times 1/10000 + 0.01 \times 9999/10000 = 0.010098$$

$$P(\text{sick} | +) = 0.99 \times 1/10000 / 0.010098 = 0.009804 \approx 1\%$$

<https://math.hmc.edu/funfacts/medical-tests-and-bayes-theorem/>

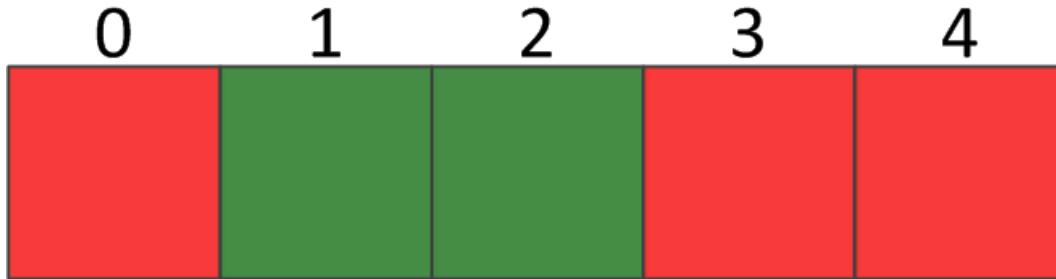
Bayesian Filter

- Application of the **Bayes' Rule** when:
 - X_t is the state vector at time t .
 - U_t is the control vector used to drive from state X_{t-1} to X_t .
 - Z_t is the observation of the state at time t
- At an instant t an estimation of the current state can be achieved by application of Bayes' Rule:

$$P(X_t|Z_t, U_t) = \frac{P(Z_t|X_t)P(X_t|Z_{t-1}, U_t)}{P(Z_t|Z_{t-1}, U_t)}$$

Grid World

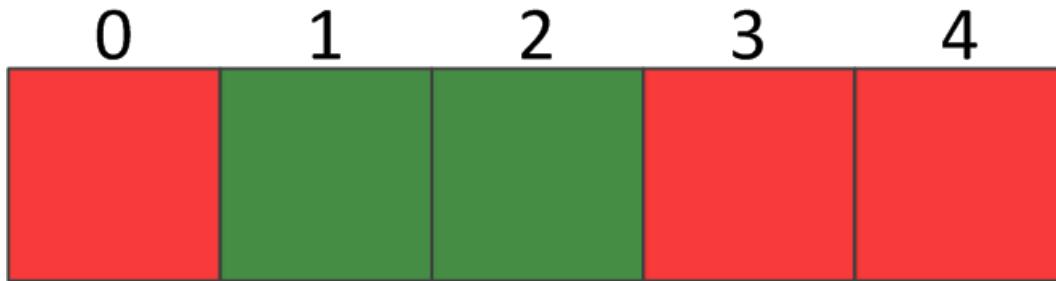
- Consider a grid world:



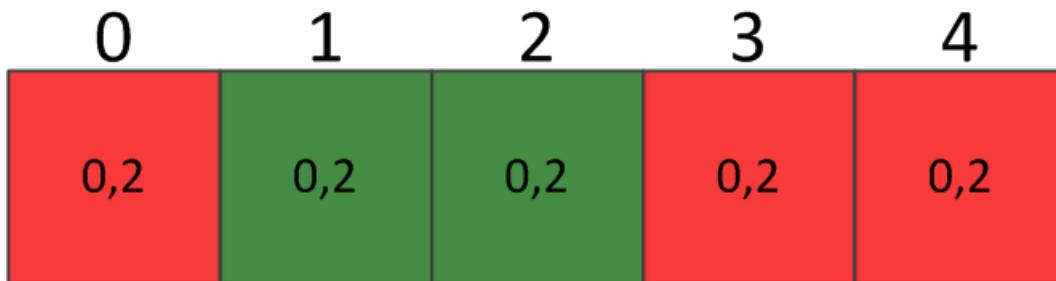
- Where is the robot?

Grid World

- Consider a grid world:



- Position probability distribution:



Grid World

- The robot has a color sensor
- Sensor model

G_M : green measure; R_M : red measure

G_C : green cell; R_C : red cell

$$P(G_M|G_C) = 0.6; P(R_M|G_C) = 1 - P(G_M|G_C) = 0.4$$

$$P(G_M|R_C) = 0.2; P(R_M|R_C) = 1 - P(G_M|R_C) = 0.8$$

- The robot measures **green**
- Which is the new position estimate?

Grid World

- Multiply cell values by $P(G_M|G_C)$ and $P(G_M|R_C)$



- Normalize



- We have just applied Bayes' Rule!

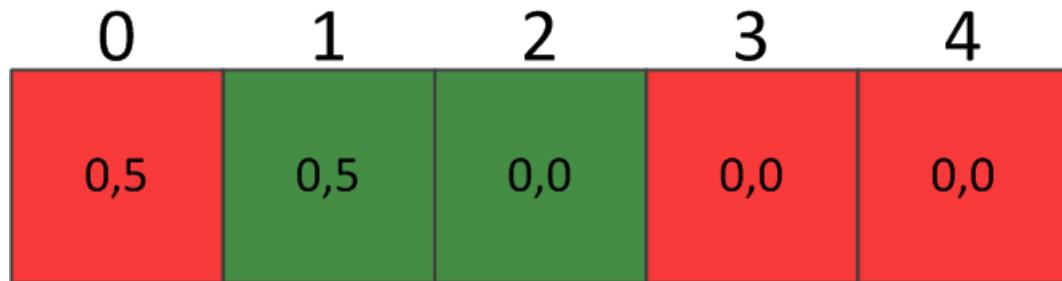
- Bayes' Rule

$$P(X_p|M) = \frac{P(M|X_p) * P(X_p)}{P(M)}$$

- $P(X_p)$ is the estimate before measurement integ.
- $P(X_p|M)$ is the posterior estimate
- $P(M)$ does not depend on X_p , so it can be considered as a constant that performs normalization
- **Measurement integration** is performed using the **product** of sensor model and previous estimate

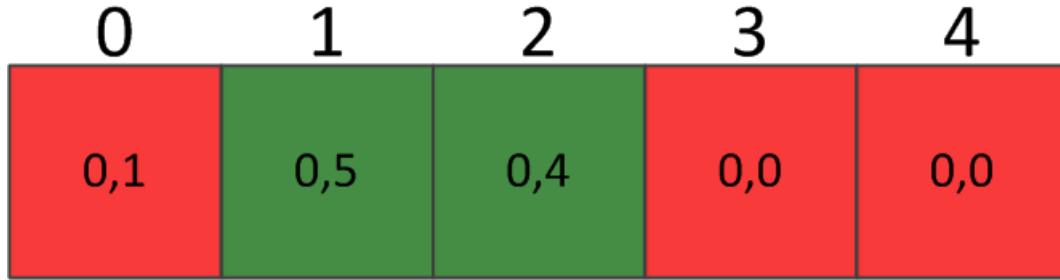
Grid World - Moving

- If the robot (tries) to move
- Action model
 - A_R : Move Right action;
 - $P(X+1| A_R, X) = 0.8; P(X| A_R, X) = 0.2$
- Initial belief



Grid World - Moving

- $P(1) = P(1 | A_R, 0)^*P(0) + P(1 | A_R, 1)^*P(1)$
- Predicted belief



- When the **robot moves** belief is updated through **convolution**

Sensor Fusion

- Continuous environments
 - Measurement Integration (Product)

$$bel(X)_t = \eta \ p(Z_t|X) \ \hat{bel}(X)_t$$

- Motion update (Convolution)

$$\hat{bel}(X)_t = \int p(X|U_t, X') bel(X')_{t-1} dX'$$

Kalman Filter

- Assumptions
 - **Linear model** (transition, action and sensor)
 - **Every estimate is a gaussian**
 - Can be characterized by **mean** and **variance**
 - **Noise is gaussian** ($\text{mean}=0$)
- Integration of measures over time
- **Markovian assumption**
 - The next estimate only depends on the previous estimate
- Considers **action model** and **physics/observation model**

Kalman Filter

- **Product of gaussians distributions
(measurement integration)**

$$\mu_P = \frac{\mu_1 \cdot \sigma_2^2 + \mu_2 \cdot \sigma_1^2}{\sigma_1^2 + \sigma_2^2} \quad \sigma_P^2 = \frac{\sigma_1^2 \cdot \sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

- **Convolution of gaussians distributions
(motion forecast)**

$$\mu_C = \mu_1 + \mu_2 \quad \sigma_C^2 = \sigma_1^2 + \sigma_2^2$$

Kalman Filter – Motion Model

$$\hat{X}_t = F_t \hat{X}_{t-1} + B_t U_t + \omega_t$$

- \hat{X}_t is the **estimated state**
- F_t is the **state transition model**
- B_t is the **control-input model**
- U_t is the **control vector**
- w_t is the **process noise** with covariance

$$Q_t : \omega_t \sim N(0, Q_t)$$

$$\hat{Z}_t = H_t X_t + v_t$$

- \hat{Z}_t is the **measurement** taken at time t
- H_t is the **observation model** of the state/event
- v_t is the **observation noise** with covariance:

$$R_t : v_t \sim N(0, R_t)$$

Kalman Filter – Implementation

- The filter state is represented by two variables:
 - X_t is the **estimate of the state** at time t
 - P_t is the **measure of estimated accuracy** of the process
- The filter works in **two steps**:

Forecast

$$\bar{X}_t = F_t X_{t-1} + B_t U_t$$

$$\bar{P}_t = F_t P_{t-1} F_t^T + Q_t$$

Measurement integration

$$K_t = \frac{\bar{P}_t H_t^T}{H_t \bar{P}_t H_t^T + R_t}$$

$$X_t = \bar{X}_t + K_t (Z_t - H_t \bar{X}_t)$$

$$P_t = (I - K_t H_t) \bar{P}_t$$

Particle Filter (Monte Carlo)

- Integration of measures over time
- Is **non-parametric** and thus can cope with several types of distributions, rather than just Gaussian
- The samples of the state are called **particles**
- **Each particle is a concrete instantiation of the state at time t**
- This filter also works in **two main steps**, applied to every single particle:
 - **Evolution** of the current state, **weighing** and **creation** of a dataset
 - **Resampling** of the dataset for the next evaluation

Particle Filter (Monte Carlo)

- Consider M the number of particles used and X_t the particle set at time t

Evolution and weighing

for $m = 0$ to M

$$x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$$

$$w_t^{[m]} = p(z_t | x_t^{[m]})$$

$$\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$$

endfor

Resampling

for $m = 0$ to M

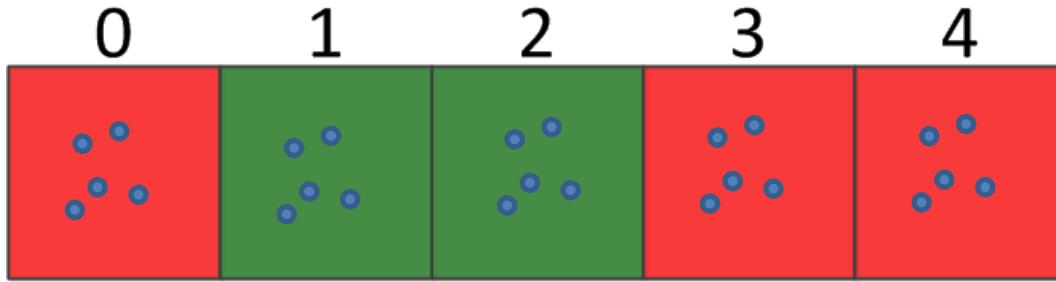
draw i with prob $\propto w_t^{[i]}$

add $x_t^{[i]}$ to X_t

endfor

Grid World - Weighing

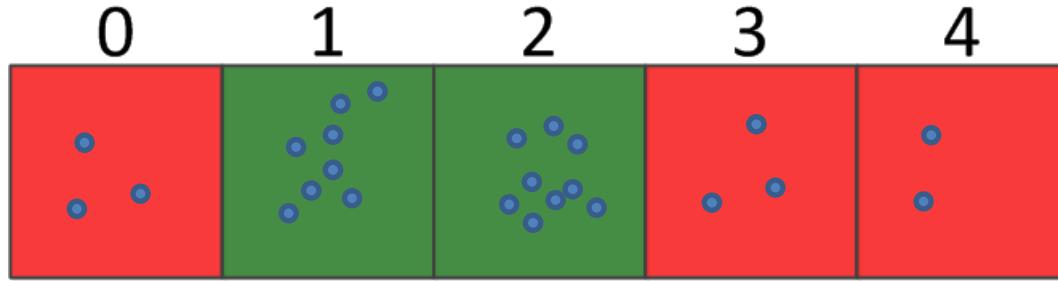
- Initial particle distribution:



- If green is seen, particles in green cells have higher weights
- They will have higher chances of getting into the new particle set

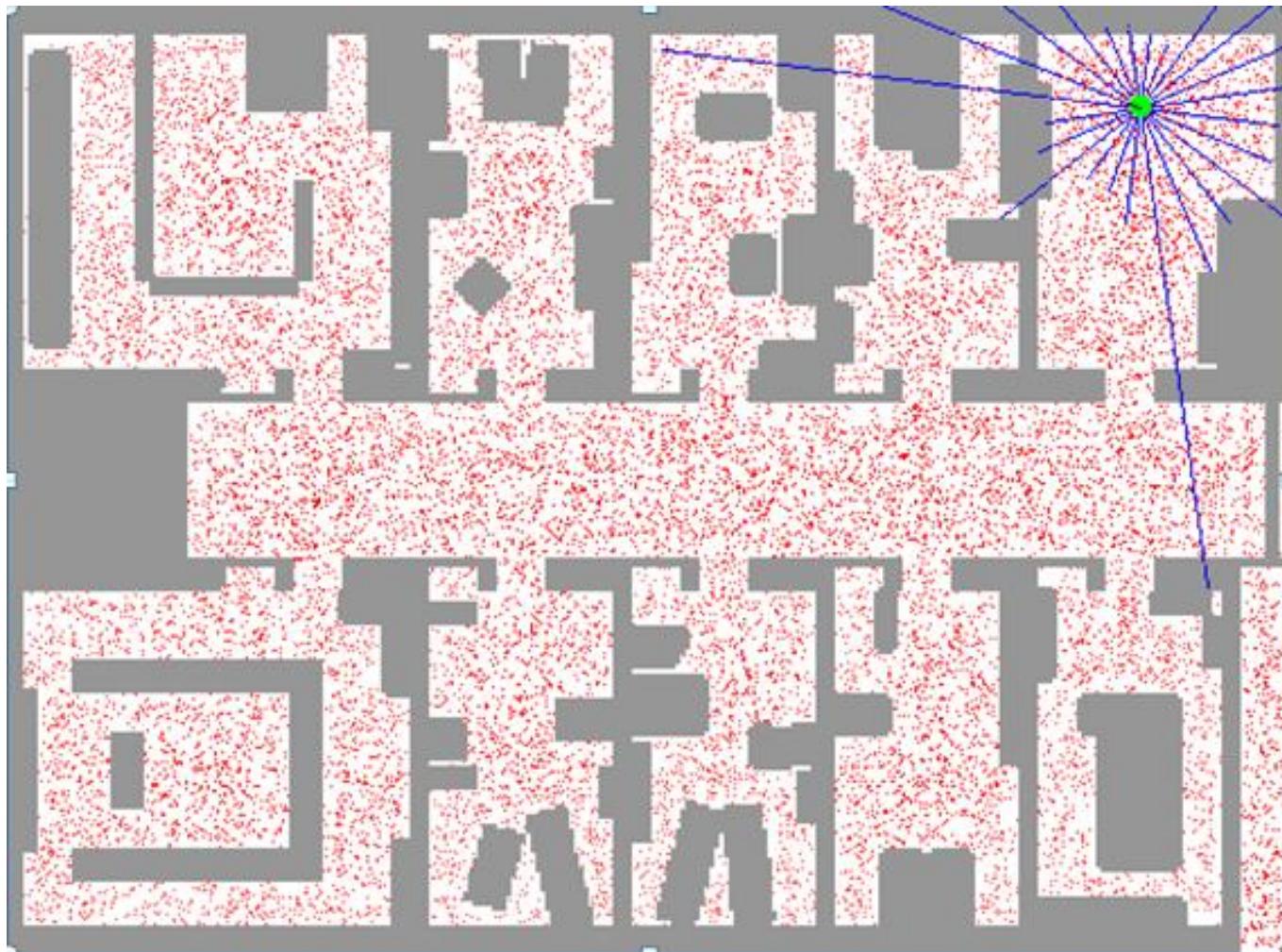
Grid World - Resampling

- After green is seen:

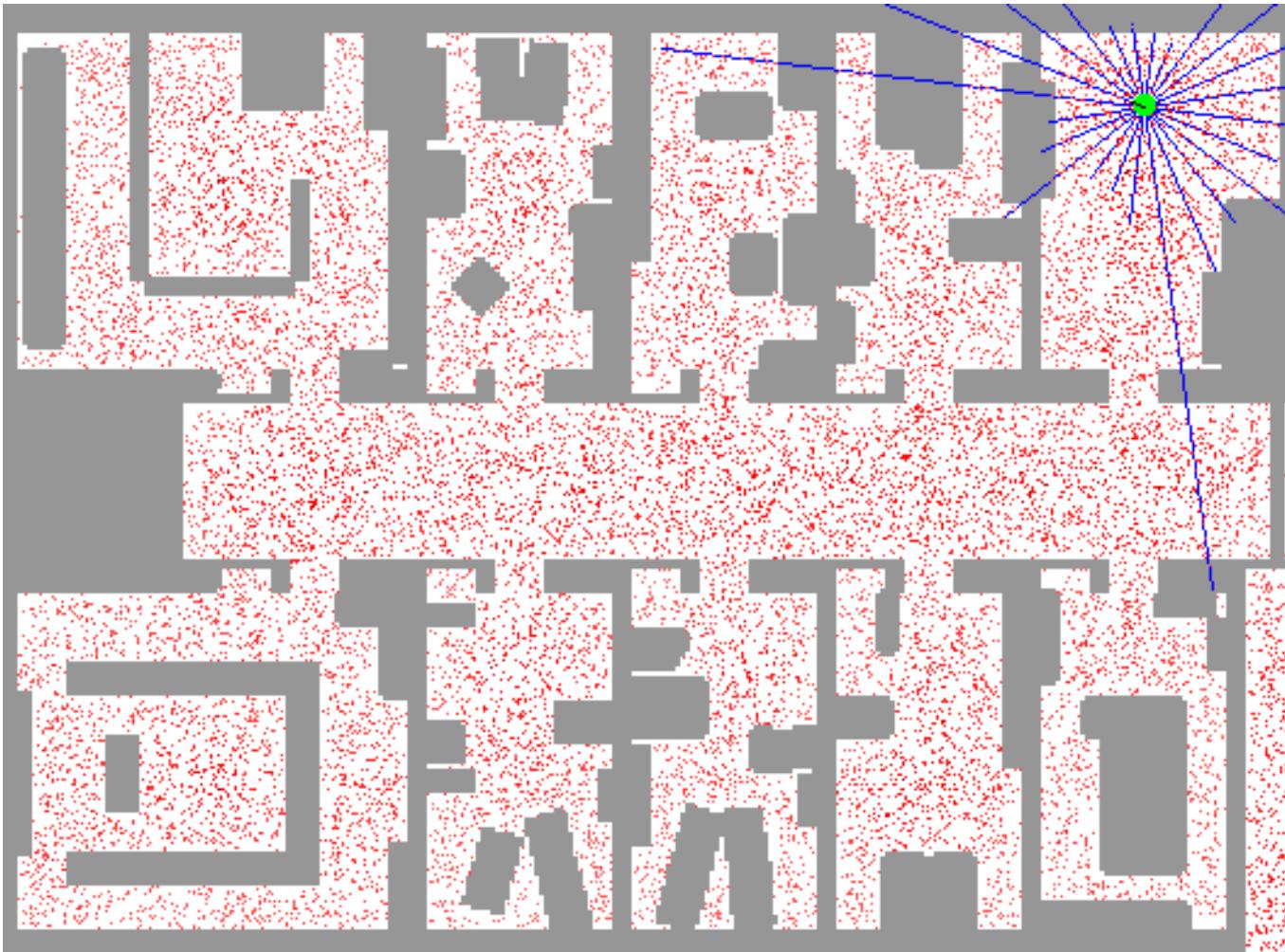


- The motion model may be applied directly to each particle

Particle Filter (Monte Carlo)



Particle Filter (Monte Carlo)



<https://rse-lab.cs.washington.edu/projects/mcl/animations/global-floor.gif>

References

- Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. The MIT Press, 2005.
- Sebastian Thrun, Artificial Intelligence for Robotics, Udacity, www.udacity.com
- Hugh Durrant-Whyte and Tom Henderson. Multisensor data fusion. In Siciliano and Khatib, editors. Springer Handbook of Robotics. 2nd ed. Springer, 2016, pages 867–890.



Robótica Móvel e Inteligente

Path Planning and Obstacle Avoidance

Artur Pereira <artur@ua.pt>

DETI / Universidade de Aveiro

Outline

- ① Navigation
- ② Path planning concepts
- ③ Environment representation
- ④ Search algorithms
- ⑤ Potential field path planning
- ⑥ Obstacle avoidance
- ⑦ Bibliography

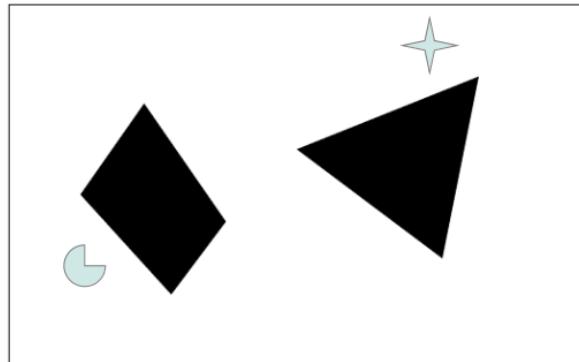
Navigation

Definition

- “Given partial knowledge about its environment and a goal position or series of positions, **navigation** encompasses the ability of the robot to act based on its knowledge and sensor values so as to reach its goal positions as efficiently and as reliably as possible.”

[Siegwart, <http://www.cs.cmu.edu/~rasc/Download/AMRobots6.pdf>]

- In mobile robotics the knowledge about the **environment** and **situation** is usually only **partially** and **uncertain**



Navigation

Issues in navigation

- Where am I?
 - localization
- Where have I been?
 - mapping
- Where should I go?
 - decision
- What's the best way to get there?**
 - Path planning**
- How do I get there?
 - Path following and obstacle avoidance (Motion)

Path planning

Definition

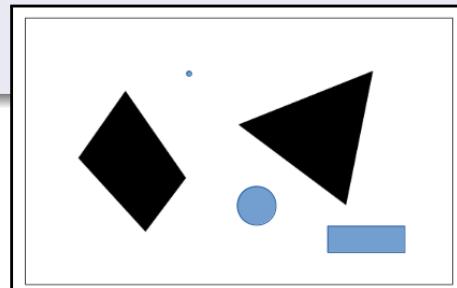
- Path planning – the task of computing a path for the robot such that it can reach the desired goal without colliding with known obstacles
 - Optimal paths can be hard to compute, specially for robots that can not move in arbitrary directions (i.e. non-holonomic robots)
-
- Path and trajectory are often used with the same meaning
 - But one can distinguish them
 - Path – only geometric considerations
 - a way from a start position/pose/configuration to a goal one
 - Trajectory – includes geometric and time considerations
 - the dynamics is also considered
 - Another term is motion, applied to mobile robots or manipulators
 - considers other constraints, like collision avoidance (of dynamic obstacles)

[2] Peter Corke, <https://robotacademy.net.au/masterclass/paths-and-trajectories/>

Path planning

Some concepts

- Configuration space (C-Space) – set of possible valid configurations (poses) of the robot
 - Defines the search space and the set of allowable paths
- Free space (F-Space) – set of valid configurations that do not intercept obstacles in the environment
 - Depends on the robot shape
- Cases to be considered:
 - Point robot .
 - Symmetric robot ●
 - Non-symmetric robot ■



Path planning

Some assumptions

- Assumption 1:
 - Often, path planning methods assume that the robot is symmetric and holonomic, and treat it simply as a point
 - If the robot is treated as a point, the obstacles must be “inflated” in order to compensate the robot radius
 - This approach greatly simplifies path planning
- Assumption 2:
 - there exists a good enough representation (map) of the environment that can be used to compute a path
- Assumption 3:
 - there exists a good enough estimation of the robot's pose

Path planning

Environment representation

- There are different ways to represent the environment
- Two common ones are **topological maps** and **geometric maps**
- geometric maps may represent a continuous geometric description
- Topological representations are formally **graphs** where:
 - **nodes** denote areas/points/sections in the environment
 - **arcs** denote adjacency between nodes



Path planning

Example of metric and topological maps

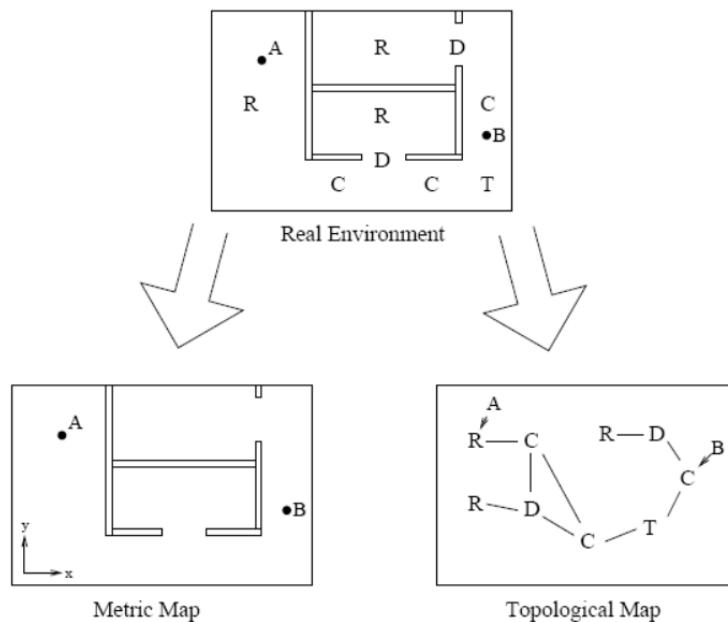
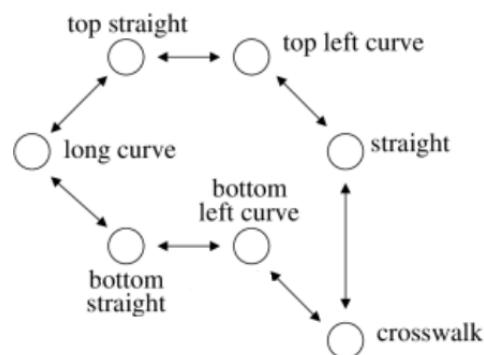
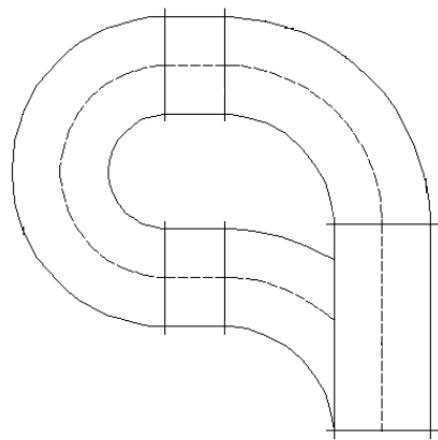


Figure from Meyer, "Map-based navigation in mobile robotics". 2003

Path planning

Another example of metric and topological maps



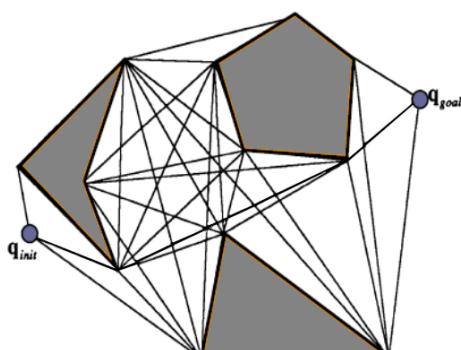
Environment representation

Approaches

- Path-planning algorithms often are only applicable to **discrete maps**
 - Transforming the possibly continuous environmental map model into a discrete representation suitable for the chosen path-planning algorithm is a must
 - Path planners differ in how they perform this discrete decomposition
-
- General strategies for decomposition:
 - **Road map** – identify routes within the free space
 - Transforming the free space into a network of 1-D curves or lines, called road-map
 - **Cell decomposition** – divide space into cells that are discriminate between free and occupied cells
 - Transforming the free space into a connectivity graph of free cells, based on adjacency
 - **Potential field** – apply a mathematical function over the space
 - Transforming the free space into a field, or gradient, that directs the robot to the goal position

Path planning approaches

Visibility graph

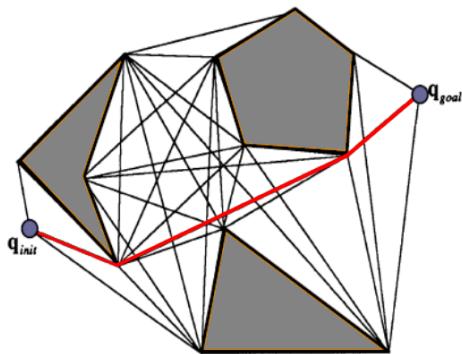


- Obstacles are treated as polygons
 - for every obstacle, a polygon including it can be defined
- A graph is defined where:
 - q_{init} , q_{goal} , and all polygon vertices are the nodes
 - for every pair of nodes which can be connected by a line segment, not passing through an obstacle, there is an edge
 - such pair of nodes can “see” each other)

- With the detected nodes and edges, a connectivity graph (visibility graph) is then generated
 - every edge can be labelled with some cost function value (for example, the Euclidian distance)

Path planning approaches

Visibility graph (2)

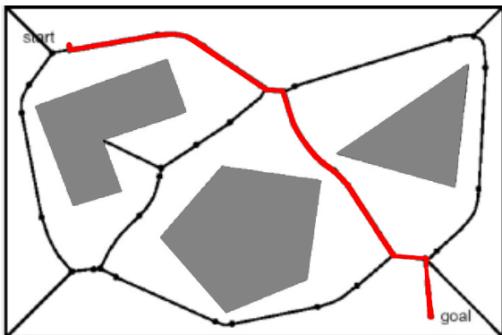


- A graph search algorithm can be used to find the shortest path along the “roads” defined by the visibility graph
- Efficient method for sparse environments

- **Problem** – may cause the robot to move too close to obstacles
 - little margin for errors in motion
- **Solution** – grow obstacles (even more), giving more clear space between robot and obstacles

Path planning approaches

Voronoi diagrams



- For each point in free space its distance to the nearest obstacle is computed
- Then, the set of points equidistant from the nearest two or more obstacle boundaries are extracted

- The Voronoi diagram is obtained by linking these points, from where the shortest path can be computed
- The result is a path of maximum distance from obstacles
 - usually far from optimal, in the sense of total path length

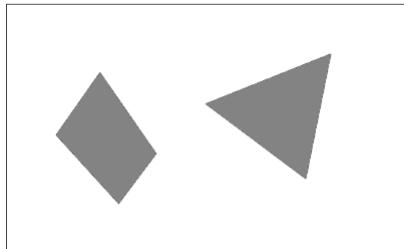
Path planning approaches

Cell decomposition

- Divide space into simple, connected regions called **cells**
 - cells can be either free, occupied or partially occupied
- Discretize the space by constructing an **adjacency graph** of the free cells
- Adjacency graph
 - **Nodes** – free cells
 - **Edges** – there is an edge between every pair of nodes whose corresponding cells are adjacent
- Locate “goal” and “start” cells and search for the **shortest path** in the adjacency graph that join them
- Typically paths are assumed to pass through the **mid-points** of the cells

Path planning approaches

Cell decomposition (2)



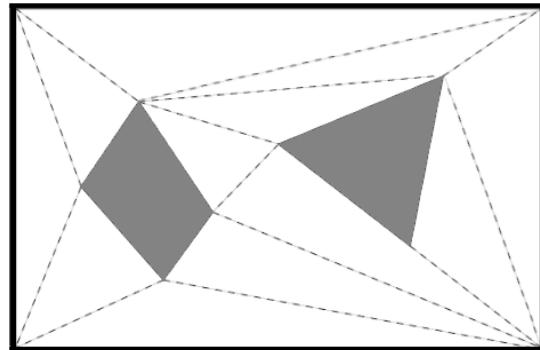
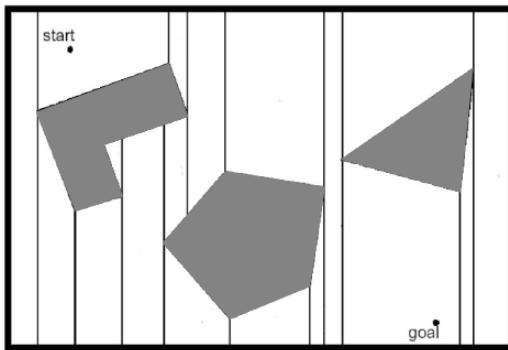
- Given an environment, how to decompose it?

- Two possible cell decomposition methods
 - **Exact cell decomposition** and **approximate cell decomposition**
- **Exact cell decomposition**
 - Free cells correspond exactly to free space
 - There is no partially occupied cells
- **Approximate cell decomposition**
 - Some free space is included in partially occupied cells
 - the partially occupied cells are considered as occupied
- Cells can have fixed-size or variable-size

Path planning approaches

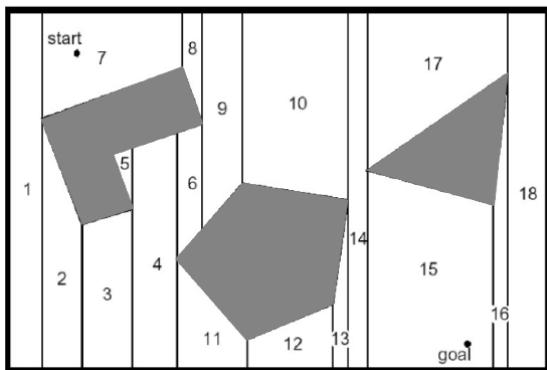
Exact cell decomposition

- The free space F is divided into a set of non-overlapping **convex** cells whose union is **exactly F**
- Examples of convex shapes: **trapezoids, triangles**
- The basic abstraction behind this method is that the position of the robot within each cell does not matter, only the ability to travel to another free cell

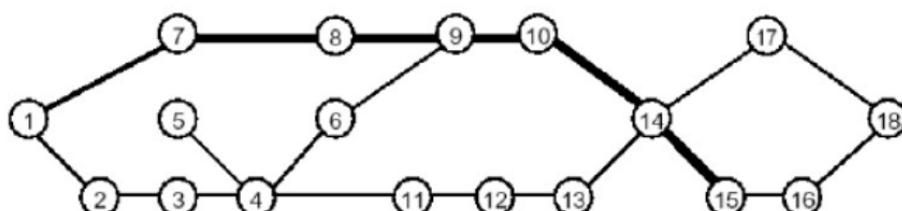


Path planning approaches

Exact cell decomposition (2)

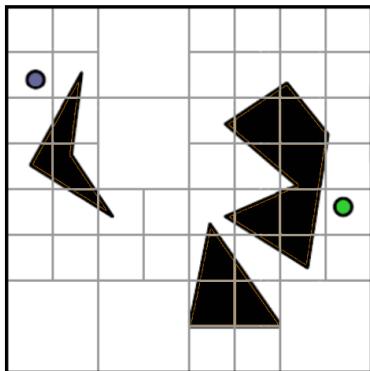


- A **connectivity graph** can be constructed, where:
 - nodes represent the free cells
 - every adjacent pair of cells is connected by an edge
- Result can be complex if the world is complex
 - Good for sparse environments



Path planning approaches

Approximate cell decomposition



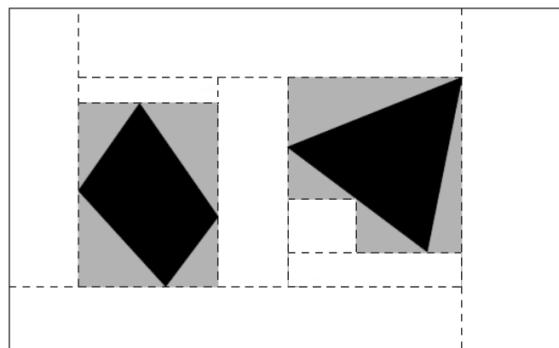
- Free space F is represented by a set of non-overlapping cells whose union is contained in F
- Cells usually have simple, regular shapes, like rectangles, squares, hexagons

- Two approaches:
 - Variable-size cell decomposition
 - Fixed-size cell decomposition

Path planning approaches

Variable-size cell decomposition

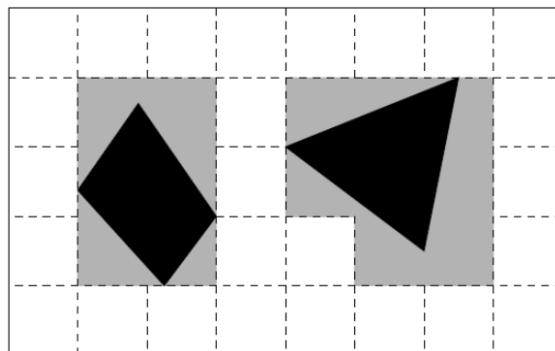
- Decomposing using variable-size cells
 - with a rectangular shape
- Grey areas are free space considered as occupied



Path planning approaches

Fixed-size cell decomposition

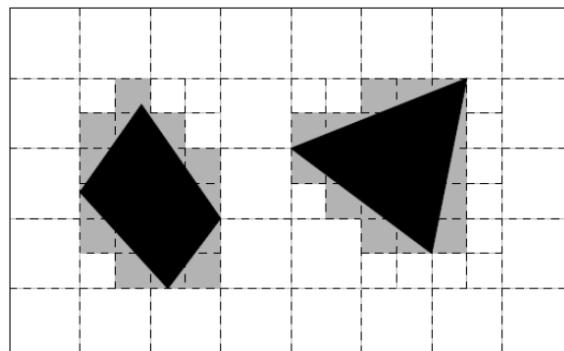
- Decomposing using fixed-size cells
 - with a square shape
- Grey areas are free space considered as occupied
- Cell size is not dependent on the obstacle size in the environment
 - narrow passage ways can be lost
- Low computational complexity of path planning



Path planning approaches

Quad-tree cell decomposition

- Decomposing using variable-size cells
 - with a square shape
- Partially occupied cells are subdivided until a given granularity
- At each level of resolution only the cells whose interiors lie entirely in the free space are used to construct the connectivity graph
- Efficient representation, **adapted to the complexity of the environment**
 - sparse environments contain fewer cells thus consuming less memory



Search algorithms

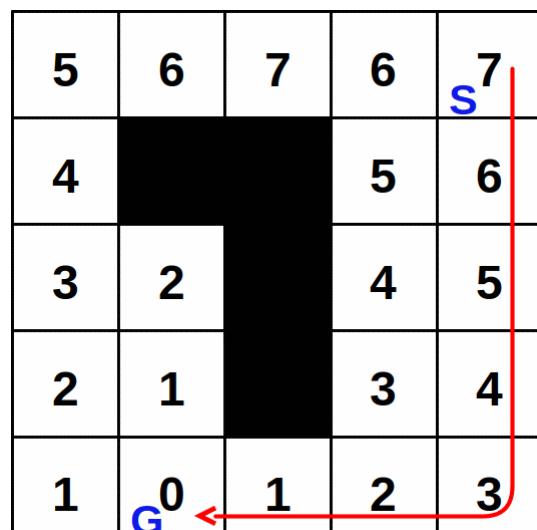
Methods

- Once a graph is obtained, finding the shortest path between start node and goal node can be done using **graph search algorithms**
- Many graph search algorithms require visiting each node in the graph to determine the shortest path
 - Computationally tractable for sparsely connected graphs
 - Computationally expensive for highly connected graphs (e.g., regular grid)
- Covered methods:
 - Wavefront expansion
 - Dijkstra's algorithm
 - A* algorithm

Search algorithms

Wavefront expansion

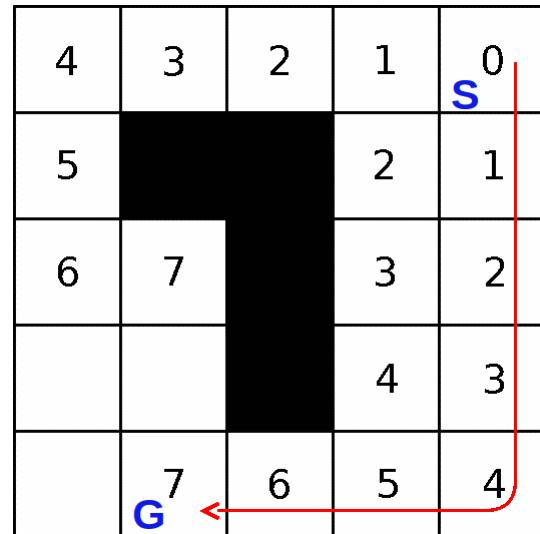
- Wavefront expansion (aka NF1 or grassfire), useful to find paths in fixed-size cell arrays
- Starting at the goal, **mark in each adjacent cell its distance to the goal** (using Manhattan distance)
- This process continues until the cell corresponding to the **start position is reached**
- The planner calculate a path to reach the goal by **linking together cells that are adjacent and closer to the goal**



Search algorithms

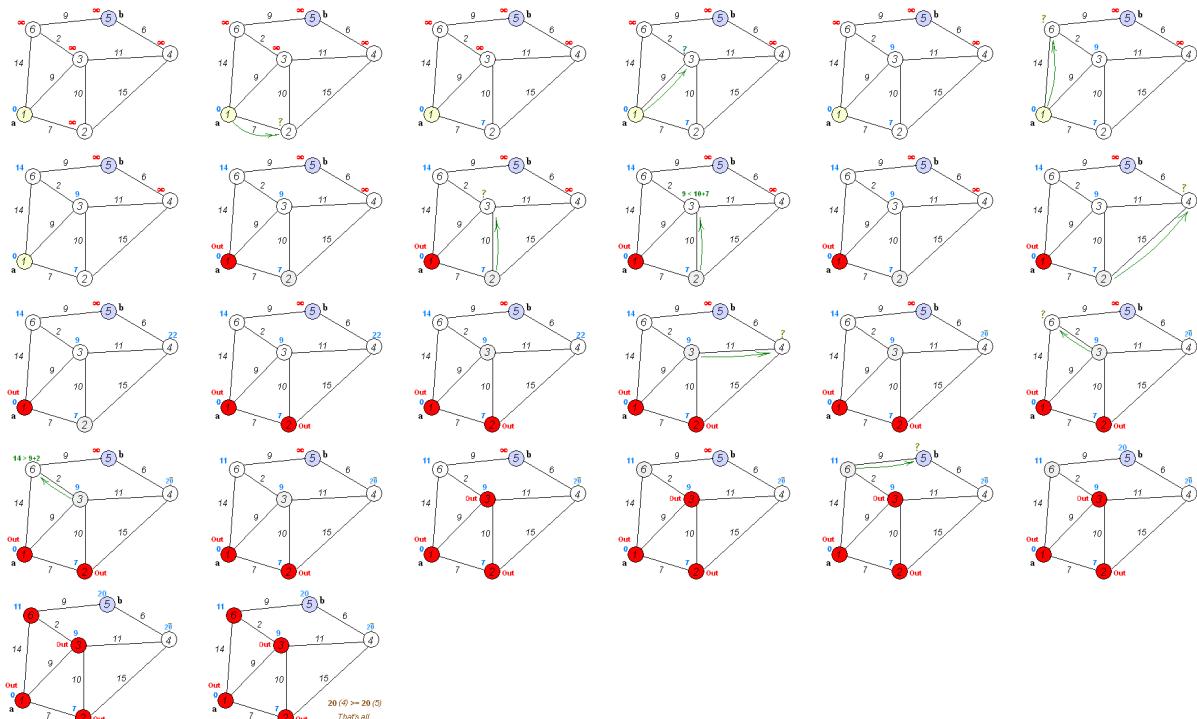
Dijkstra's algorithm – cell environment

- Beginning at the start node, the algorithm marks all adjacent neighbors with the **cost to get there**
- It then **proceeds to the node with the lowest cost** marking all of its adjacent nodes with the lowest cost to reach them
- Once all adjacent neighbors of a node have been marked, the algorithm **proceeds to the node with the next lowest cost not visited yet**
- Once the algorithm visits the goal node, it terminates
- The **path to goal** may be obtained starting from the goal node and following the **edges pointing towards the lowest node cost**



Search algorithms

Dijkstra's algorithm – graph example



Taken from [https://en.wikipedia.org/wiki/Dijkstra's_algorithm](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)

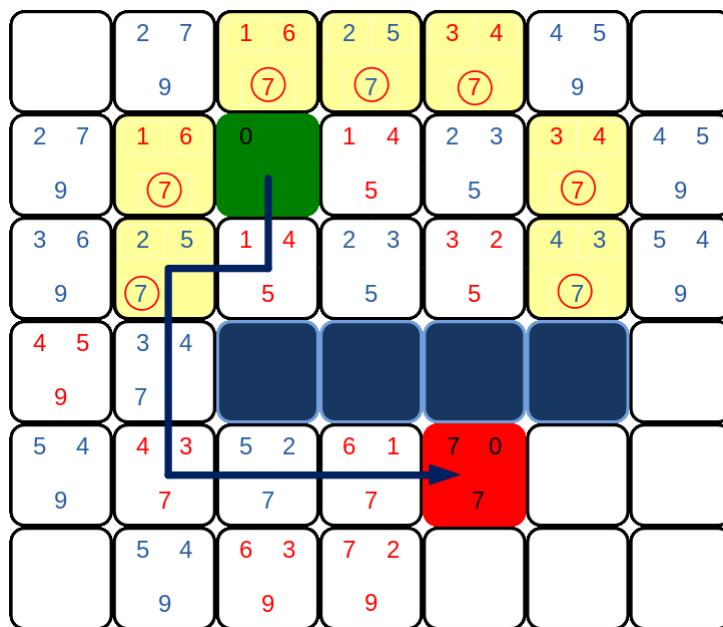
Search algorithms

A* search algorithm

- The idea is to use a **cost function to rank** the choices, choose the best one first, and try it
- Cost function:
 - $f^*(n) = g(n) + h^*(n)$ – ('*' means they are estimates)
 - where:
 - $g(n)$ is the cost of going from the start to node n
 - $h^*(n)$ is an estimated cost of going from node n to the goal
- h^* is a “heuristic function” – (a way of **guessing** the cost of going from node n to **goal**)
 - the robot can’t “see” the path between node n and the goal
 - $h^*(n)$ should never be greater than $h(n)$: $h^*(n) \leq h(n)$**
 - Must always underestimate remaining cost to reach goal**
 - The Euclidian (straight line) distance is often a good choice

Search algorithms

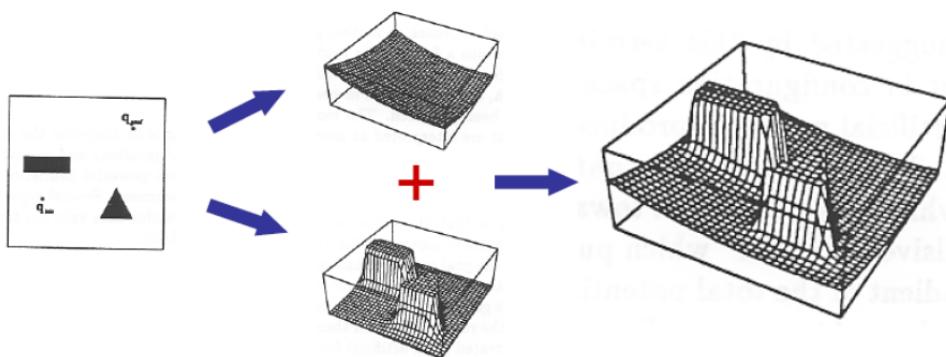
A* search algorithm (2)



Potential field path planning

Main idea

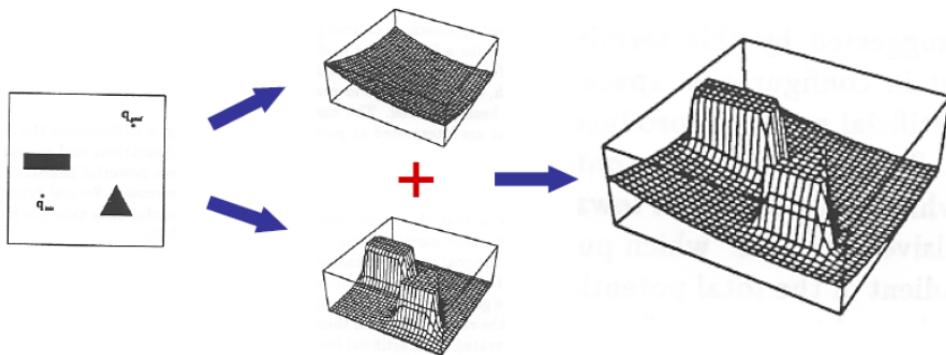
- Think the robot as a **particle** in a **potential field**
- Define a potential function over the free space that has a **global minimum at the goal**
- Define high potentials for the obstacles
- The robot follow the steepest descent of the potential function



Potential field path planning

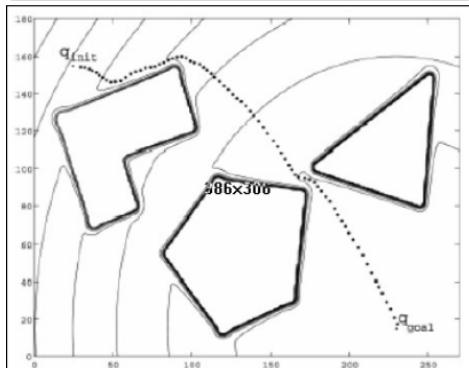
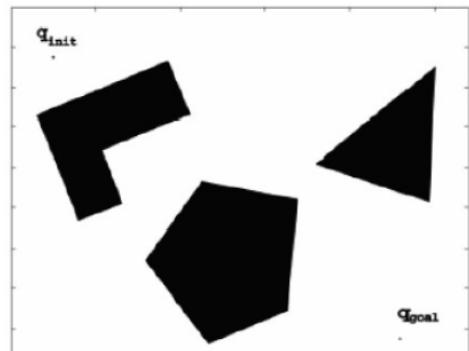
Rolling down the hill

- The **goal** location generates an **attractive potential**, pulling the robot towards the goal
- The **obstacles** generate a **repulsive potential**, pushing the robot far away from the obstacles
- The negative gradient of the total potential is treated as an artificial force applied to the robot
- Generated robot movement is similar to a ball rolling down the hill

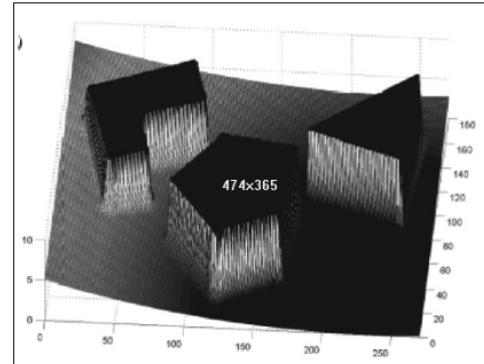


Potential field path planning

Problems



- Often leads to oscillating motion
- Trapped in local minima in the potential field
- Parameter tuning problems



Navigation

Issues in navigation

- Where am I?
 - localization
- Where have I been?
 - mapping
- Where should I go?
 - decision
- What's the best way to get there?
 - Path planning
- **How do I get there?**
 - Path following and **obstacle avoidance** (Motion)

Obstacle avoidance

Purpose

- In general, the environment is **not fully modeled**
 - Some obstacles (chairs, for example) may not be represented
 - The environment might also **change dynamically**
 - for example, people moving around
 - In such cases, to navigate, the robot may need to modify the planned path
 - reacting, re-planning
-
- The purpose of the obstacle avoidance algorithms is to avoid collisions with obstacles, while pursuing the goal/plan
 - Obstacle avoidance relies on
 - Information about the goal (position, plan)
 - current pose (on the map)
 - Recent sensory information (a local map)

Obstacle avoidance

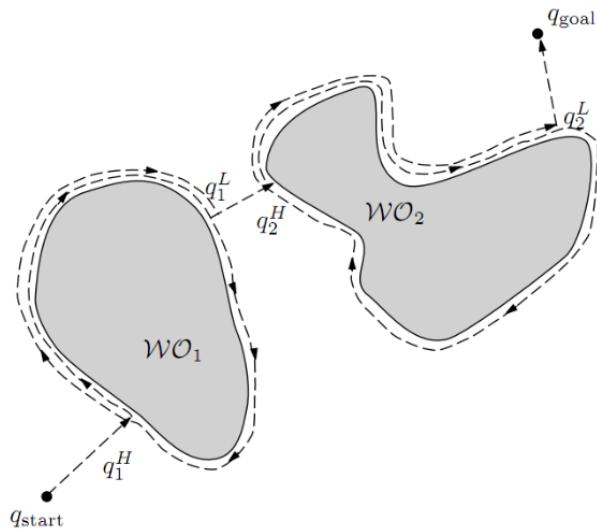
Methods

- Efficient obstacle avoidance should be optimal with respect to
 - The overall goal
 - The actual speed and kinematics of the robot
 - The on-board sensors
 - The current and future risk of collision
- Covered methods:
 - Bug1
 - Bug2
 - Vector field histogram

Obstacle avoidance

Bug1 algorithm

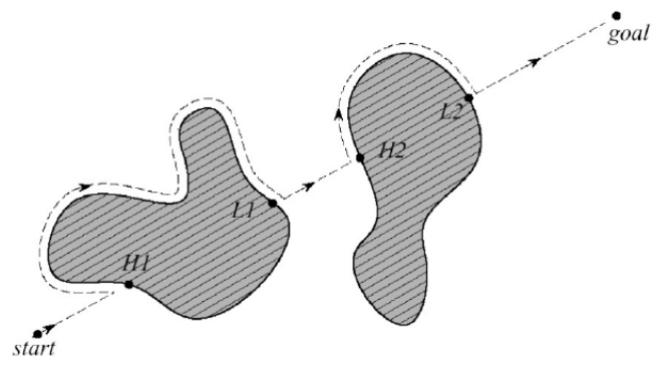
- Algorithm:
 - Go in direction to the goal until reach it or hit an obstacle
 - If goal reached, finish
 - Do a full tour around the obstacle, storing the closest point to the goal
 - Go to that point
 - Repeat
- Comment:
 - Inefficient but does the job



Obstacle avoidance

Bug2 algorithm

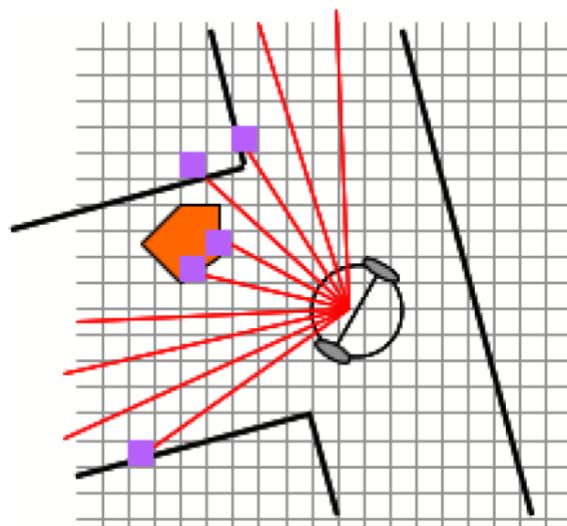
- Algorithm:
 - Go in direction to the goal until reach it or hit an obstacle
 - If goal reached, finish
 - Contour the obstacle, from left or right, until reach the line between start and goal
 - Repeat, keeping the same side (left or right) as before
- Comment:
 - More efficient but can fail



Obstacle avoidance

Vector field histogram (VFH)

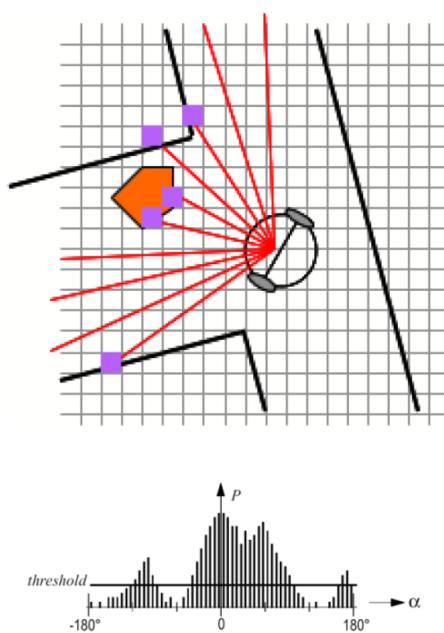
- Creates a local map of the environment around the robot
- Environment is represented as an **occupancy grid** (2D Cartesian grid called the **histogram grid**)
- Each **grid cell (*i, j*)** value holds the **confidence** that there is an **obstacle** at that location
- The grid is updated by relatively recent sensor data



Obstacle avoidance

Vector field histogram (VFH) – 2

- Information in the histogram grid is converted to a simpler representation, called the **polar histogram**
- The **polar histogram** retains the statistical information but reduces the amount of data that needs to be handled in real-time
- A **threshold** transforms the polar histogram in a **binary diagram** with passable regions
- A sector with low obstacle density (below threshold) is a candidate to travel away from obstacle
- The one chosen depends on the target point



Bibliography

- “Introduction to Autonomous Mobile Robots”, Second Edition, Roland Siegwart et al., MIT Press, 2011
- “Principles of Robot Motion: Theory, Algorithms, and Implementations”, Howie Choset et al., MIT Press, Boston, 2005
- “Artificial Intelligence: A Modern Approach”, 3rd edition, Russel and Norvig, Pearson, 2009
- “Introduction to Autonomous Mobile Robots”, R. Siegwart, I. Nourbakhsh, D. Scaramuzza
- “The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots”, J. Borenstein and Y. Koren
- “VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots”, I. Ulrich and J. Borenstein



Robótica Móvel e Inteligente

Mobile Robot Localization

Artur Pereira <artur@ua.pt>

DETI / Universidade de Aveiro

Outline

- ① Localization
- ② Markov localization
- ③ Kalman filter localization
- ④ Grid localization
- ⑤ Monte Carlo localization
- ⑥ Localization in CAMBADA
- ⑦ Bibliography

Navigation

Questions and topics

- Where am I?
 - localization
- Where have I been?
 - mapping
- Where should I go?
 - decision
- What's the best way to get there?
 - Path planning
- How do I get there?
 - Path following and obstacle avoidance (Motion)

Localization

Introduction

- How to determine the pose of a mobile robot relative to a given map of the environment?
 - Using sensors – beacons for triangulation, distance sensors, compass, odometry, inertial sensors, motion orders, ...
 - Using an appropriate, accurate enough map of the environment
- Difficulties:
 - In general, the pose cannot be sensed directly
 - it has to be inferred from data
 - A single sensor measurement is usually insufficient to determine the pose
 - robot has to integrate data over time and/or from different sources
 - The exact pose of a robot can not, in general, be determined
 - pose must be given by a probability distribution
 - the robot only knows the probability of being at a given pose

Localization

The localization problem

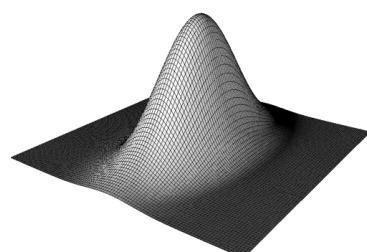
- **Goal:**
 - Localize the robot in a known map of the environment
- **Inputs:**
 - Map of the environment
 - Perceptions and actions of robot
- **Output:**
 - Estimation of pose relative to the map
 - In 2D spaces, this is expressed as the triple (x, y, θ) , where (x, y) is the robot's position and θ its heading
 - In 3D spaces, 6 coordinates may be required, 3 for position and 3 for heading (roll, pitch and yaw)
- There are different approaches to tackle this problem

Localization

Markov Localization

- Probabilistic state estimation is applied to the localization problem through Bayes filters
 - It is called **Markov Localization**
- Markov assumption:
 - **Past and future are independent**, if one knows the current state
 - Sensor measures do not depend on previous measures, if position is known
- In localization the state is the **robot's pose**

- **Pose** is given by a belief function
 - it is the probability distribution of the estimated pose of the robot for every possible pose



Localization

Markov Localization

Algorithm Markov_localization($bel(x_{t-1})$, u_t , z_t , m):

for all x_t do

$$\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}, m) bel(x_{t-1}) dx_{t-1}$$

$$bel(x_t) = \eta p(z_t | x_t, m) \overline{bel}(x_t)$$

endfor

return $bel(x_t)$

- $bel(x_{t-1})$ is the belief at time $t-1$; u_t the actions at time interval $[t-1, t)$; z_t the measurements at time t ; and m the map of the environment
- $\overline{bel}(x_t)$ is the belief at time t based only on the actions
- $bel(x_t)$ is the belief at time t based on actions and measurements
- η is a normalization factor (from Bayes filter)

Localization

Markov Localization

- Splitting actuation and measurement
 - Prediction phase – update previous estimate only based on actuation
 - Correction phase – correct prediction based on measurements

Algorithm Markov_localization($bel(x_{t-1})$, u_t , z_t , m):

for all x_t do

$$\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}, m) bel(x_{t-1}) dx_{t-1}$$

endfor

for all x_t do

$$bel(x_t) = \eta p(z_t | x_t, m) \overline{bel}(x_t)$$

endfor

return $bel(x_t)$

Localization

Markov Localization

- Transposing to the discrete domain

Algorithm Markov_localization($bel(x_{t-1})$, u_t , z_t , m):

for all x_t do

$$\overline{bel}(x_t) = \sum p(x_t | u_t, x_{t-1}, m) bel(x_{t-1})$$

endfor

for all x_t do

$$bel(x_t) = \eta p(z_t | x_t, m) \overline{bel}(x_t)$$

endfor

return $bel(x_t)$

Localization

Markov Localization – prediction phase

$$\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}, m) bel(x_{t-1}) dx_{t-1}$$

$$\overline{bel}(x_t) = \sum p(x_t | u_t, x_{t-1}, m) bel(x_{t-1})$$

- Incorporates only motion model
- Input:
 - Previous belief distribution: $bel(x_{t-1})$
 - Action taken: u_t
- How does u_t change bel ?
 - Every possible value for x_{t-1} has to be considered on its probability of transition to x_t

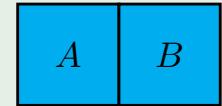
Localization

Markov Localization – prediction example

$$\overline{\text{bel}}(x_t) = \sum p(x_t | u_t, x_{t-1}, m) \text{bel}(x_{t-1})$$

- Consider a world with 2 cells, A and B
- Assume the following motion model on action *left*

$$\begin{aligned} P(A | \text{left}, A) &= 0.99 & P(B | \text{left}, A) &= 0.01 \\ P(A | \text{left}, B) &= 0.12 & P(B | \text{left}, B) &= 0.88 \end{aligned}$$



- Assume the following previous belief
- $\text{bel}(x_{t-1}) = (0.3, 0.7)$
- Which $\overline{\text{bel}}(x_t)$ after action *left*?

$$\overline{\text{bel}}(x_t) = (P_A, P_B)$$

$$P_A = P(A | \text{left}, A) * P(A) + P(A | \text{left}, B) * P(B) = 0.99 * 0.3 + 0.12 * 0.7 = 0.381$$

$$P_B = P(B | \text{left}, A) * P(A) + P(B | \text{left}, B) * P(B) = 0.01 * 0.3 + 0.88 * 0.7 = 0.619$$

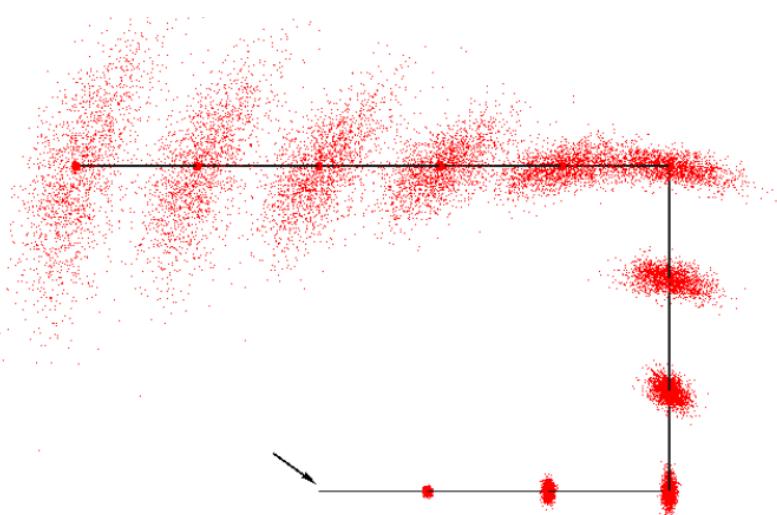
Hence:

$$\overline{\text{bel}}(x_t) = (0.381, 0.619)$$

Localization

Markov Localization – prediction example (2)

- Example of evolution on pose estimation based only on motion model
 - every point represents a possible pose
 - as robot moves, points scatter



Localization

Markov Localization – correction phase

$$bel(x_t) = \eta p(z_t | x_t, m) \overline{bel}(x_t)$$

- Incorporates sensor model

- Input:

- Predicted belief distribution: $\overline{bel}(x_t)$
 - Sensor model

- Based on Bayes formula

$$p(x_t | z_t) = \frac{p(z_t | x_t) * p(x_t)}{p(z_t)}$$

- $p(z_t)$ does not depend on x and may be substituted by a constant

Localization

Markov Localization – correction example

$$bel(x_t) = \eta p(z_t | x_t, m) \overline{bel}(x_t)$$

- Consider the previous world and the belief after prediction

$$\overline{bel}(x_t) = (0.381, 0.619)$$



- Assume the following sensor model

$$\begin{array}{lll} P(A|A) = 0.80 & P(B|A) = 0.15 & P(N|A) = 0.05 \\ P(A|B) = 0.70 & P(B|B) = 0.23 & P(N|B) = 0.07 \end{array}$$

- Assume the sensor measures A

- What is the belief after the correction phase?

$$\begin{aligned} \overline{bel}(x_t)/\eta &= P(A|A) * \overline{bel}(A), P(A|B) * \overline{bel}(B) \\ &= (0.80 * 0.381, 0.23 * 0.619) = (0.3048, 0.1437) \end{aligned}$$

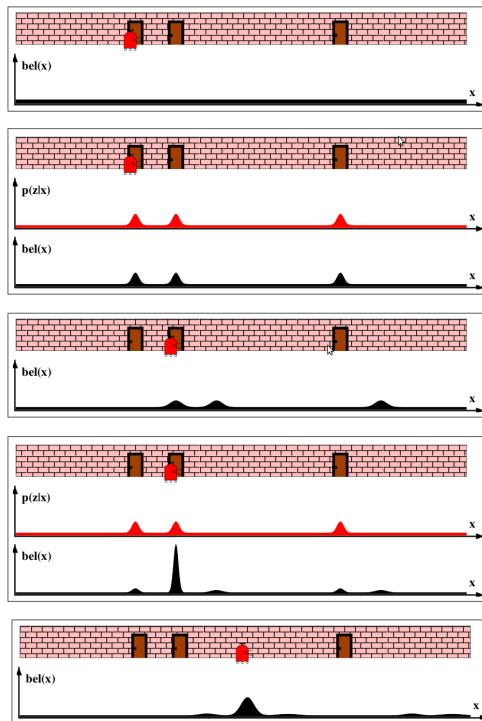
Choosing η as to normalize the belief

$$\overline{bel}(x_t) = (0.6816, 0.3184)$$

Markov localization

Illustration example

- (a) Assuming the initial pose is unknown, belief is a uniform distribution
- (b) Robot senses it is facing a door
 - Integration of sensor data results in a multimodal distribution
- (c) Robot moves some distance to the right
 - convolution with motion model shifts and flattens belief
- (d) Robot senses it is facing a door
 - integration of sensor data allows robot to localize itself
- (e) Robot moves some distance to the right
 - convolution with motion model shifts and flattens belief, but robot keeps itself localized (with less confidence)



Taken from "Probabilistic robotics", Thrun, Burgard & Fox.

Localization

Kalman filter localization

- A case of Markov localization
- Implements belief computation in continuous states
- Belief, motion model and sensor model are represented by Gaussians (**mean** and **covariance**)
 - Belief shape is unimodal
- Prediction phase

$$\mu_C = \mu_1 + \mu_2$$

$$\sigma_C^2 = \sigma_1^2 + \sigma_2^2$$

- Correction phase

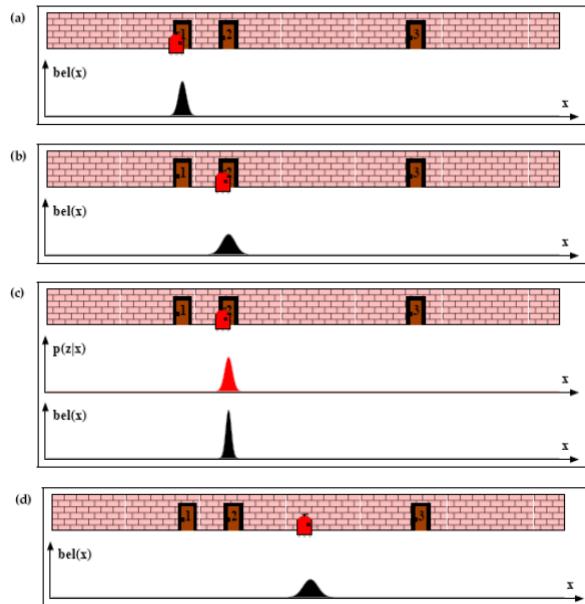
$$\mu_P = \frac{\mu_1 \cdot \sigma_2^2 + \mu_2 \cdot \sigma_1^2}{\sigma_1^2 + \sigma_2^2}$$

$$\sigma_P^2 = \frac{\sigma_1^2 \cdot \sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

Kalman filter localization

Illustration example

- (a) Initial belief is a Gaussian distribution
- (b) Motion model is applied, increasing uncertainty
- (c) Sensor data is integrated, resulting in a variance smaller than variances of belief and sensor model
- (d) Motion model is applied, increasing uncertainty



Taken from "Probabilistic robotics", Thrun, Burgard & Fox.

Kalman filter localization

Extended Kalman filter

- Kalman filters' linear assumption is rarely fulfilled
- **Extended Kalman filters (EKF)**
 - Assume next state and measurement can be non linear

$$x_t = f(u_t, x_{t-1}) + \varepsilon_t$$

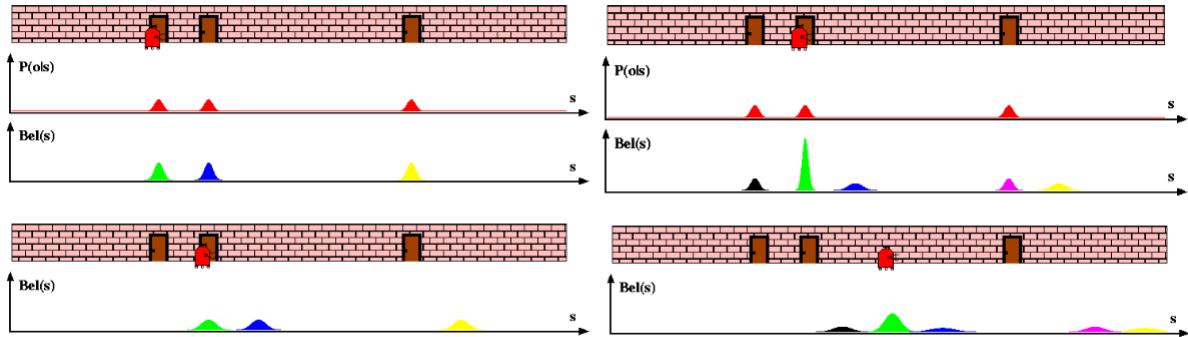
$$z_t = h(x_t) + \delta_t$$

- Moreover, instead of matrices F_t and H_t **Jacobians** derived from f and h are used

Kalman filter localization

Multi-Hypothesis Tracking

- Extension to (extended) Kalman filter
- Belief is represented by multiple Gaussians



Taken from "Probabilistic robotics", Thrun, Burgard & Fox.

Gaussian Localization Summary

- Unimodal Gaussian is a good uncertainty representation for tracking
 - It is not good for global localization
- Not good for hard spatial constraints
 - Unable to process negative information
 - Close to wall, but not inside wall
- Linearization can be an issue
 - depends on degree of nonlinearity
 - depends on degree of uncertainty
- Features must be sufficient and distinguishable
 - Correspondence variables

Grid Localization

Introduction

- Grid decomposition of the **pose space**
- Uses a **histogram filter** to represent posterior belief
- Belief is given by a set of probability values

$$\text{bel}(x_t) = \{p_{k,t}\}$$

where $p_{k,t}$ is defined over a grid cell

- Choosing the **resolution** for the grid cell is a key point
 - High resolution \Rightarrow slow computation
 - Low resolution \Rightarrow information loss

- Can be used to solve the global localization problem
- Not bound to unimodal distributions
- Can process raw sensor measurements

Grid Localization

Algorithm

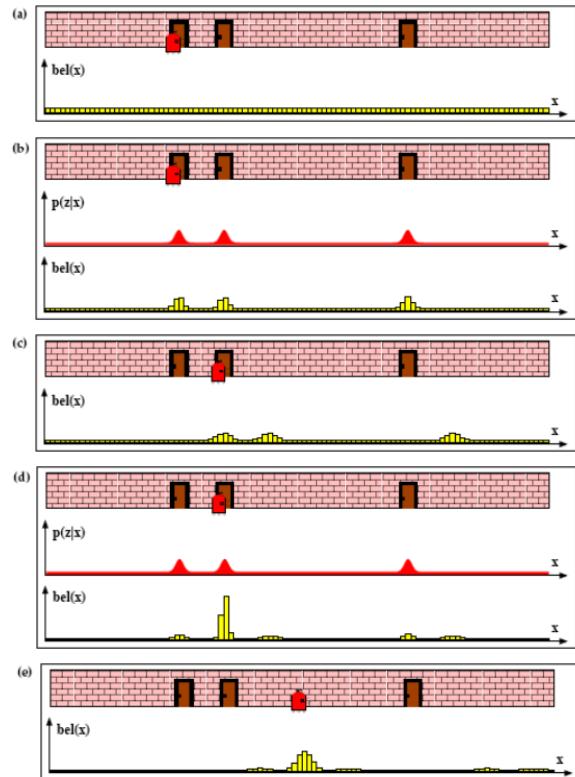
```
Algorithm Grid_localization( $\{p_{k,t-1}\}$ ,  $u_t$ ,  $z_t$ ,  $m$ ):  
    for all  $k$  do  
         $\bar{p}_{k,t} = \sum_i p_{i,t-1} \text{motion\_model}(\text{mean}(\mathbf{x}_k), u_t, \text{mean}(\mathbf{x}_i))$   
         $p_{k,t} = \eta \bar{p}_{k,t} \text{measurement\_model}(z_t, \text{mean}(\mathbf{x}_k), m)$   
    endfor  
    return  $\{p_{k,t}\}$ 
```

- $\{p_{k,t-1}\}$ is the belief at time $t-1$, u_t the actions at time interval $[t-1, t]$, z_t the measurements at time t , and m the map of the environment
- $\{\bar{p}_{k,t}\}$ is the belief at time t based only on the actions
- $\{p_{k,t}\}$ is the belief at time t based on actions and measurements
- η is a normalization factor (from Bayes filter)

Grid Localization

Illustration example

- (a) Belief is a uniform distribution
- (b) First integration of sensor data
 - result is multimodal
- (c) Convolution with motion model, shifts and flattens belief
- (d) Second integration of sensor data, robot localizes itself
- (e) Moving along

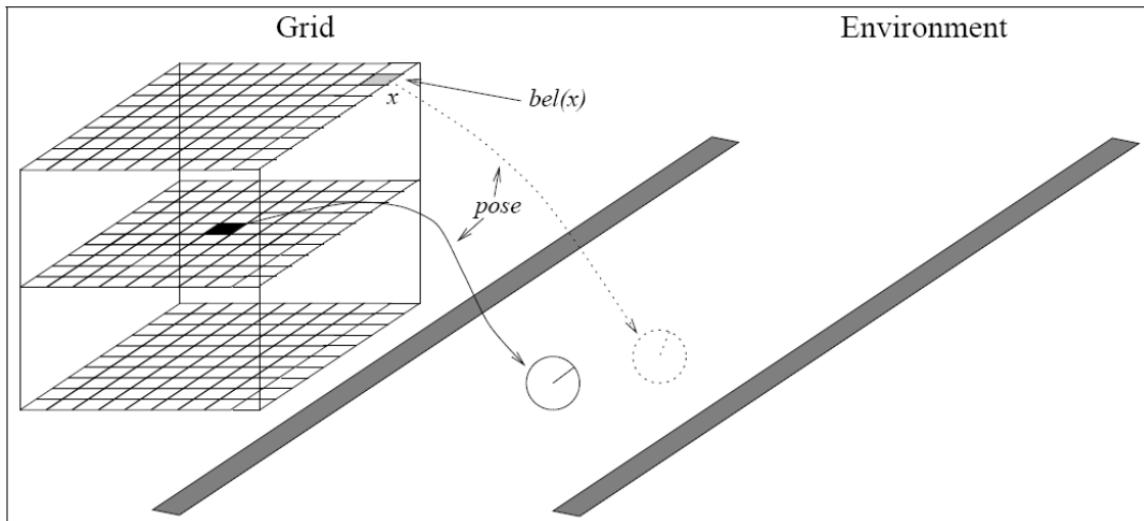


Taken from "Probabilistic robotics", Thrun, Burgard & Fox.

Grid Localization

Example for a 2D pose

- A grid to represent a 2D pose is **cubic**
 - each plan represents a possible robot orientation



Taken from "Probabilistic robotics", Thrun, Burgard & Fox.

Monte Carlo localization

Introduction

- Based on random (educated) guesses drawn in the **pose space**
 - These guesses are known as **particles**
- Belief is given by a set of particles

$$\text{bel}(x_t) = \{x_t^{[k]}\}$$

where $x_t^{[k]}$ represents a pose

- Measurement is used to determine the importance weight of particles
- Weights are used to influence a random selection of particles
 - Heavier particles are more likely to be selected

- Choosing the **number of particles** is a key point

- Big number of particles \Rightarrow slow computation
- Small number of particles \Rightarrow information loss

- Can be used to solve the global localization problem
- Not bound to unimodal distributions

Monte Carlo localization

Algorithm

Algorithm MCL(X_{t-1}, u_t, z_t, m):

$$\bar{X}_t = X_t = \emptyset$$

for $i = 1$ to M **do**

$$x_t^{[i]} = \text{sample_motion_model } (u_t, x_{t-1}^{[i]}, m)$$

$$\omega_t^{[i]} = \text{sample_mesurement_model } (z_t, x_t^{[i]}, m)$$

$$\bar{X}_t = \bar{X}_t + \langle x_t^{[i]}, \omega_t^{[i]} \rangle$$

end for

for $i = 1$ to M **do**

draw $x_t^{[i]}$ with probability $\propto \omega_t^{[i]}$

$$X_t = X_t + x_t^{[i]}$$

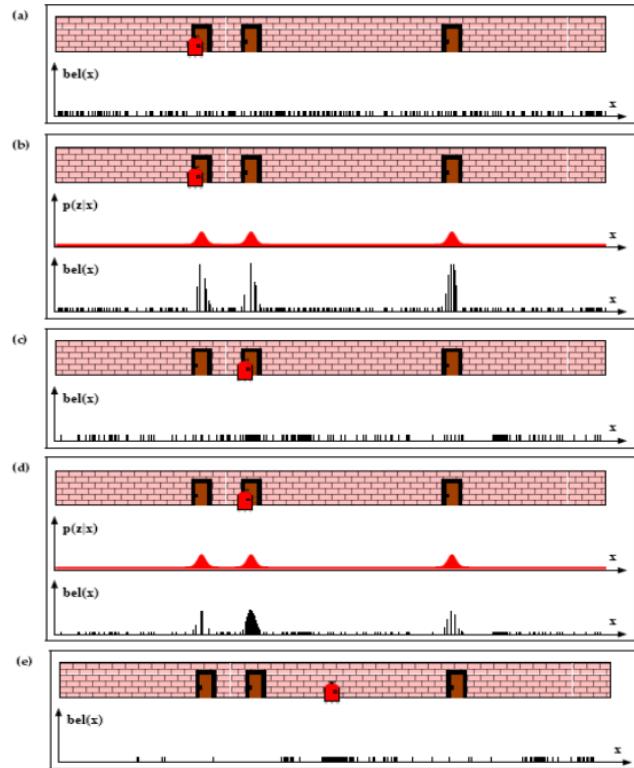
end for

return X_t

Monte Carlo localization

Example

- (a) Pose particles drawn at random and uniformly
- (b) Importance factor assigned to each particle
 - set of particles hasn't changed
- (c) After resampling and incorporating robot motion
- (d) New measurement assigns new importance factors
- (e) New resampling and motion



Taken from "Probabilistic robotics", Thrun, Burgard & Fox.

Monte Carlo localization

Example (2)

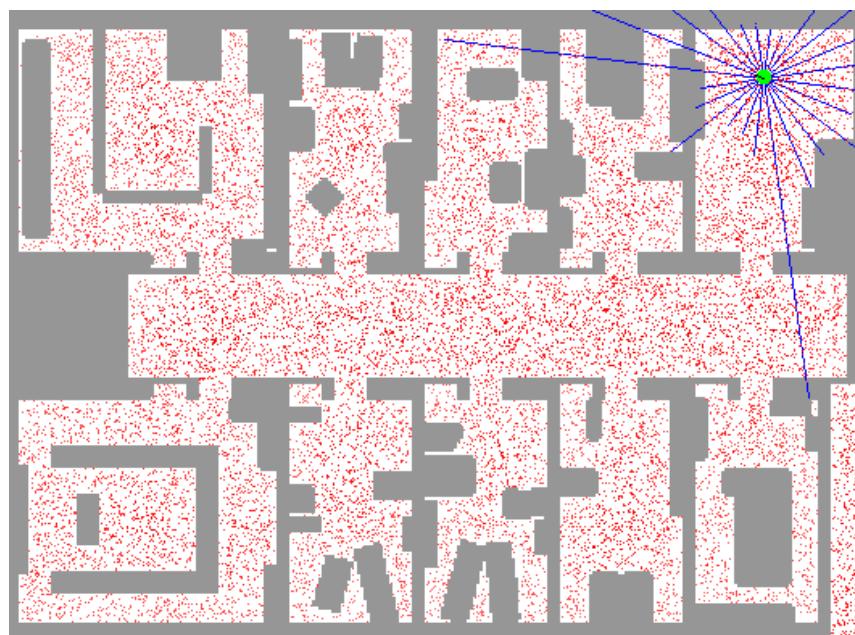


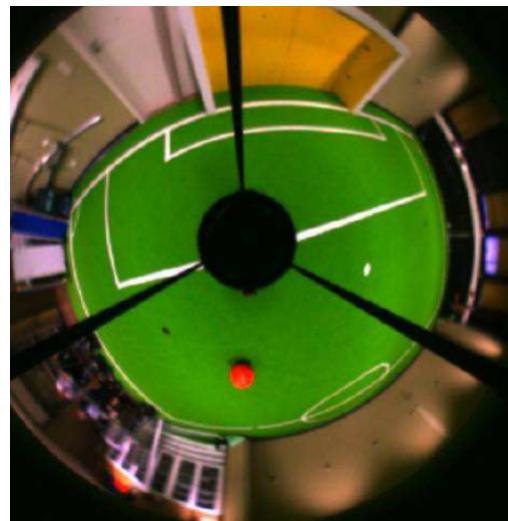
Image source <https://rse-lab.cs.washington.edu/projects/mcl>

Download image; it is an animated gif

Localization in CAMBADA

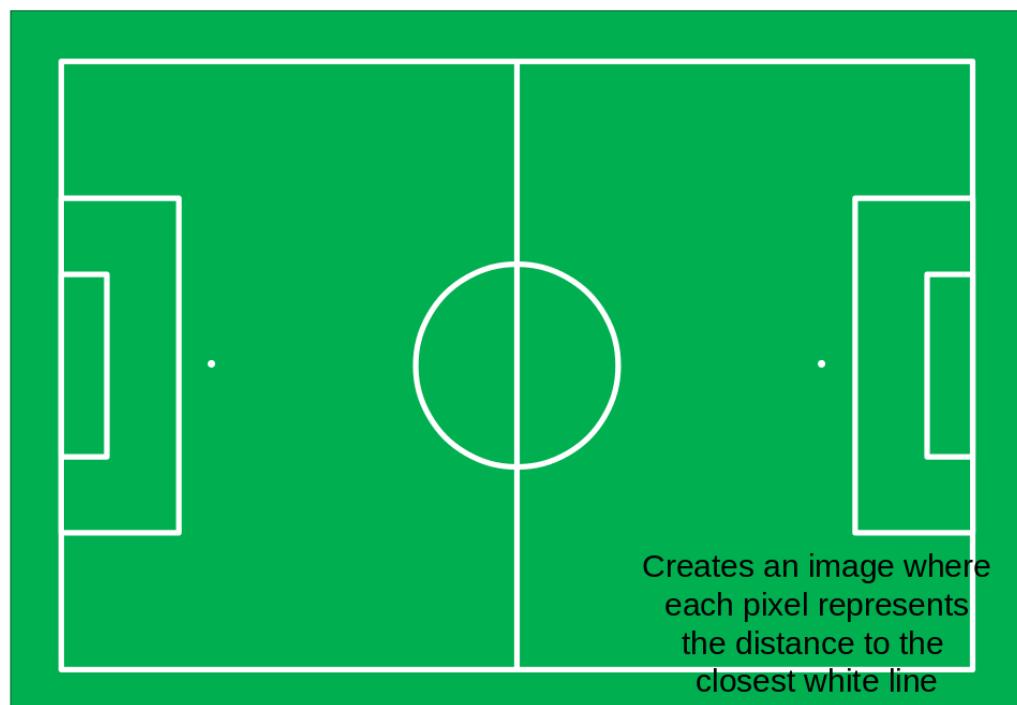
Approach

- Based on Tribots localization
- Uses white lines seen by the robot
 - captured using an omni camera
- A correction map converts pixels to real distances
 - this map is constructed in a calibration phase
- A distance map of the field is used to correct robot pose
 - this map is constructed in advance and kept in a lookup table



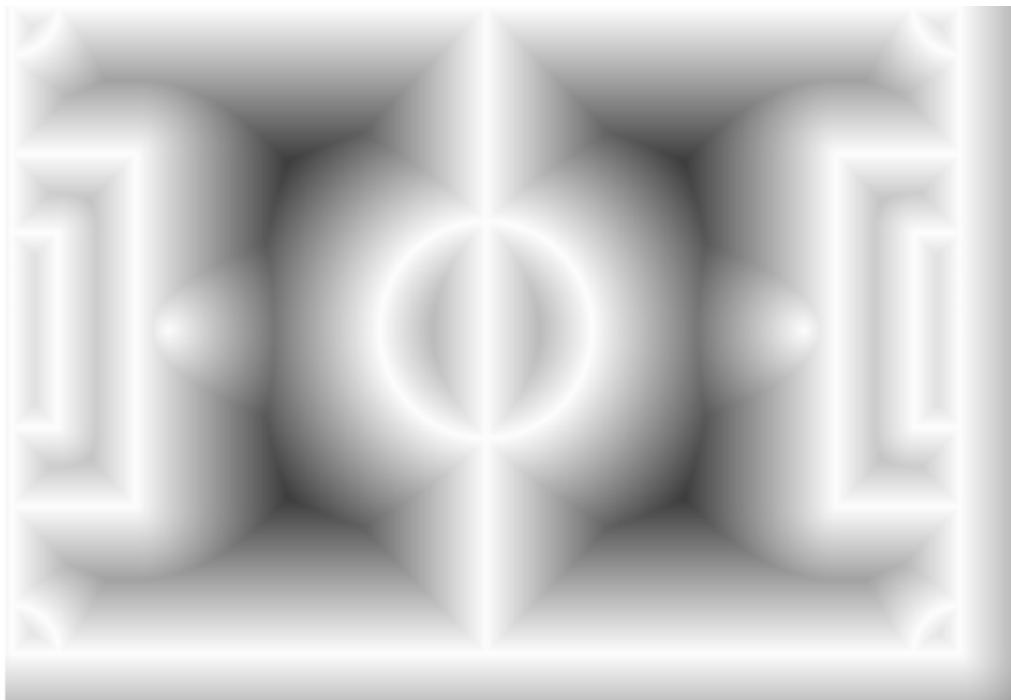
Localization in CAMBADA

Building the field LUT



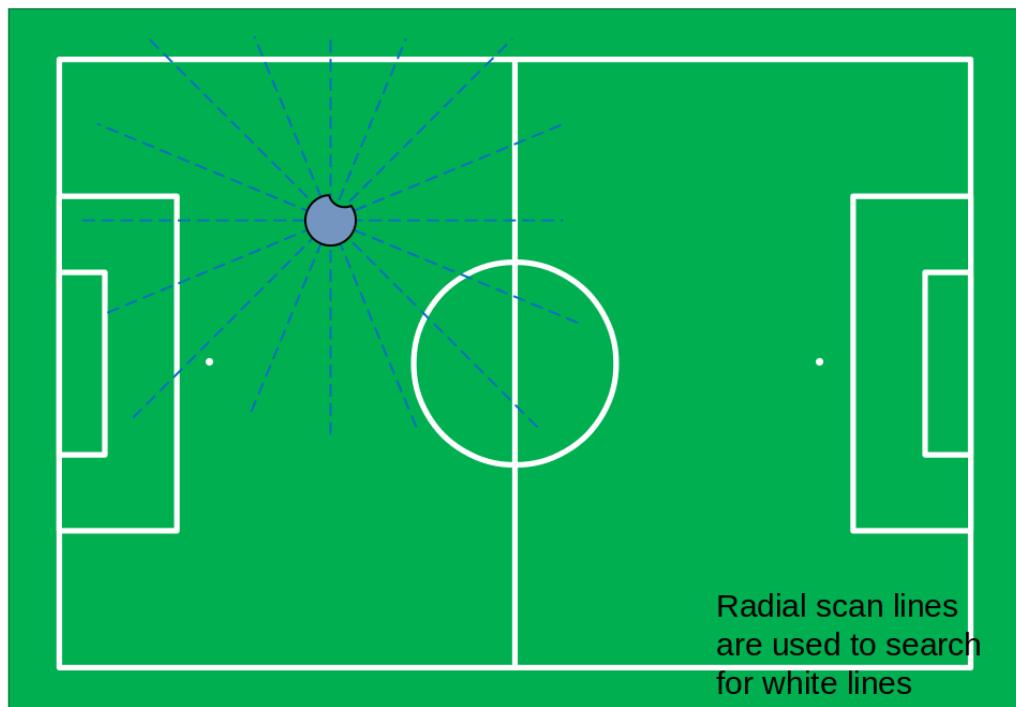
Localization in CAMBADA

Building the field LUT



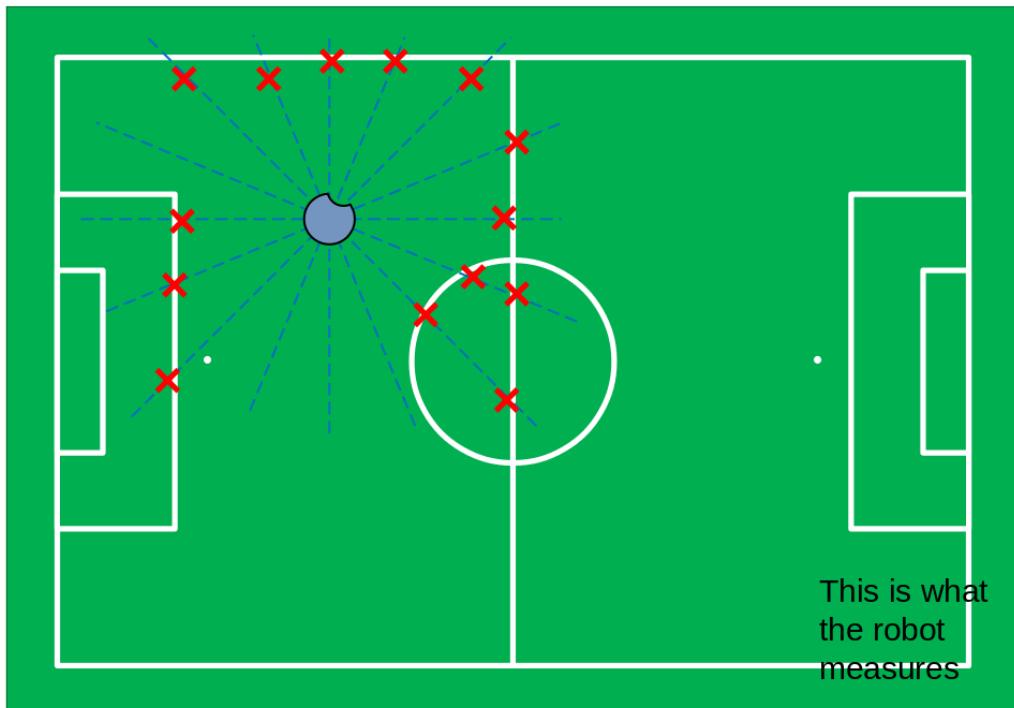
Localization in CAMBADA

Getting visual lines, real pose



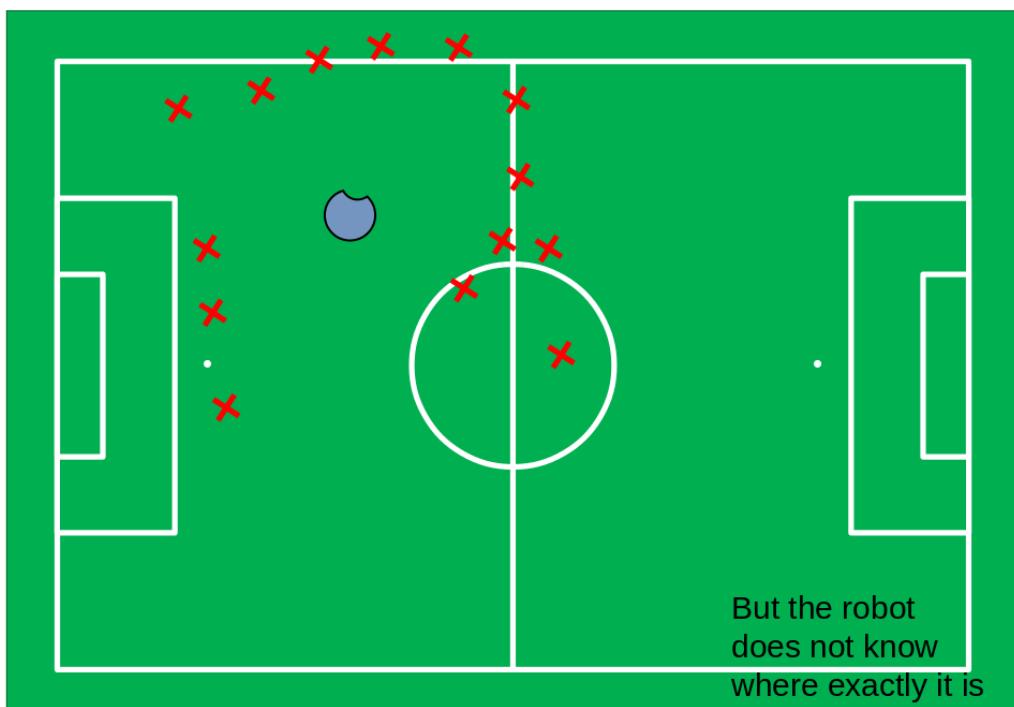
Localization in CAMBADA

Getting visual lines, real pose



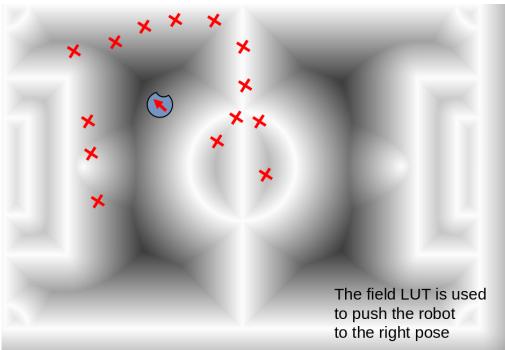
Localization in CAMBADA

Lines in estimated pose

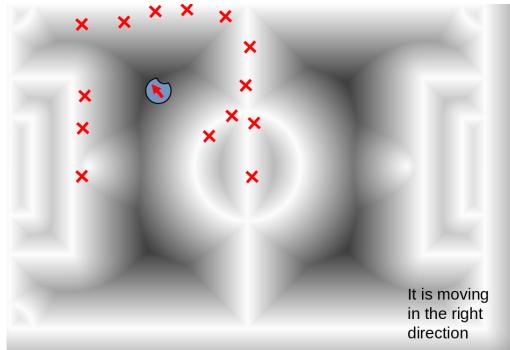


Localization in CAMBADA

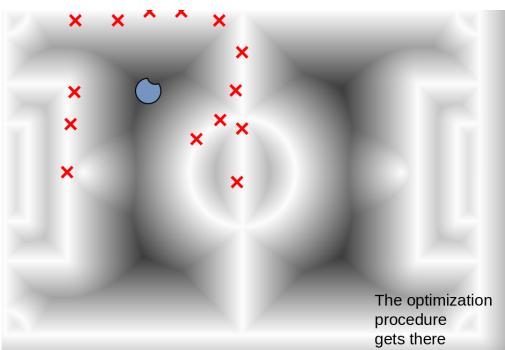
Correcting pose



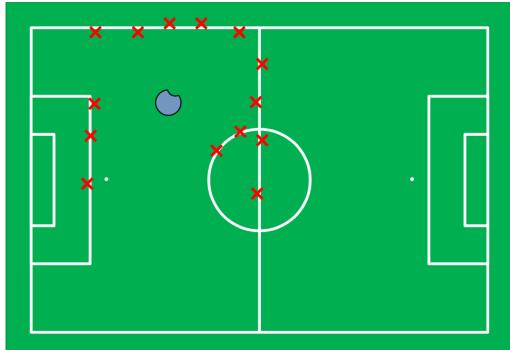
The field LUT is used to push the robot to the right pose



It is moving in the right direction



The optimization procedure gets there

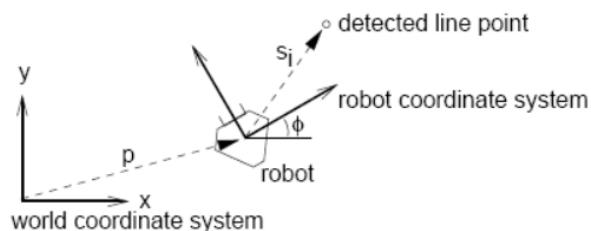


Localization in CAMBADA

Error function

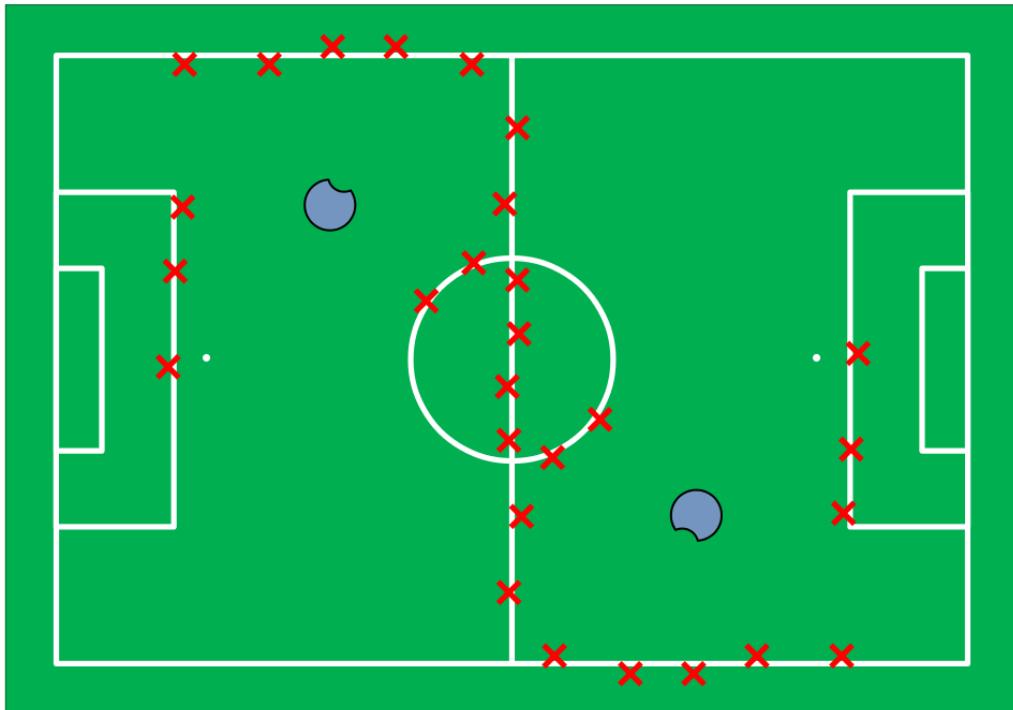
$$\underset{\mathbf{p}, \phi}{\text{minimize}} \quad E := \sum_{i=1}^n \text{err}(d(\mathbf{p} + \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix} \mathbf{s}_i))$$

- \mathbf{p} and θ are the position and heading
- \mathbf{s}_i is the position of a detected white line
- Mapping $d(\cdot)$ gives the distance from a point in the field to the closest white line



Localization in CAMBADA

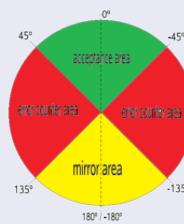
Symmetric position problem



Localization in CAMBADA

Tracking

- Robot optimizes previous position (updated with odometry) and also 4 positions with:
 - fixed offsets of 60cm in xx and yy positive and negative dirs
 - small random heading offset
 - The optimized position with the smallest error is taken as the best estimate
- Detection of symmetric position
 - Compass based, if possible
 - compass divided into 4 regions
- Detection of lost condition
 - Compass based, if possible
 - Forces global localization algorithm



Localization in CAMBADA

Global localization

- A grid of trial points is used as candidate position for optimization
 - Grid spans one half of the field
 - Resolution of 1m over xx and yy
- Initial heading may be:
 - Based on compass (allows use without human intervention)
 - Fixed, ex: robot oriented towards positive xx (for fatidic fields)
- Optimized position with smallest error is chosen
- A set of 4 neighbors of smallest error position (using 40cm offsets) are still checked for better precision

Bibliography

- “Probabilistic Robotics”, Sebastian Thrun, Wolfram Burgard, Dieter Fox, MIT Press, Cambridge, Massachusetts, London England, 2005.
- “Calculating the perfect match: An efficient and accurate approach for robot self-localisation”, Martin Lauer, Sascha Lange and Martin Riedmiller, RoboCup 2005: Robot Soccer World Cup IX, LNCS.
- “The Robotics Primer”, Maja J. Mataric, The MIT Press

Robótica Móvel e Inteligente

José Luís Azevedo, Bernardo Cunha, Pedro Fonseca,
Nuno Lau e Artur Pereira

Robot architecture:

- **Sensors**
- **Actuators**
- **Locomotion**

Actuators in mobile robotics

An actuator can be broadly defined as an active device that converts a primary energy source into physical movement.

Given this definition, the actuators can be generally classified considering:

1. The type of **primary energy** which is used to generate motion in the actuator
2. The type of **generated movement**

For each of the above types of classification it is also possible to classify the actuators according to the type of technology (physical principle of energy transformation into physical movement).

NOTE: some devices which are classified as distinct actuators may, actually, be a more sophisticated version of simple actuators (for example through the integration of the control system at the actuator level).

Actuators in mobile robotics

Classification of the actuators according to the **type of primary energy** that is used to generate movement in the actuator:

1. Electrical. Transform primary electrical energy, usually stored in batteries (...rechargeable), into the intended movement. As we will see, there are different technologies depending on the characteristics of the specific application of the actuator. In mobile robotics, electrical actuators represent the main class of actuators used due to its diversity and adaptability to every type of application



Examples of a small and medium size electrical DC motors

Actuators in mobile robotics

Classification of the actuators according to the **type of primary energy** that is used to generate movement in the actuator:

2. Pneumatic. Powered by energy stored as compressed air (or another compressed gas). These actuators are used in situations where **accurate and easy to control movements is desired** but where the required force is not a critical criteria.

Requires very high pressures and therefore an extremely robust and heavy container.

Less efficient in converting energy than electrical solutions, therefore being less common in mobile robotics applications.



Example: pneumatic pistons

Actuators in mobile robotics

Classification of the actuators according to the **type of primary energy** that is used to generate movement in the actuator:

3. Hydraulic. They use the flow and pressure of a fluid to convert the primary energy into linear and/or rotational motion or torque.

Such actuators are used only in **situations where the force required is extremely high**. They also require **heavy and bulky mechanical structures**.

Typically found in large machines such as autonomous mobile robots working in mines. In the field of robots covered by this course this is not a relevant type of actuator.



Actuators in mobile robotics

Classification of the actuators according to the type of **generated movement** produced by the actuator:

1. **Rotary.** The primary energy is converted into rotating motion, which in turn may be continuous, positional or a source of application of force (torque)
2. **Linear.** The primary energy is converted into linear motion. The purpose of this movement can also include the direct application of a force, the positioning of a mechanical element or the execution of a continuous repetitive motion (*reciprocating system*).

Actuators in mobile robotics

NOTES:

1. When the objective is the direct application of a force, it is common to find solutions where **the primary power applied to the actuator is partially stored in a mechanical device** (e.g. a spring) so that the original configuration can be restored when the primary energy source is removed. Passive energy storage elements can also adjust target positions based on external forces (**compliant actuators as opposed to stiff**).
2. In some cases, the very primary energy of electric, pneumatic and / or hydraulic actuators can in fact be of a different nature, particularly when the required power is very significant.

One example of what is stated above is the use of an internal combustion engine in which the primary energy, derived from the combustion of hydrocarbons, is in turn converted into electrical, pneumatic or hydraulic energy.

Example of hydrocarbons as primary source

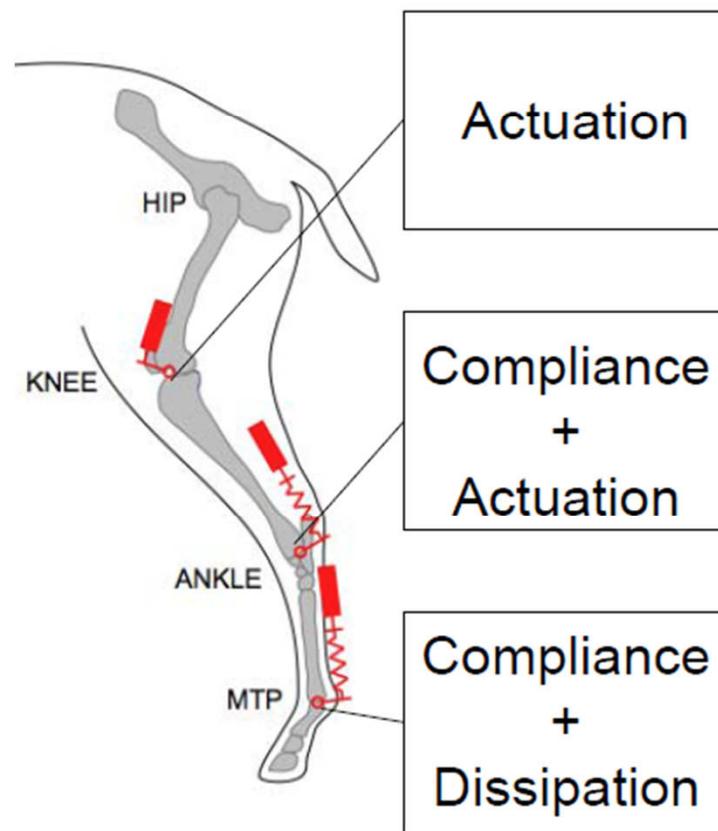
BigDog by Boston Dynamics

- powered by a **two-stroke, one-cylinder**, 15-HP go-kart engine operating at over 9,000 RPM.
- **engine drives an hydraulic pump**, which in turn drives the hydraulic leg actuators.
- Each leg has four actuators
 - Each actuator unit consists of a hydraulic cylinder, servo valve, position sensor, and force sensor.

http://www.bostondynamics.com/img/BigDog_Overview.pdf

Multi-jointed Legs

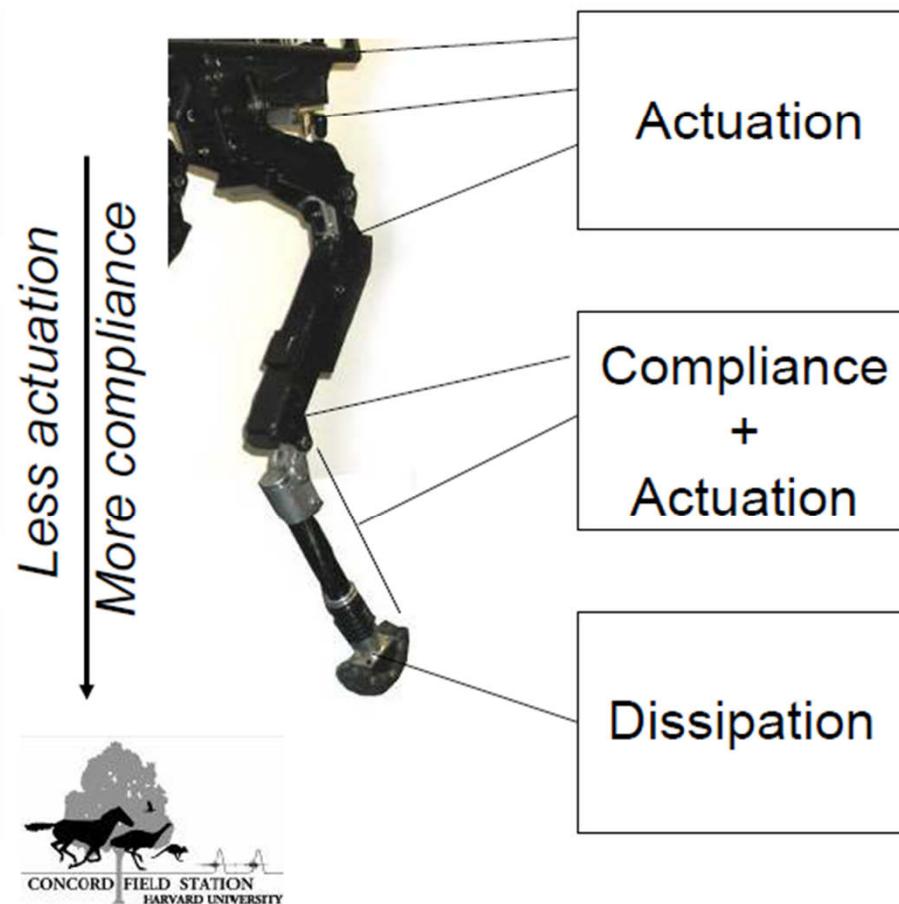
Animal



BostonDynamics



BigDog



http://www.bostondynamics.com/img/BigDog_Overview.pdf

Actuators in mobile robotics



Rotary Actuators

The most common rotary actuators used in autonomous mobile robots (if we exclude the very large robots) use electricity as its primary source of energy.

These actuators, are commonly referred to as "drivers" or "motors".

Because they are an essential component in the mechanism of locomotion of robots, they may in turn be of various types regarding the way the electrical energy is transformed into mechanical energy.

Given their relevance in intelligent mobile robotics, there are three types that must be highlighted:

- **Brushed DC motors**
- **Brushless DC motors**
- **Stepper motors**

Rotary Actuators

Regardless of their types, electric motors may be characterized by a set of attributes whose values may be more or less important according to its application. In general, it is the specific application what determines the required characteristics of the motor, and hence, its final selection. Some of the most important parameters to consider are:

- Nominal supply voltage [V] - Range(3 .. 48)*
- Torque response as a function of rotation speed [Nm/min⁻¹]
- Stall torque [Nm] - Range(mNm .. 100sNm)
- Steady duty maximum torque (BMrc) [Nm] - Range(mNm .. 100sNm)
- Typical current in steady duty [A] - Range(mA .. 100A)
- Nominal angular velocity with no load [min⁻¹] - Range(0.1 .. 50,000RPM)**
- Nominal angular velocity in steady duty (VAnrc)] - Range(0.1 .. 35,000RPM)**
- Nominal power (typ: BMrc * VAnrc) [W]] - Range(mW .. 500W)

* Indicative figures for mobile robotics applications

** The rotation speed of the main shaft is often dependent on the mechanical gearing box which is coupled to the motor

Rotary Actuators

We may also consider other parameters such as

- Startup current [A] - Range(mA .. >100A)
- Peak current [A] - Range(mA .. >100A)
- Mechanical time constant[s] - Range(ms .. >100ms)
- Rotor inertia [gcm^2] - Range(0.1 .. >2500 gcm^2)
- Motor volume [cm^3] - Range(2 cm^3 .. >200 cm^3)
- Weight [Kg] - Range(<10g .. >10Kg)

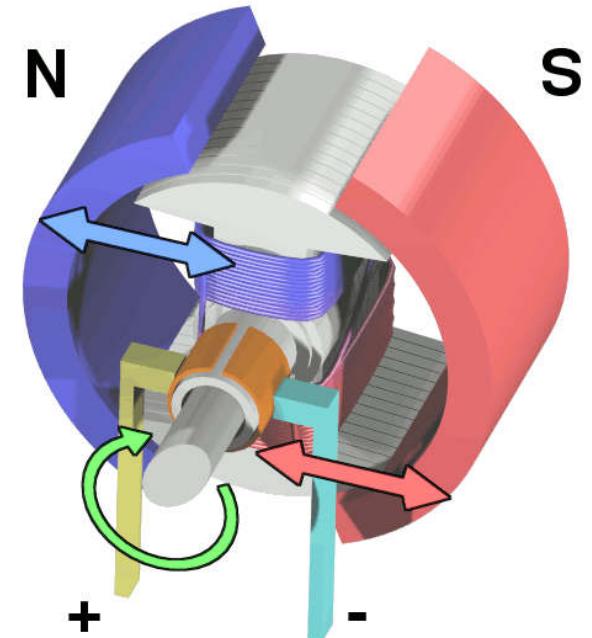
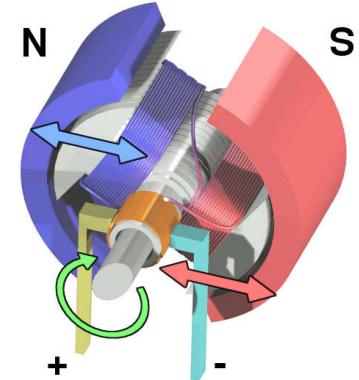
Depending on the technology and the application, we may also need to consider other parameters such as:

- Efficiency [%]
- Number of poles
- Magnetic characteristics of the rotor and / or stator of the motor
- Mechanical rotor configuration (inrunner or outrunner)
- Geometry of the motor (e.g. planar motors)

Brushed DC motors

In the version shown in the image, there are two poles (**permanent magnets**) which constitute the **stator**.

The **rotor** includes two coils wounded around ferromagnetic armatures that, when traversed by current, generate a magnetic field that, in conjunction with the field of the permanent magnets, generates a **rotational force**.



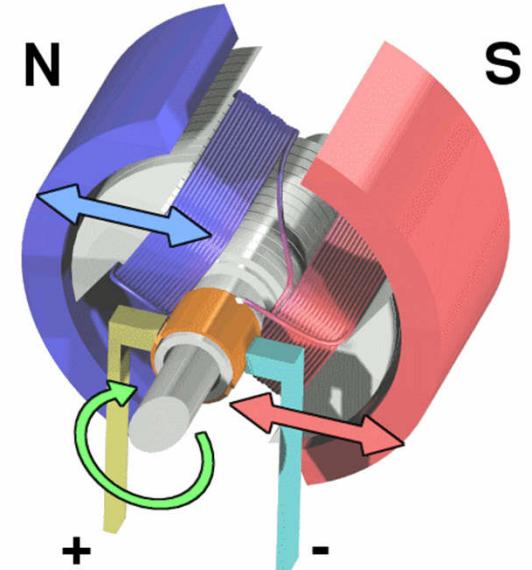
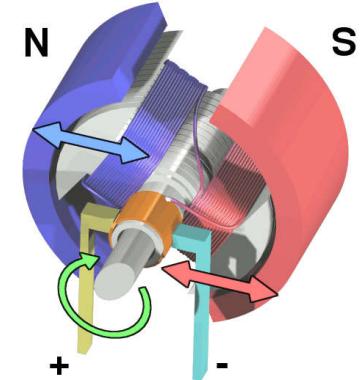
http://en.wikipedia.org/wiki/Brushed_DC_electric_motor

Brushed DC motors

Switching the direction of current flow for ensuring that the torque remains in the same direction **is performed mechanically** (by the so called **brushes**).

In a simple version as that of the image (only two poles) there is a balance point that cancels the forces. A motor in this configuration can not start by itself.

In real solutions, the number of coils in the armored rotor is, at least, equal to or greater than three.

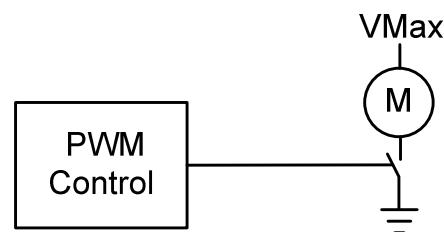
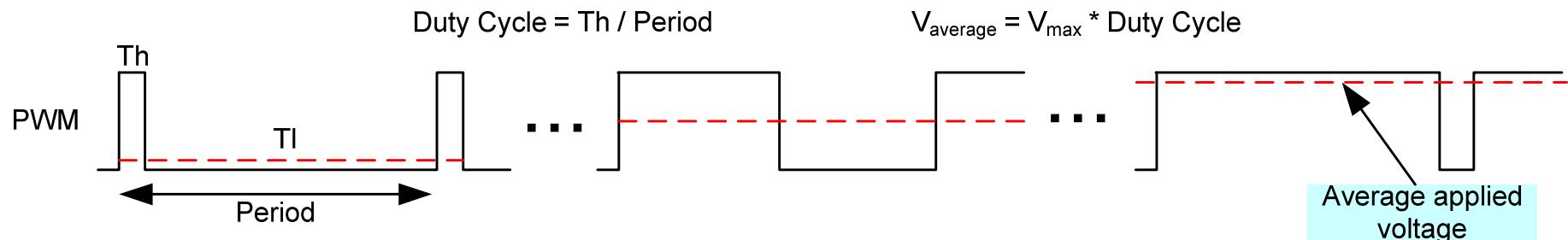


http://en.wikipedia.org/wiki/Brushed_DC_electric_motor

Brushed DC motors

Speed control of DC motors can be carried out by directly **varying the voltage applied to the terminals**.

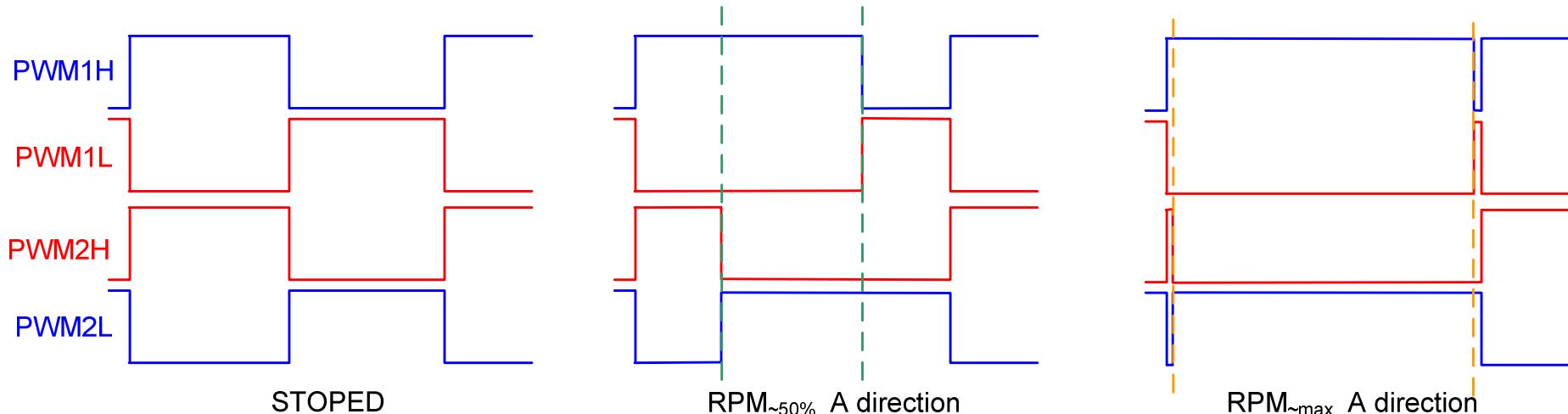
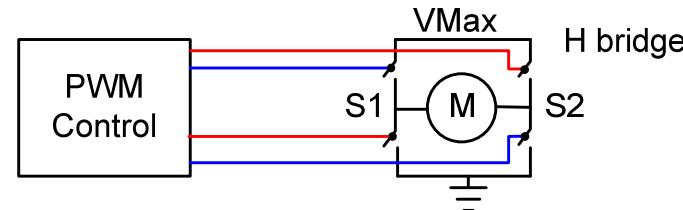
Since most of the current controllers are digital, this voltage variation is done by modulating the pulse width of a constant voltage (**PWM** controller) that is applied at a fixed frequency to the motor terminals.



Brushed DC motors

To **change the direction of rotation** of a DC motor, the polarity of the applied **voltage must be reversed**.

For that an H-bridge circuit is normally used, allowing the full control of the motor, in terms of speed and direction of rotation.



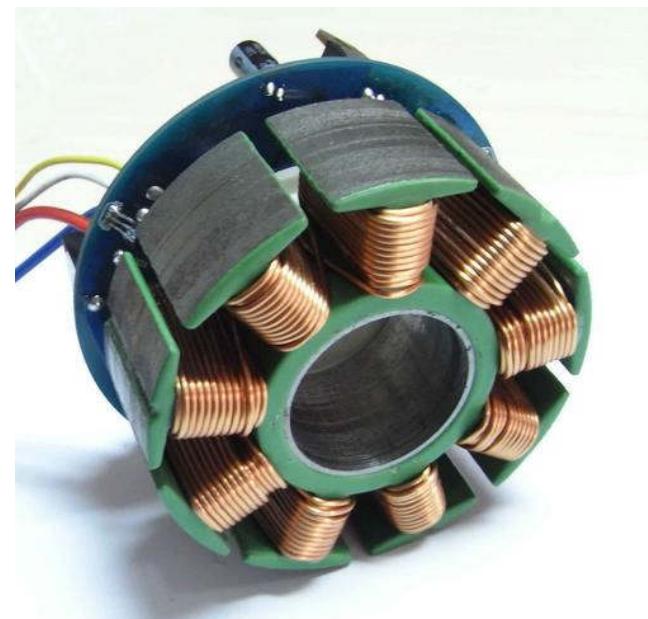
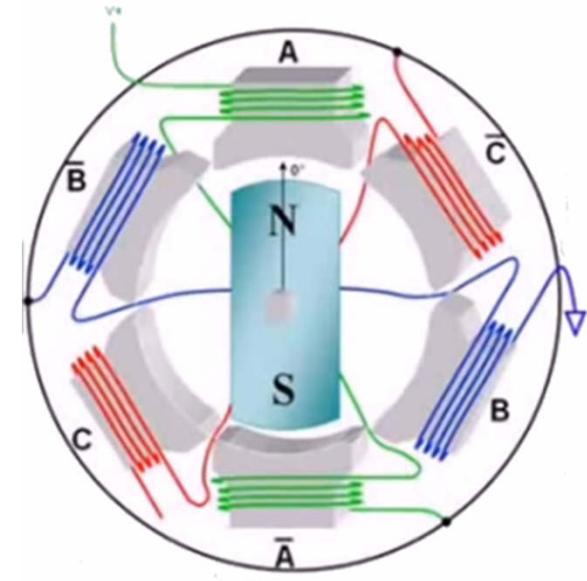
BrushLess DC motors (BLDC)

A brushless DC motor uses **permanent magnets, mounted on the rotor**, and a fixed armature on which a group of winding pairs of electromagnetic poles are mounted.

These **poles are excited by an externally applied voltage**.

This configuration **eliminates the need for a mechanical switching system (brushes)**

However, it **requires an electronic control system which ensures the correct sequential switching of the poles**, in order to generate the necessary torque that produces the correct rotation of the rotor.



BrushLess DC motors(BLDC)

The BLDC motors require virtually **no mechanical maintenance** and exhibit greater **efficiency** when compared with the brushed

In contrast, to control a BLDC is more complicated, in particular since **it requires a mechanism to provide information on the absolute position of the rotor** (Hall effect sensors or EMF measurements). The # of pairs of transistors should be equal to $2 * \# \text{poles}$ (or $4 * \# \text{poles}$ for bi-directional control).



800W Out-runner at 9,000RPMs

Mechanical configurations for BLDC solutions include inner rotors (in-runners), outer rotors (out-runners) or overlapping rotors and stators (planar motor).

From the electrical point of view, the interconnection of the poles can also adopt triangle or star configurations.

Stepper Motors

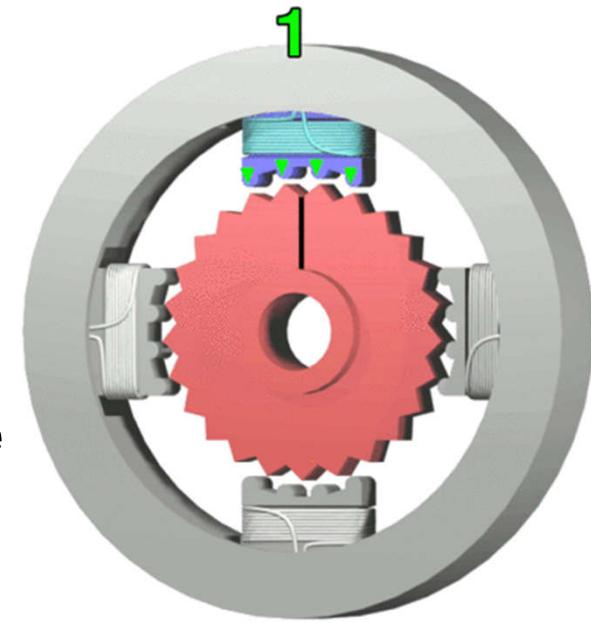
The stepper motor consists of a **rotor containing multiple magnets arranged on a central piece of iron**, and a **set of electrical coils disposed about the periphery** ferromagnetic materials, which is also **toothed**.

When a **current** is injected in one of the coils, it **generates a magnetic field** that attracts a subset of the permanent magnets until their **teeth become aligned**.

In this configuration, **the teeth of the other coils** are **slightly out of phase** relative to the rotor teeth.

Thus, when switching the current to the next coil, a new binary is generated to trigger a new alignment.

By **sequentially feeding the coils**, one can generate a rotary motion to a controlled-speed or put the rotor in a desired fixed angle.



Brief comparison between the three types of motors

Brushed DC	Brushless DC	Stepper Motors
+ cheap	+ efficiency	+ very precise (can be used in open loop)
+ simple to control	+ large range of rotational speeds	+ good for planar topologies
+ large power range	+ maintenance	+ useful in positioning applications
+ very simple startup	+ power vs. volume	+ maintenance
-	+ good for planar topologies - price	-- reasonable compromise between complexity and control
- mechanical wear-out	- complex control	-- price
- efficiency	- complex startup and/or need for accurate position sensors	-- efficiency
- power vs. volume		- mechanical vibration

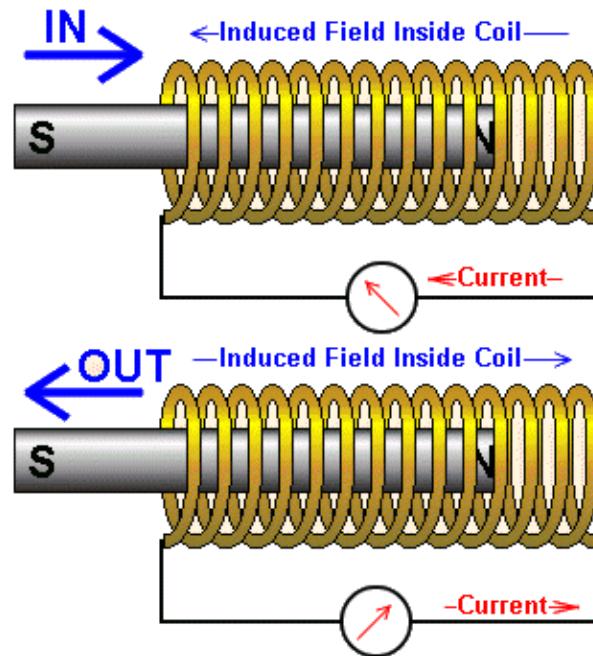
Solenoids

A solenoid is typically built from a winding (**coil**) mounted around a **movable plunger** within an air gap.

The plunger consists of a ferromagnetic material or a permanent magnet (which is the case of image below).

If it consists of a ferromagnetic material, when a **current is applied to the coil a magnetic field is produced** that, in turn, creates a dipole field in the plunger which generates a force to **align its geometrical center with the center of the coil**.

In those situations in which the plunger is a permanent magnet, the direction of the current and the starting position of the plunger generates a force that may be repulsive or attractive

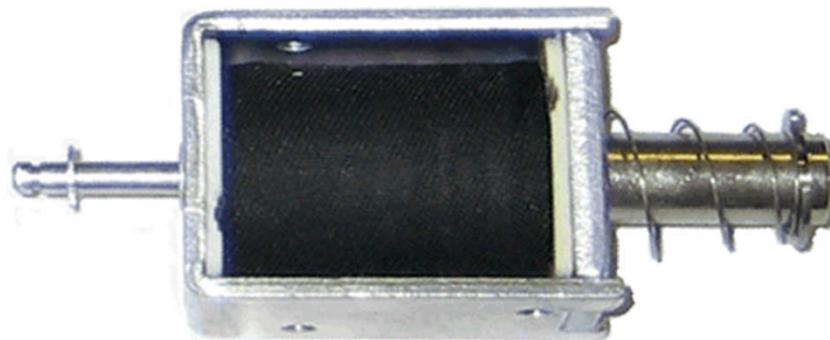


Solenoids

In most cases, the solenoid starts from a position where the plunger is geometrically offset with respect to the center, and is rapidly attracted in the opposite direction when the coil is energized.

In order to maintain the exerted force, the plunger is mechanically locked before reaching the equilibrium point.

To reset the initial conditions, a passive element (eg. spring) that accumulates the resulting energy from the shift, pushes the plunger again to the starting position when power is removed.

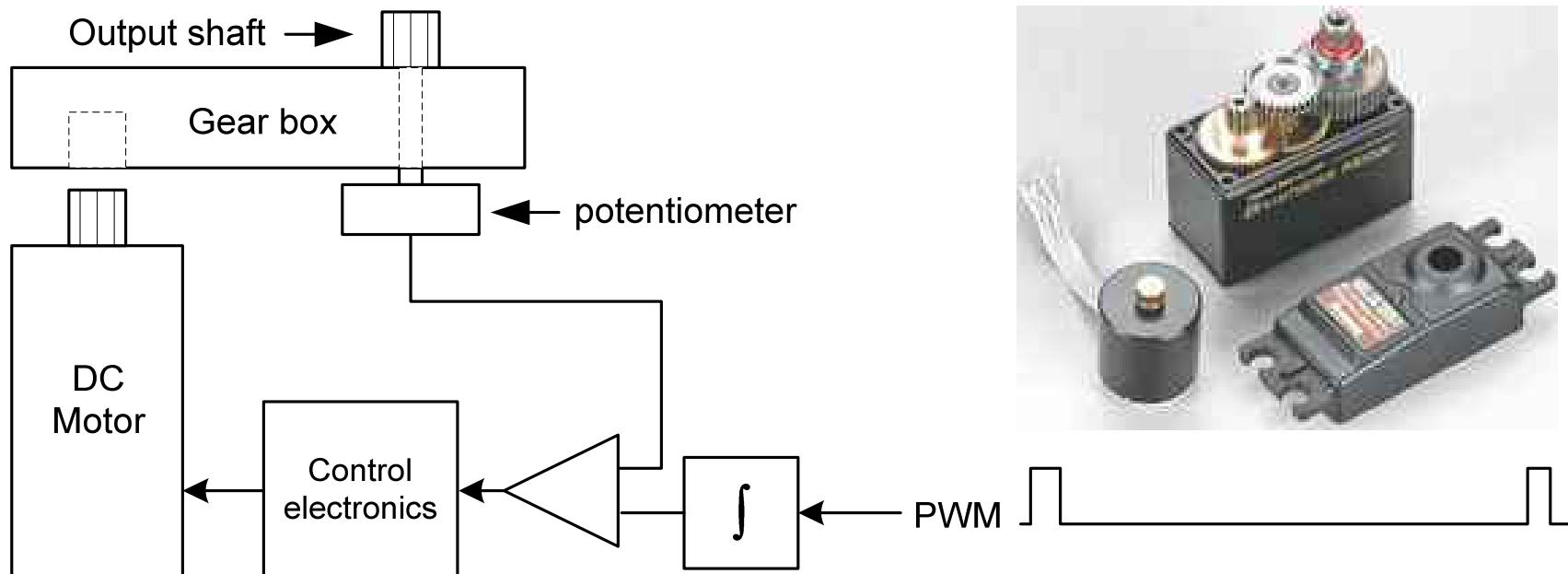


Servo Motors

Servo (or **servo motor**) is the name given to a combination of a **DC (or BLDC) motor**, a **reduction gear box** and an **electronic control loop**.

Servos are intended, usually, for applications of **angular positioning** which require **good accuracy** (which can also be transformed into linear movement)

In the simplest servos, the control is carried out by applying a PWM signal to the servo. The feedback loop includes a potentiometer which indicates the angular position of the output shaft.



Servo Motors

The main parameters to consider when choosing a servo are:

- Maximum available torque
- Angular velocity range
- Maximum covered angle (or free rotation)
- Angular resolution
- Gear box materials
- Type of control
- Dimension and weight

Hobby servo



Industrial servo



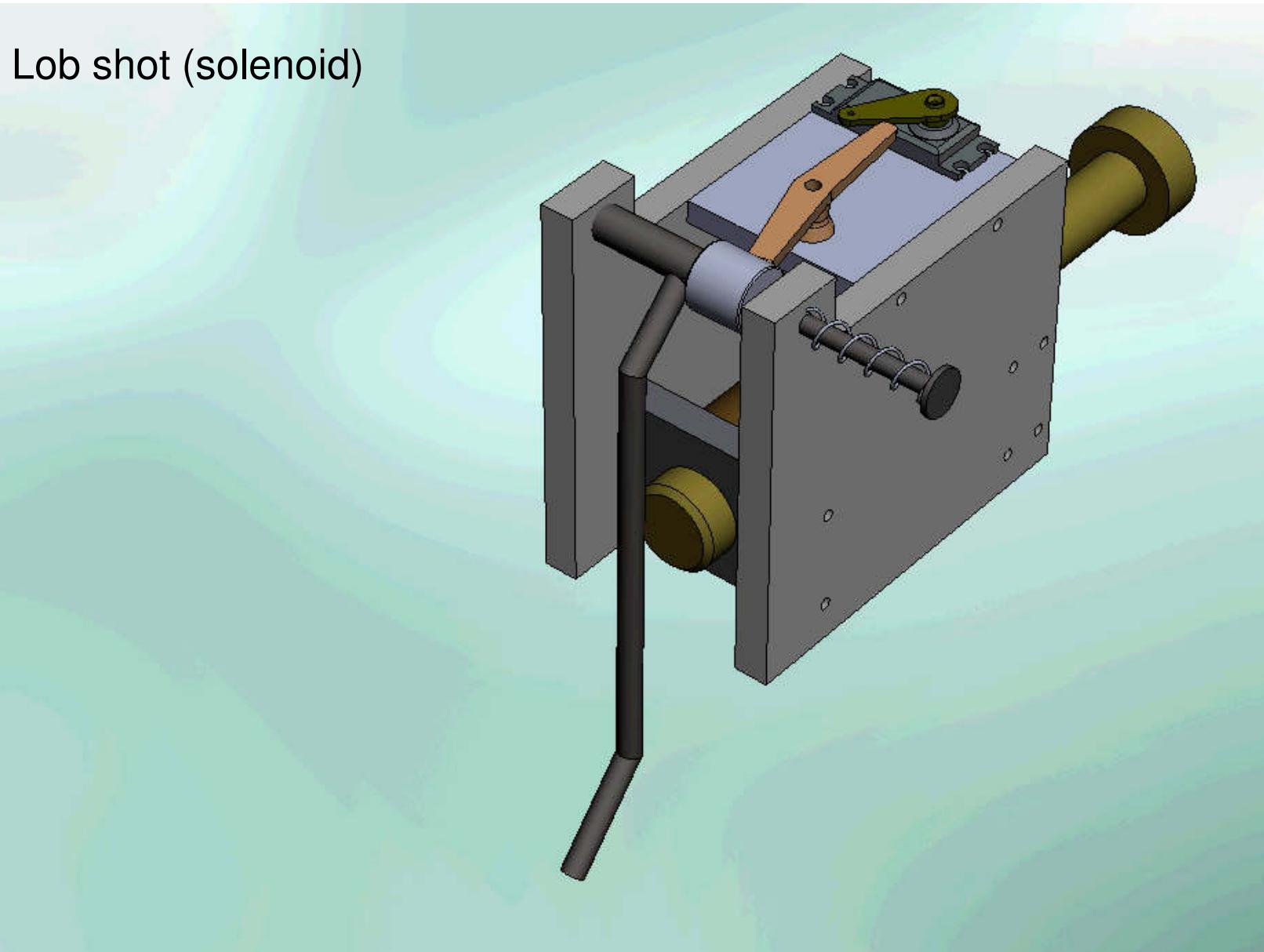
E.g. – In the case of Dynamixel servos, control is performed via digital commands sent through a digital multi-drop bus, allowing, among others, a precise control (within some limits) of parameters such as end angle, angular velocity and maximum torque to be applied.



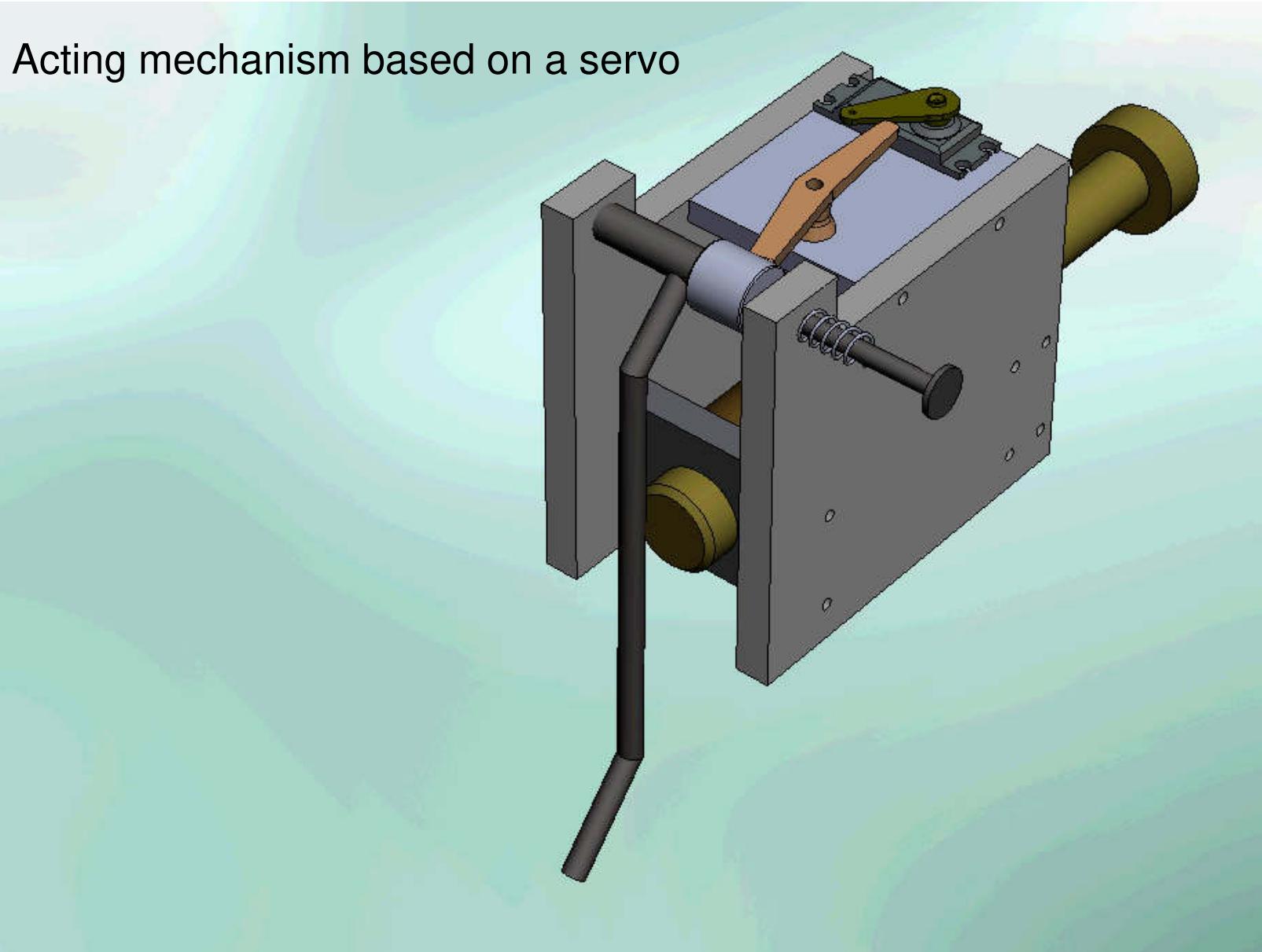
RobotWorx

Dynamixel servo

Combination of actuators - example (CAMBADA)

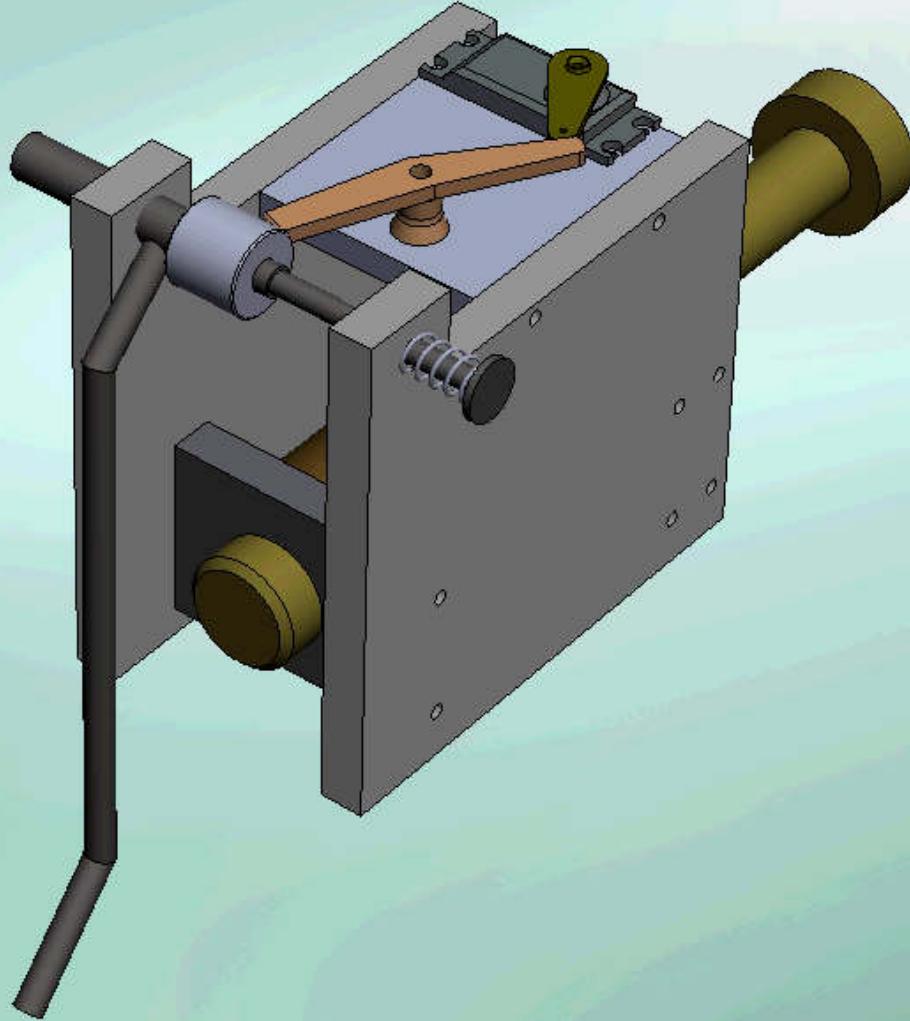


Combination of actuators - example (CAMBADA)



Combination of actuators – example (CAMBADA)

Low shot (solenoid)

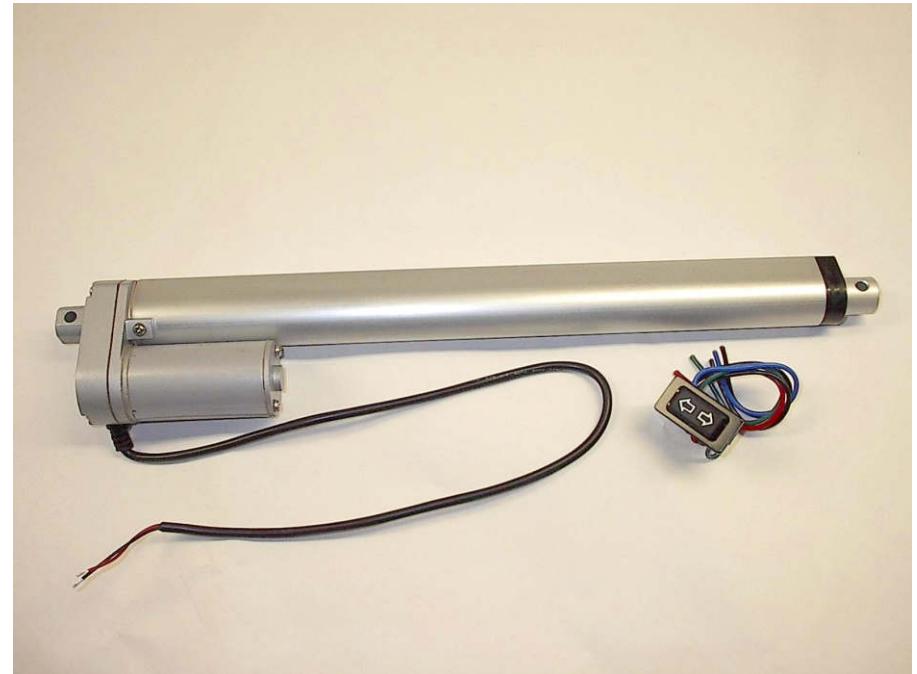


Linear Actuators

A linear actuator normally comprises a DC motor associated with a worm screw on which, or from which, the actuating element is moved.

Depending on the direction of rotation of the DC motor, the actuating element approaches or moves away from the original boundary of motion. The inclusion of a potentiometer and/or limit switches is common in this type of actuators.

Equally common are the linear actuators using pneumatic or hydraulic energy. In these cases the actuator is associated to a piston that is subject to positive or negative pressures to trigger the desired movement.



Other actuators

Pneumatic artificial muscles - also known as **air muscles** are based on the stretch and contraction of an elastic tube by varying the air pressure within the same. Contractions can raise up to about 40% of the original dimension.

Wire muscle - based on a compound known as Nitinol or Flexinol, this type of string can **shrink (up to about 5%) when subjected to an electric current**. Can be used in small dimension applications.

Electroactive polymers – based on plastic materials (EAPs or EPAM: electroactive Polymer Artificial Muscle) they **can expand substantially due to an electrical signal applied**. This type of actuator does not support, however, large mechanical loads.

Piezo-electric motors - used in micro and nano robotics, they use the piezo-electric principle to produce electro-mechanical vibration (tens of kilohertz) which can be transformed into rotational or linear motion.



Mobile robot locomotion

Robótica Móvel e Inteligente

José Luís Azevedo, Bernardo Cunha, Pedro Fonseca,
Nuno Lau and Artur Pereira
IRIS/IEETA – DETI, Universidade de Aveiro

Outline

- Introduction
 - Tracked locomotion, legged locomotion, wheeled locomotion
- Wheeled mobile robots
 - Wheel types and wheel configurations
 - Differential drive, Ackerman steering, Tricycle drive, Synchro drive, Omnidirectional drive
- Kinematic models of some popular wheeled configurations
 - Differential drive
 - Tricycle drive
 - Omnidirectional drive

Mobile robot

- A combination of various physical and computational units (hardware and software)
- Organized in a set of sub-systems:
 - **Sensing**: measures properties of the robot environment
 - **Reasoning**: maps measurements into high-level action commands
 - **Control**: transforms high-level action commands into low-level actions
 - **Actuation**: transforms low-level action commands into physical actions
 - **Locomotion**: maps physical actions into movement, e.g, defines how the robot moves in its environment
 - **Communication**: provides communication with other robots, or with an external system

Locomotion



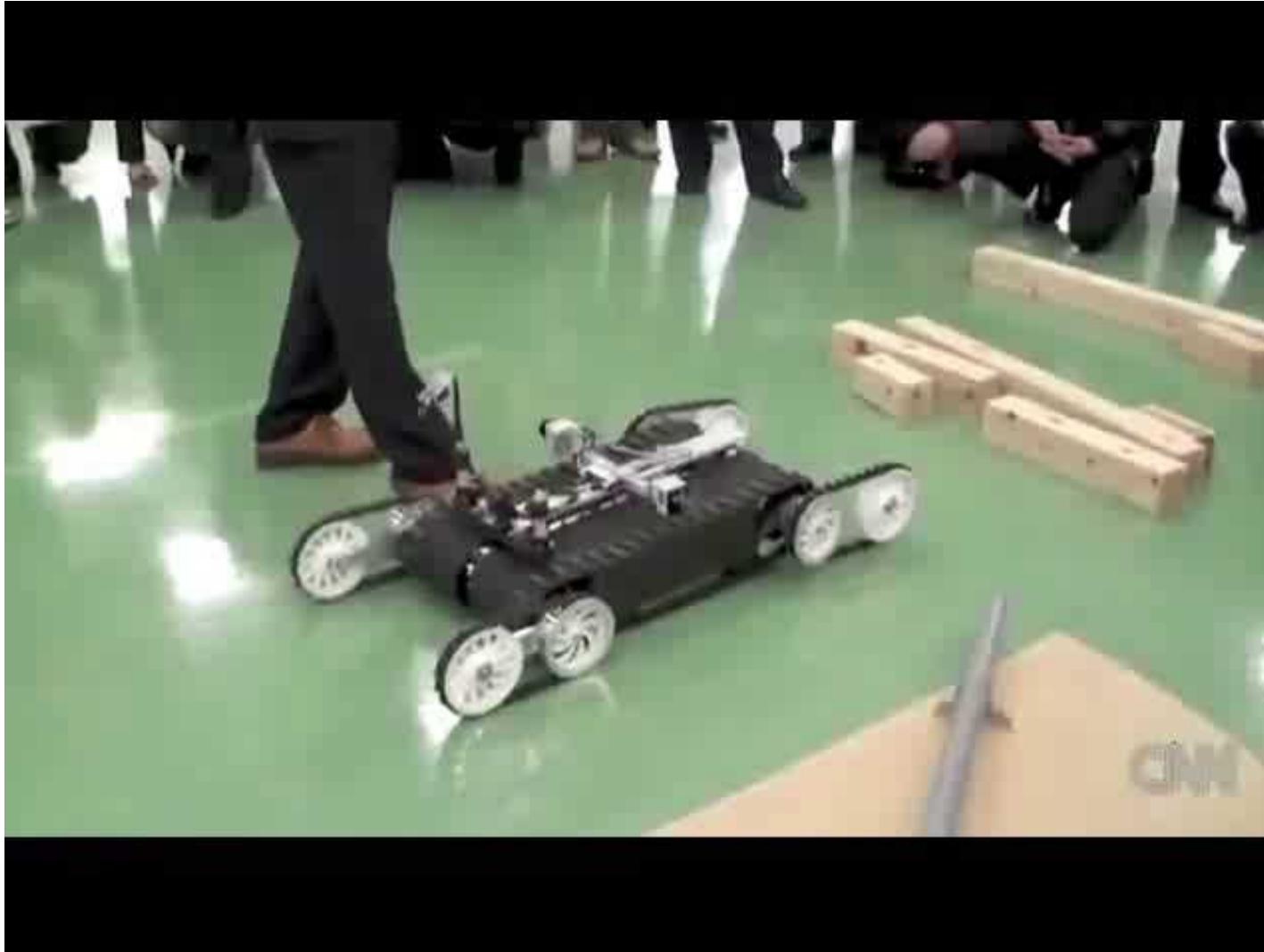
- The physical process that allows the robot to move in its environment
- Several solutions available:
 - Tracked locomotion
 - Legged locomotion
 - Wheeled locomotion

Tracked locomotion



- **Great traction power** - the track contact area with the ground is greater than the one provided by a wheel
- Locomotion system well suited for **robots that evolve in very rough terrain** (e.g., in natural disaster situations)
- **Change of direction is achieved by sliding the tracks**, which makes it very **difficult to use odometry** as a method of localization
- Requires a large amount of power to turn
- The robots that use this type of movement are **typically teleoperated**

Tracked locomotion

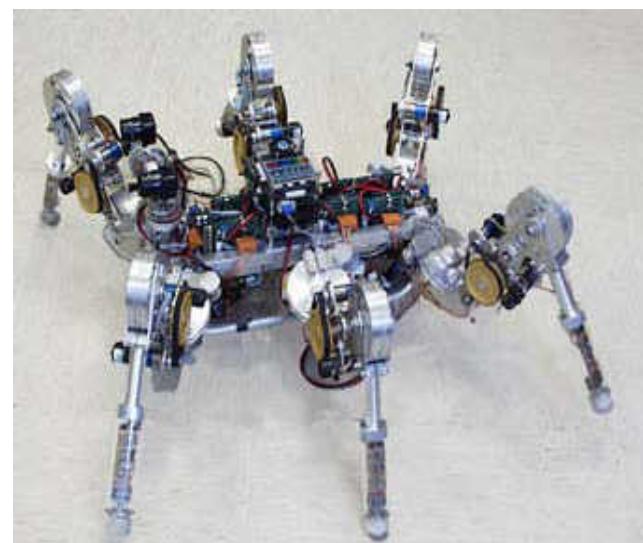
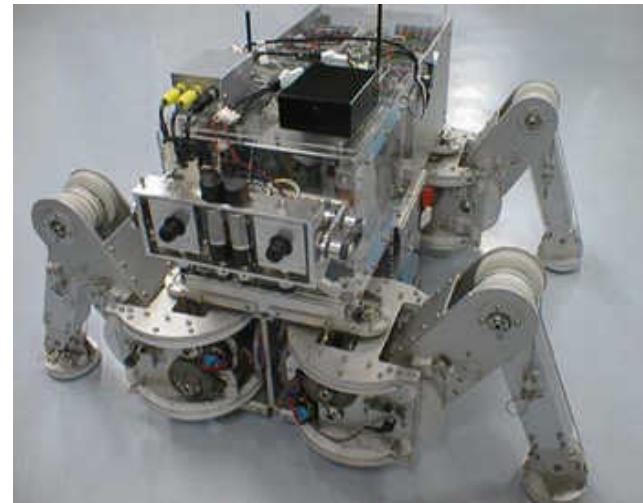


Legged locomotion

- Locomotion with legs is many times **based on living beings** (as those that move in difficult environments)
- The implementation of this type of locomotion system in robots is **complex**:
 - Mechanical complexity
 - Stability
 - Power consumption



Legged locomotion



Legged locomotion (AlphaDog – Boston Dynamics)



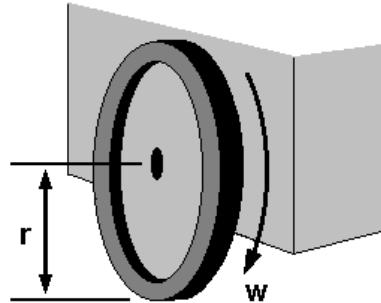
Wheeled locomotion



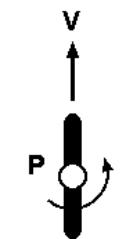
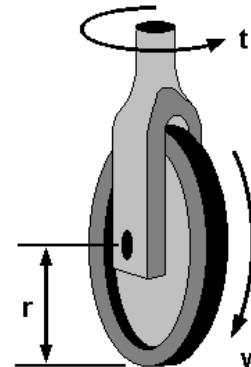
- The wheeled locomotion solution is the **most suitable for common applications**
 - rolling is very efficient!
- The **configuration and type of wheel** to use is **dependent on the application**
- Main constraint: **flat terrain** (or slightly irregular)
- Bigger wheels allow the robot to overcome bigger obstacles. However:
 - Motors with higher torque are needed (or gearboxes with higher reduction ratios, i.e., lower output speed for the same motor)

Wheel types

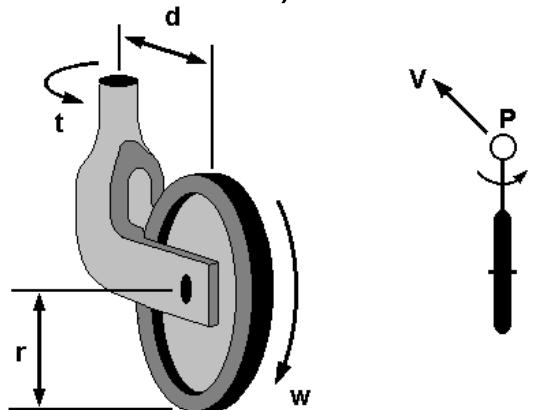
Standard wheel



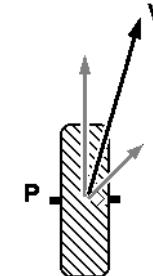
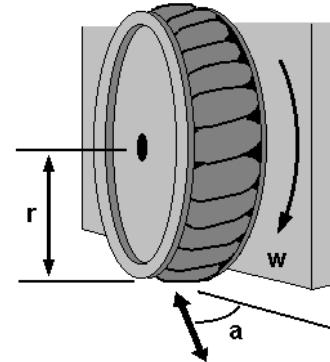
Steered standard wheel



Off-centered orientable wheel
(castor wheel)

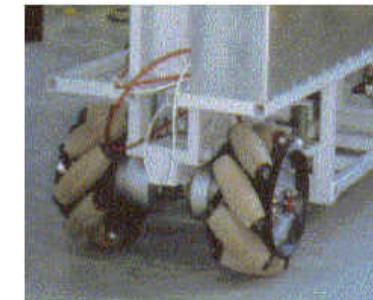
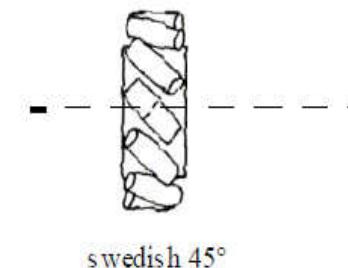
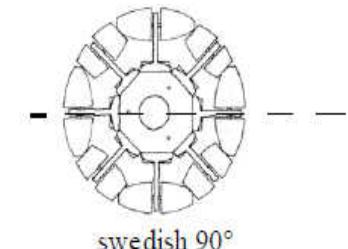
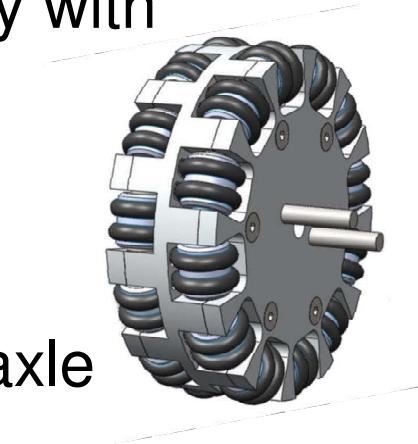


Swedish wheel (omnidirectional)



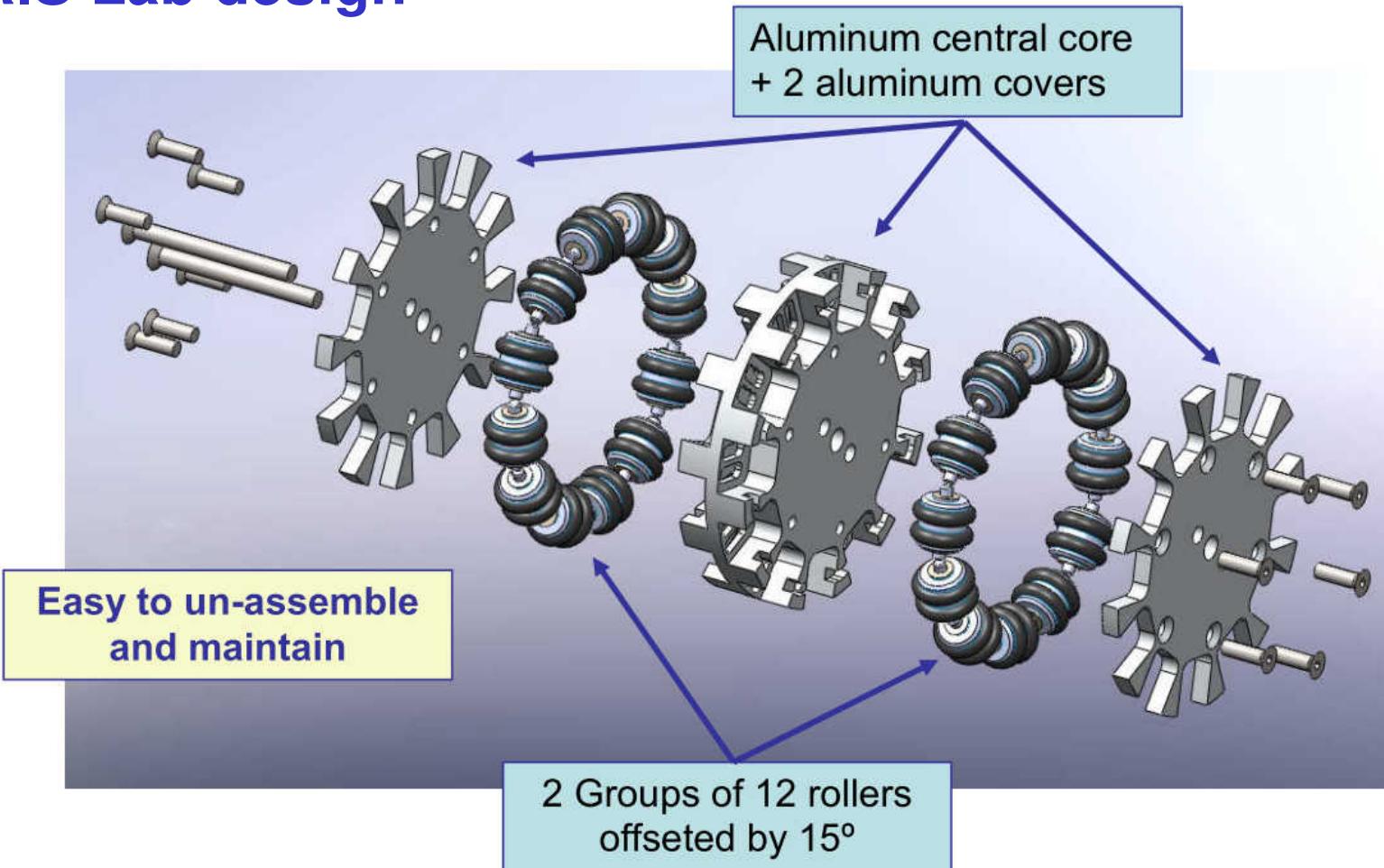
Wheel types – Swedish wheel

- Small rollers around the wheel circumference, with axes antiparallel to the main axis
- The wheel can be driven with full force, but will also slide laterally with very low friction
- Omnidirectional property
- Three degrees of freedom:
 - Rotation around the wheel axle (motorized)
 - Around the rollers
 - Around the contact point with the ground



Wheel types – Swedish wheel

- IRIS Lab design



Wheeled locomotion – static stability



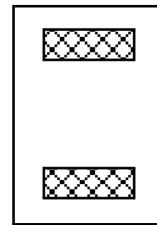
- **Two wheels**
 - Minimum number of wheels to achieve stability
 - **Center of mass must be below the axle** that links the wheels
- **Three wheels**
 - Stable configuration
 - **Center of mass must be inside the triangle formed by the ground contact points of the wheels**
- **Four wheels**
 - Stable configuration
 - **Requires a suspension system** to compensate for irregularities in the environment where the robot has to move
- **More than four wheels**
 - Configuration dependent

Wheel configurations¹

- 2 wheels



One steering wheel in the front and one traction wheel in the rear

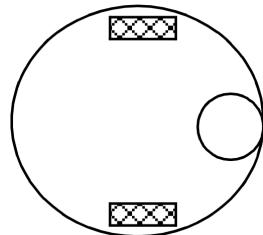


Two-wheel differential drive with the center of mass below the axle

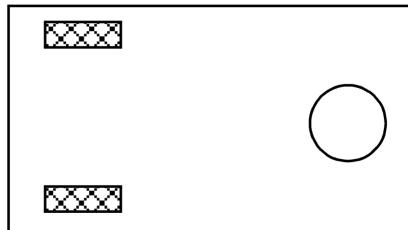
1) From: *R. Siegwart, I. Nourbakhsh*

Wheel configurations

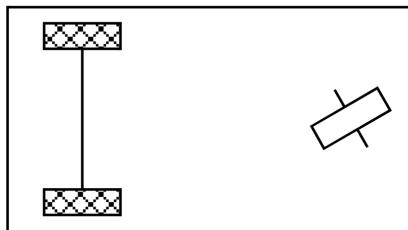
- 3 wheels



**Two-wheel centered differential drive
with a third point of contact**



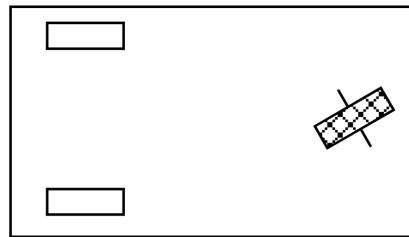
**Two independently driven wheels in the
rear/front, one steered free wheel
(unpowered) in the front/rear**



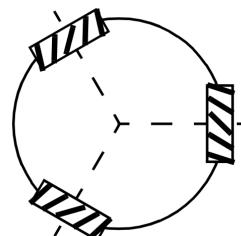
**Two connected traction wheels
(differential gear) in rear, one steered
free wheel in front**

Wheel configurations

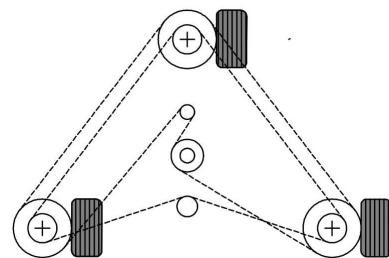
- 3 wheels



Two free wheels in rear, one steered traction wheel in front



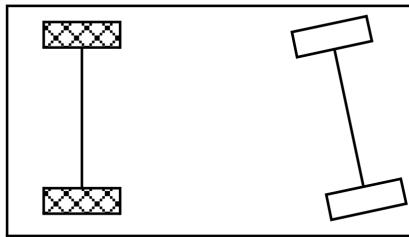
Three motorized Swedish or spherical wheels arranged in a triangle; omnidirectional movement is possible



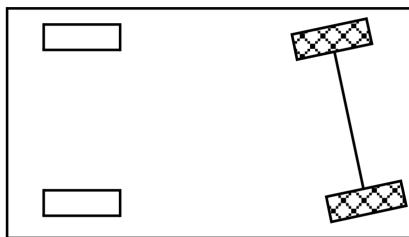
Three synchronously motorized and steered wheels; the chassis orientation is not controllable

Wheel configurations

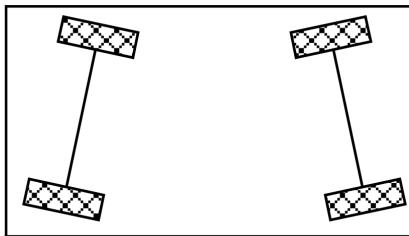
- 4 wheels



Two motorized wheels in the rear, two steered wheels in the front; steering has to be different for the two wheels to avoid slipping/skidding.



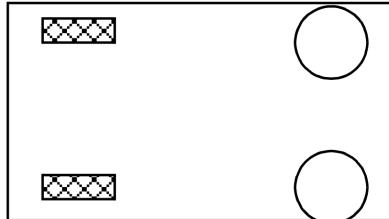
Two motorized and steered wheels in the front, two free wheels in the rear; steering has to be different for the two wheels to avoid slipping/skidding.



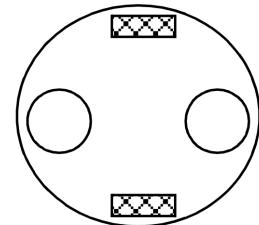
Four steered and motorized wheels

Wheel configurations

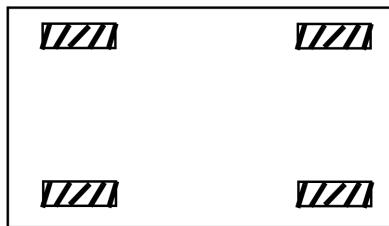
- 4 wheels



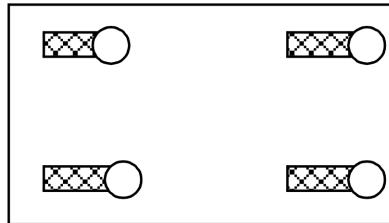
Two traction wheels (differential) in rear/front, two omnidirectional wheels in the front/rear



Two-wheel differential drive with two additional points of contact



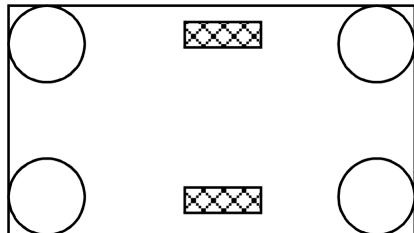
Four omnidirectional wheels



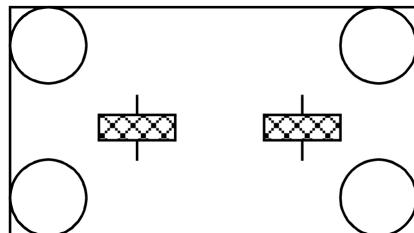
Four motorized and steered castor wheels

Wheel configurations

- 6 wheels



Two traction wheels (differential) in center, one omnidirectional wheel at each corner



Two motorized and steered wheels aligned in center, one omnidirectional wheel at each corner

Non-standard configurations

SHRIMP (EPFL)

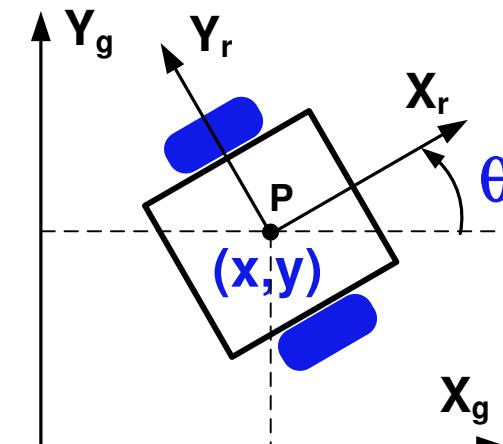


- **Locomotion**
 - The process that causes the movement of the robot
 - In order to produce a motion, forces must be applied to the robot
- **Dynamics**
 - The study of motion, in which forces are modeled
- **Kinematics**
 - Modeling the motion without considering the forces that cause the object to move

Local and global reference frames

- **global** reference frame: {X_g, Y_g}
- **local** (robot) reference frame: {X_r, Y_r}
- Orthogonal rotation matrix:

$$R(\theta) = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$$\xi R = [x \ y \ \theta]^T$$

- Mapping velocities **from global reference frame** to robot reference frame:

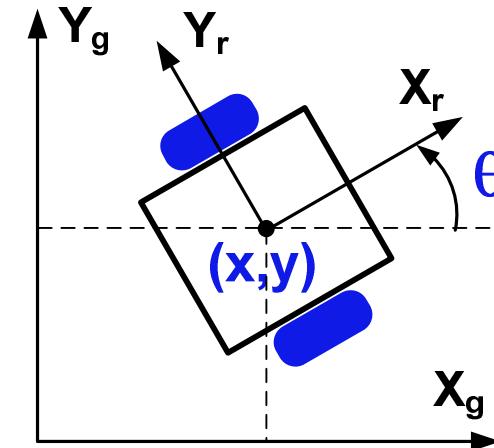
$$\dot{\xi} R = R(\theta) \dot{\xi} G = R(\theta) \begin{bmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix}^T \quad \dot{\xi} R = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix}^T$$

(\dot{x} – linear velocity along X_g, \dot{y} – linear velocity along Y_g, $\dot{\theta}$ – angular velocity)

Local and global reference frames

- Inverse of the orthogonal rotation matrix:

$$R(\theta)^{-1} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



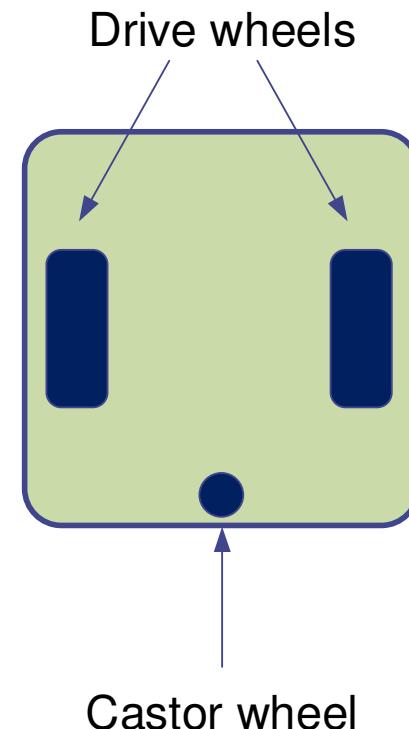
- Mapping velocities from robot reference frame to global reference frame:

$$\dot{\xi}_G = R(\theta)^{-1} \dot{\xi}_R = R(\theta)^{-1} [V_x \ V_y \ \dot{\theta}]^T \quad \dot{\xi}_G = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} [V_x \ V_y \ \dot{\theta}]^T$$

(V_x – linear velocity along X_r , V_y – linear velocity along Y_r , $\dot{\theta}$ – angular velocity)

Differential drive

- **Common configuration:**
 - 2 active independent drive wheels
 - 1 or 2 passive castor wheels
- Robot follows a **trajectory** which is **defined by the speed of each wheel**
- Trajectory is **sensitive to differences in the relative velocity of the two wheels**
 - caused by asymmetries in motors and/or wheels
 - a small error results in a path different from that intended
- **Easy mechanical implementation**



Differential drive – kinematics

ω_R – angular velocity, right wheel

ω_L – angular velocity, left wheel

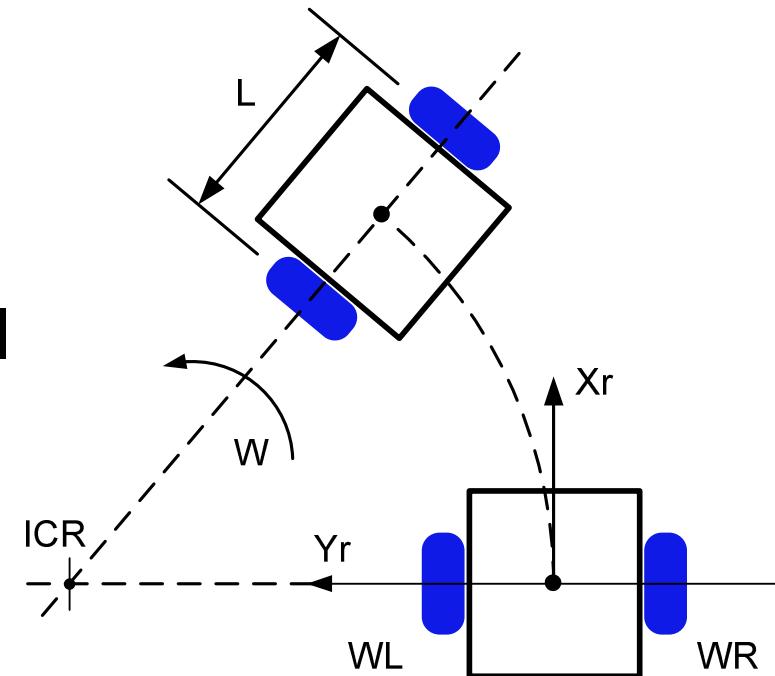
v_R – linear velocity, right wheel

v_L – linear velocity, left wheel

ω – angular velocity of the robot about ICR

r – wheel radius

L – distance between wheels



Differential drive – kinematics

$$V_R(t) = W_R(t) \times r$$

$$V_L(t) = W_L(t) \times r$$

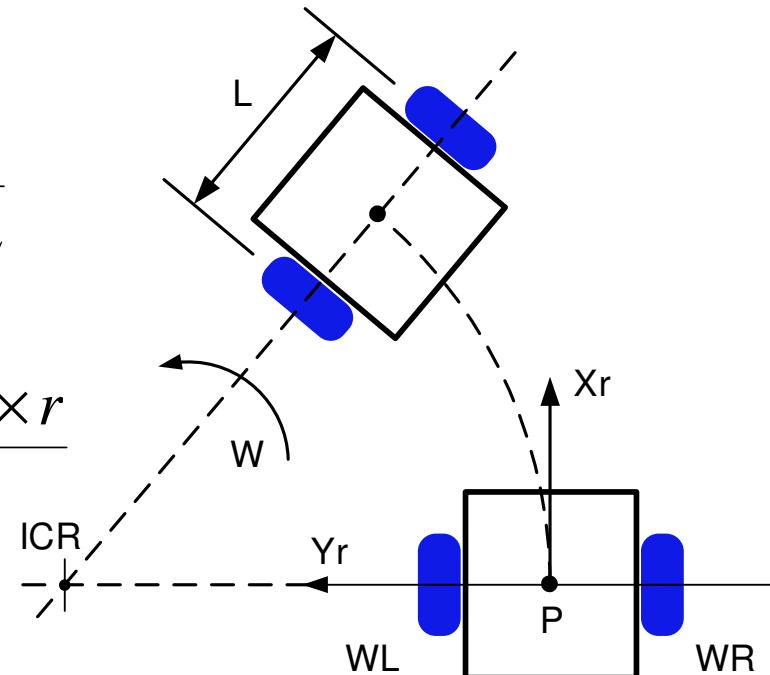
$$V_X(t) = \frac{V_R(t)}{2} + \frac{V_L(t)}{2} = W_R(t) \frac{r}{2} + W_L(t) \frac{r}{2}$$

$$V_Y(t) = 0$$

$$W(t) = \frac{V_R(t)}{L} - \frac{V_L(t)}{L} = \frac{W_R(t) \times r - W_L(t) \times r}{L}$$

Kinematic model in local frame

$$\begin{bmatrix} V_X(t) \\ V_Y(t) \\ W(t) \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ 0 & 0 \\ -r/L & r/L \end{bmatrix} \begin{bmatrix} W_L(t) \\ W_R(t) \end{bmatrix}$$

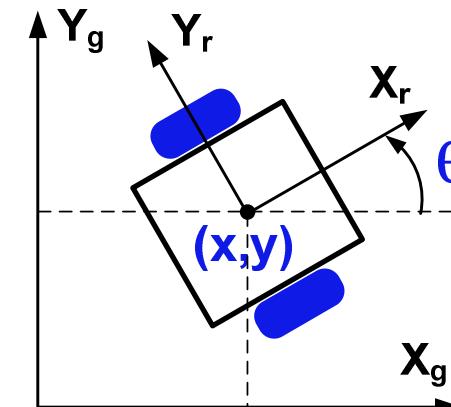


Differential drive – kinematics

Kinematic model in world frame

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos \theta(t) & -\sin \theta(t) & 0 \\ \sin \theta(t) & \cos \theta(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x(t) \\ 0 \\ W(t) \end{bmatrix}$$

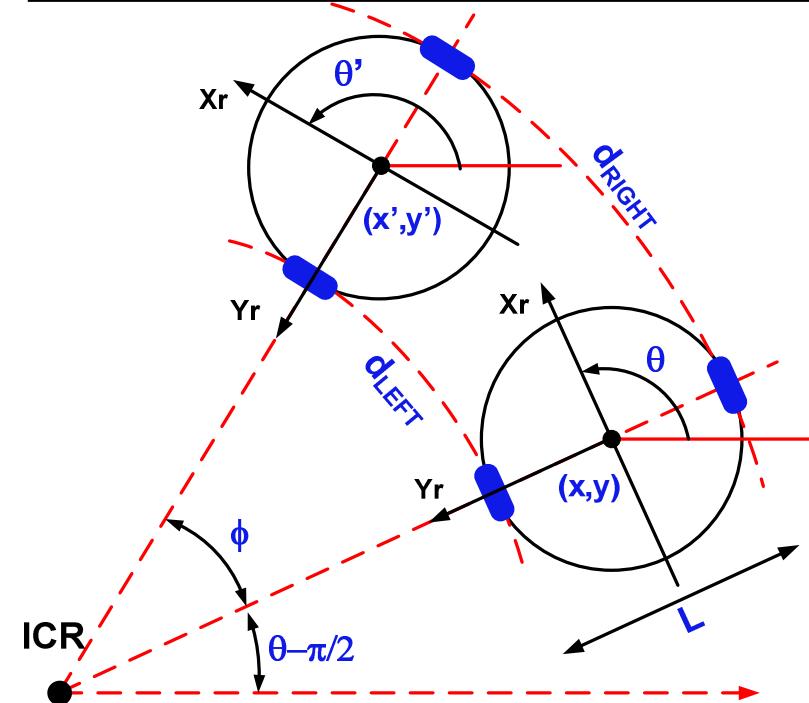
$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos \theta(t) & 0 \\ \sin \theta(t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V(t) \\ W(t) \end{bmatrix}$$



Differential drive – position estimation

- (x, y, θ) – pose (position and orientation) of the robot in the **world frame**
- Supposing the robot's pose is (x, y, θ) , the **position estimation** consists in finding (x', y', θ') given:
 - d_{RIGHT} - distance travelled by the right wheel
 - d_{LEFT} - distance travelled by the left wheel
 - d_{RIGHT} and d_{LEFT} measured by wheel encoders

Robot is moving counter-clockwise



L - distance between robot wheels

Differential drive – position estimation

- Over a small time period, the robot's motion can be approximated by an arc

$$d_{CENTER} = \frac{d_{RIGHT} + d_{LEFT}}{2}$$

$$\phi = \frac{dist}{R}$$

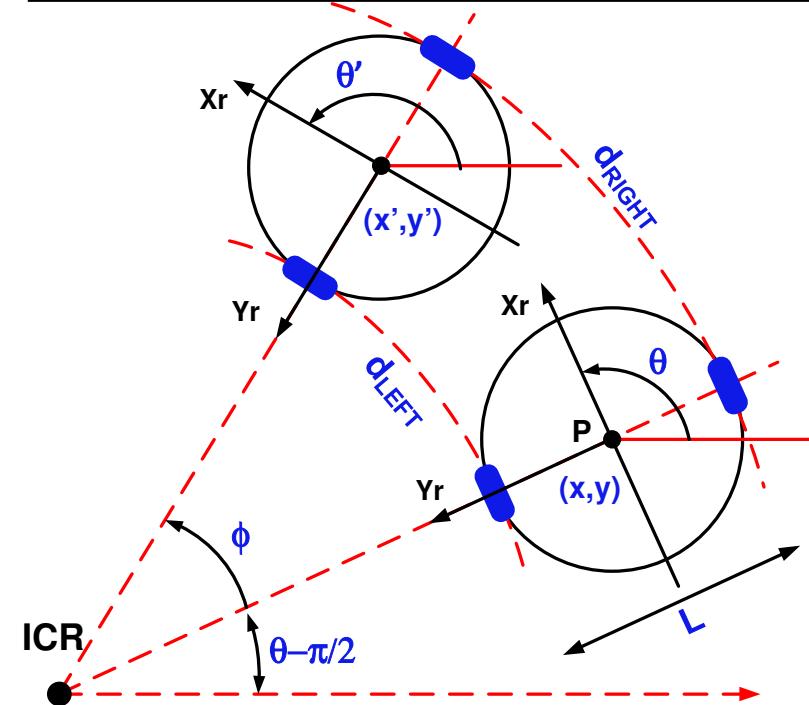
$$\phi \times R_{RIGHT} = d_{RIGHT}$$

$$\phi \times R_{LEFT} = d_{LEFT}$$

$$\phi = \frac{d_{RIGHT} - d_{LEFT}}{L}$$

$$(R_{RIGHT} - R_{LEFT} = L)$$

Robot is moving counter-clockwise



ICR: Instantaneous Center of Rotation

Differential drive – position estimation

$$d_{CENTER} = \frac{d_{RIGHT} + d_{LEFT}}{2}$$

$$\phi = \frac{d_{RIGHT} - d_{LEFT}}{L}$$

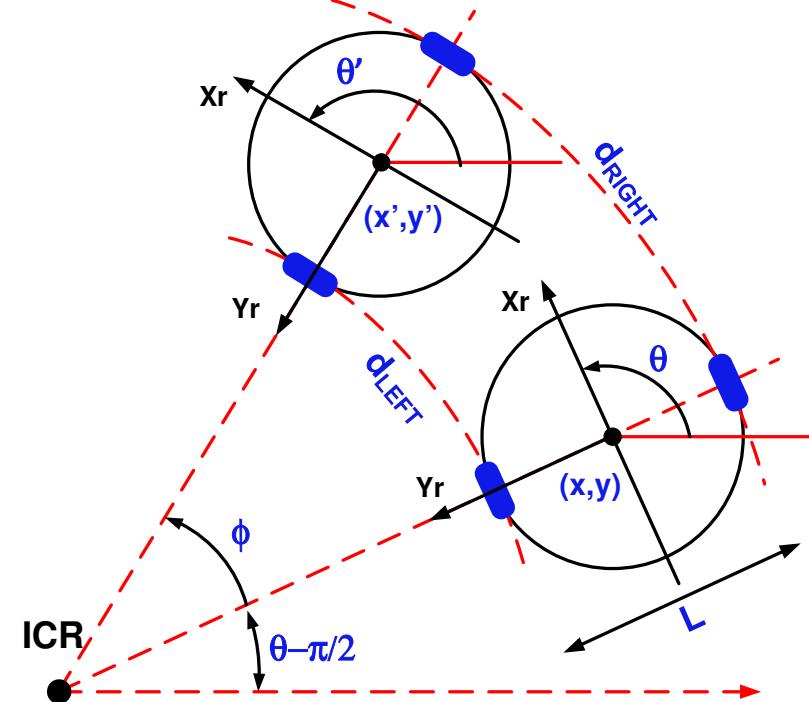
- For small displacements, such that $\sin(\phi) \approx \phi$ and $\cos(\phi) \approx 1$:

$$x' = x + d_{CENTER} \times \cos(\theta)$$

$$y' = y + d_{CENTER} \times \sin(\theta)$$

$$\theta' = \theta + \phi$$

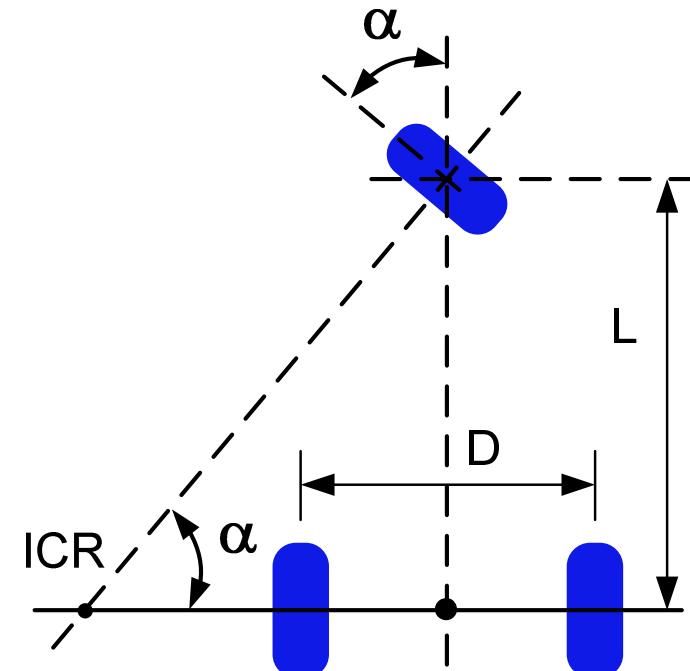
Robot is moving counter-clockwise



ICR: Instantaneous Center of Rotation

Tricycle drive

- **Three wheels**: two rear wheels and one (steering) front wheel
- Two possible configurations of traction:
 - **Front wheel is passive** - the two rear wheels are driving wheels (must use differential gear)
 - **Driving wheel on the front** (rear wheels are passive) – easier to implement
- **Main problems** of the front wheel drive configuration:
 - When going uphill, the driving wheel may **lose traction** due to the displacement of the center of mass
 - The traction **contact area** with the ground is half of the rear wheel drive configuration

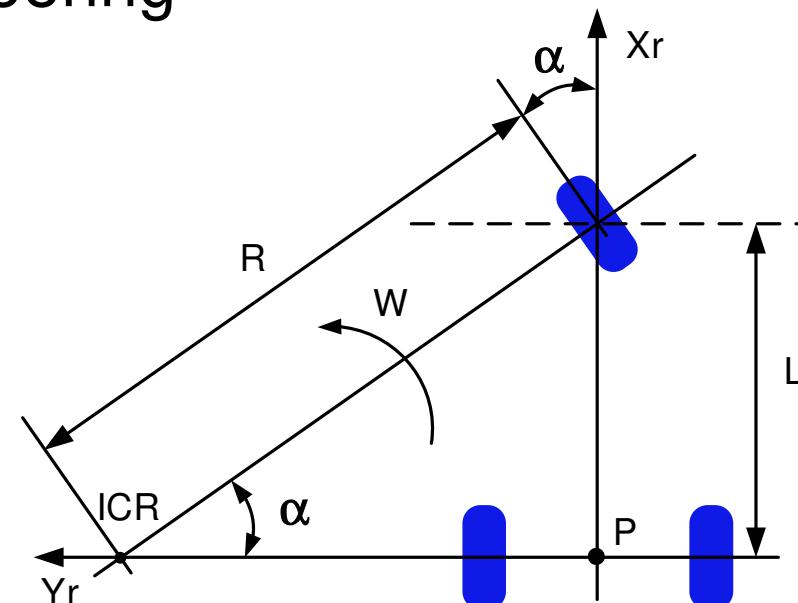


Tricycle drive – kinematics

V_s – linear velocity of the steering wheel

W_s – angular velocity of the steering wheel

r – steering wheel radius



W – angular velocity of the robot about ICR

α – steering angle

Tricycle drive – kinematics

$$V_s = W_s \times r$$

(linear velocity of the steering wheel)

$$W_s = \frac{V_s}{r}$$

(angular velocity of the steering wheel)

$$R = \frac{L}{\sin \alpha}$$

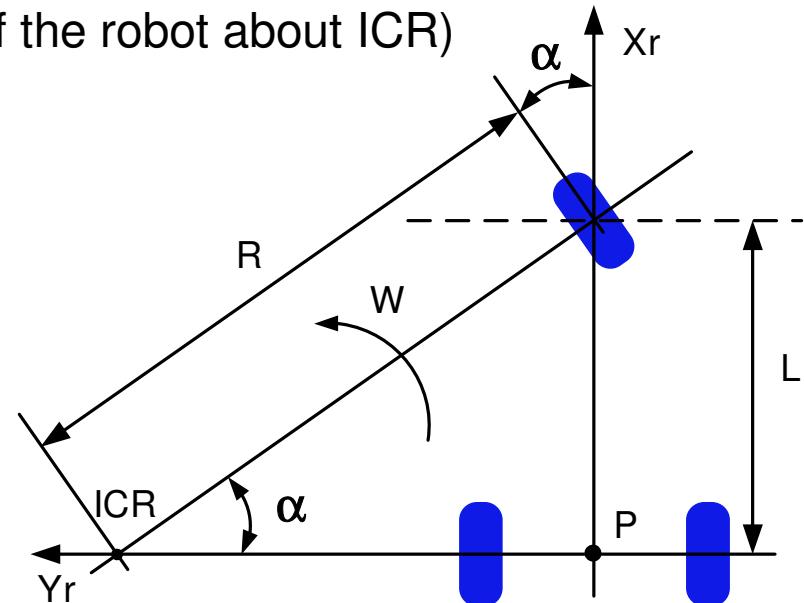
$$W = \frac{V_s}{R} = \frac{V_s \times \sin \alpha}{L}$$
 (angular velocity of the robot about ICR)

Kinematic model in local frame

$$VX(t) = V_s(t) \times \cos \alpha(t)$$

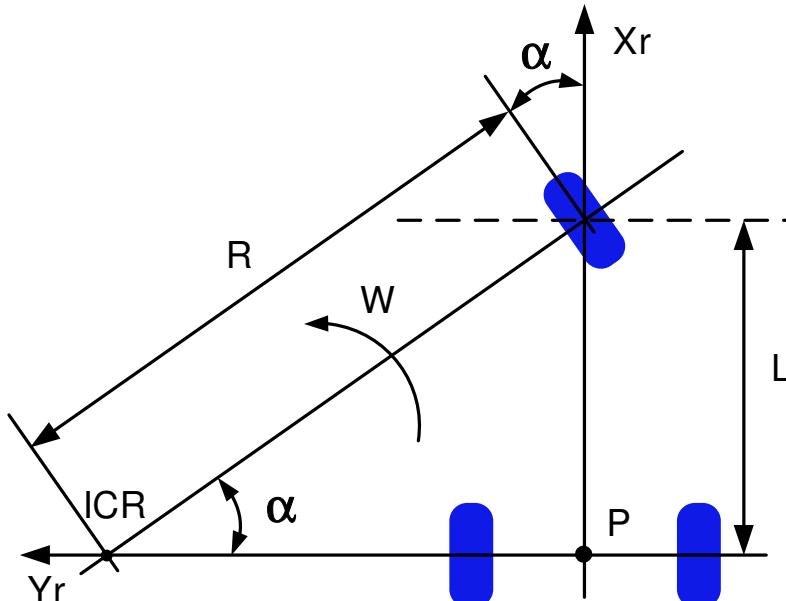
$$VY(t) = 0$$

$$W(t) = \frac{V_s(t)}{L} \times \sin \alpha(t)$$



Tricycle drive – kinematics

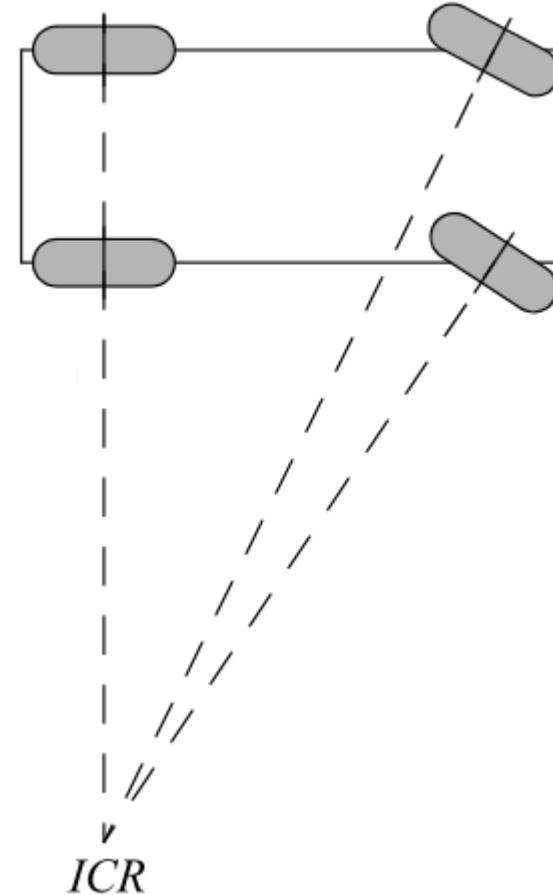
Kinematic model in world frame



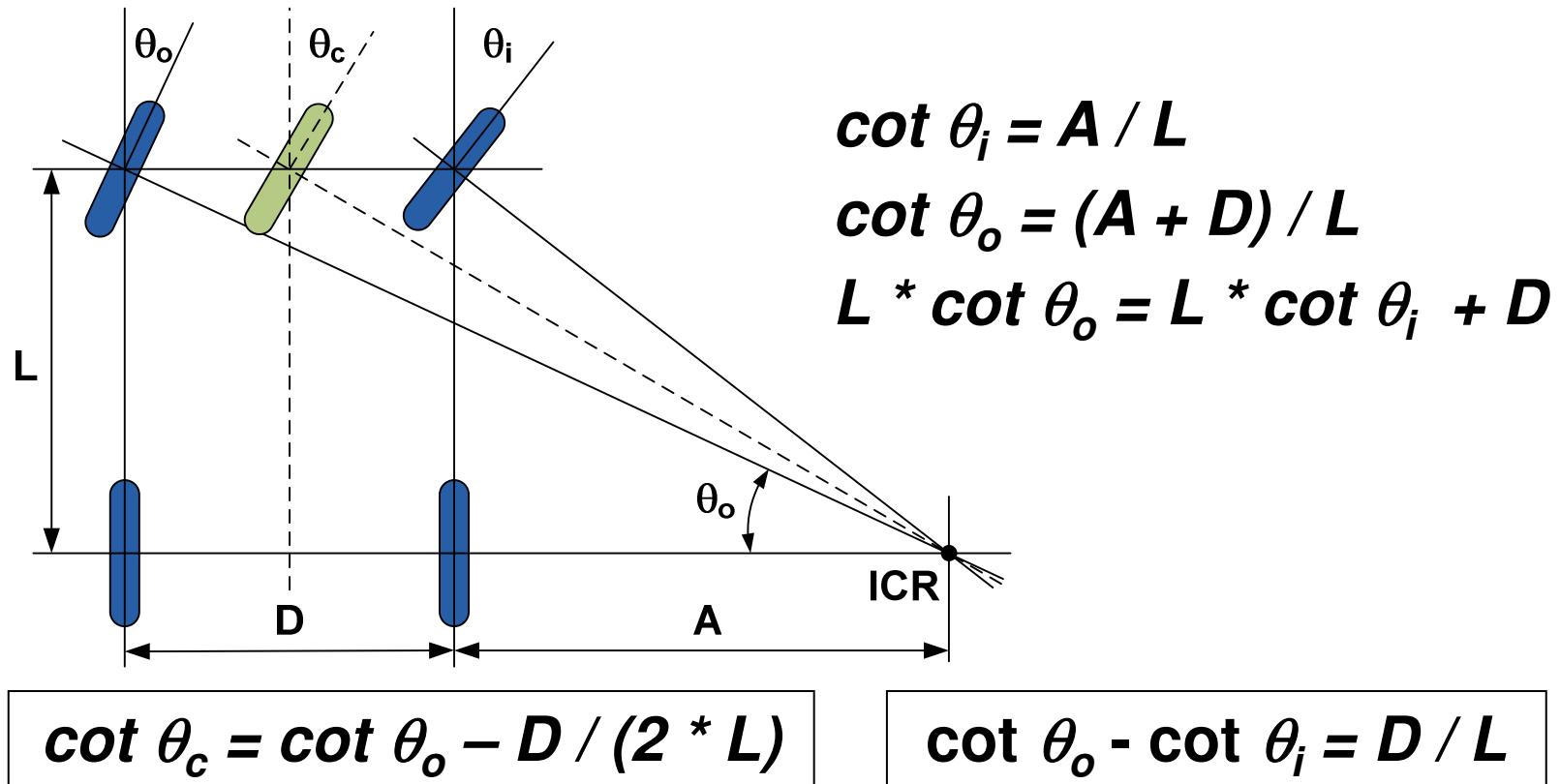
$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos \theta(t) & -\sin \theta(t) & 0 \\ \sin \theta(t) & \cos \theta(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_s(t) \times \cos \alpha(t) \\ 0 \\ \frac{V_s(t)}{L} \times \sin \alpha(t) \end{bmatrix}$$

Ackerman steering

- Generally the **method of choice for outdoor autonomous robots**
- The **inside front wheel is turned slightly more than the outside wheel** (reduces tire slippage)
- The extension of the axis of all four wheels intersects a common point ICR
- 4 or 3 wheel system support rear and/or front traction
- A **differential gear must be used in the traction axel** (unless a single motorized wheel is used in that axel)

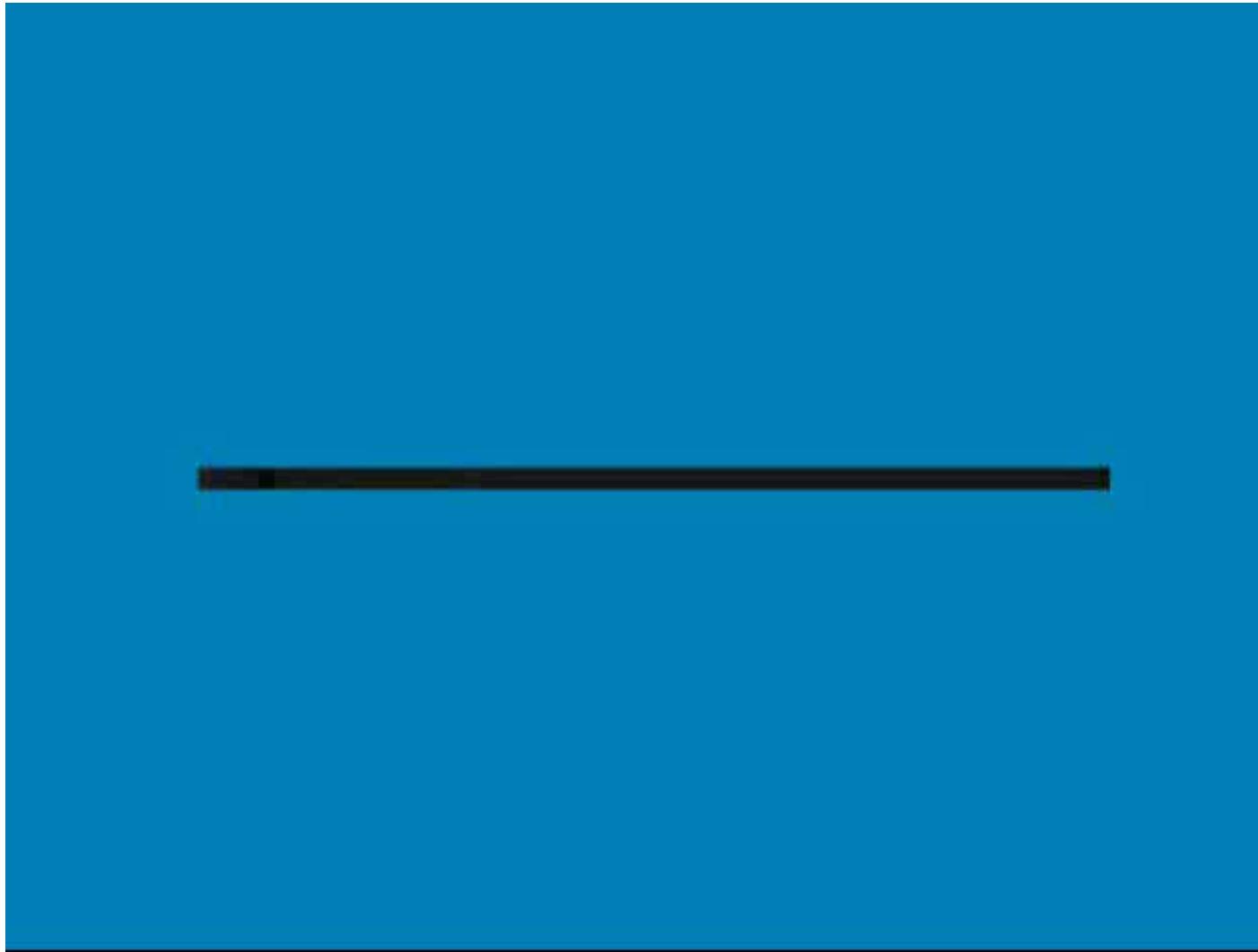


Ackerman steering

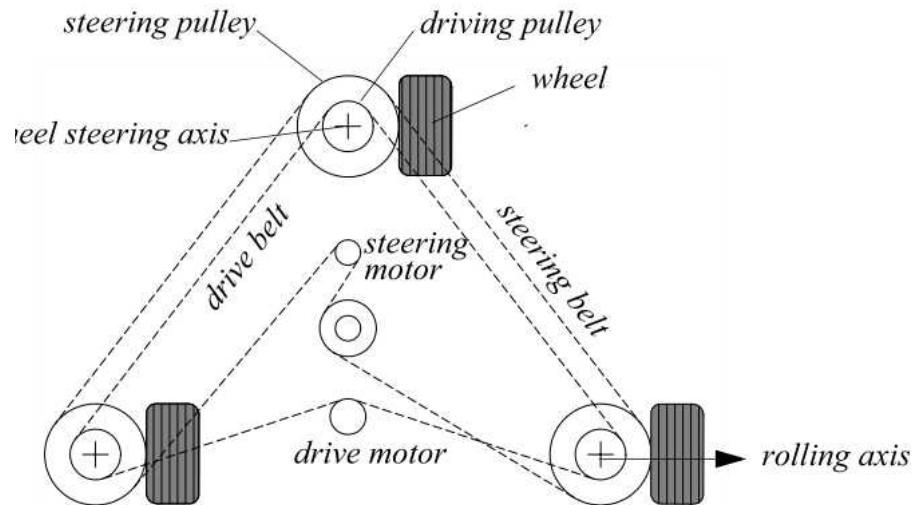
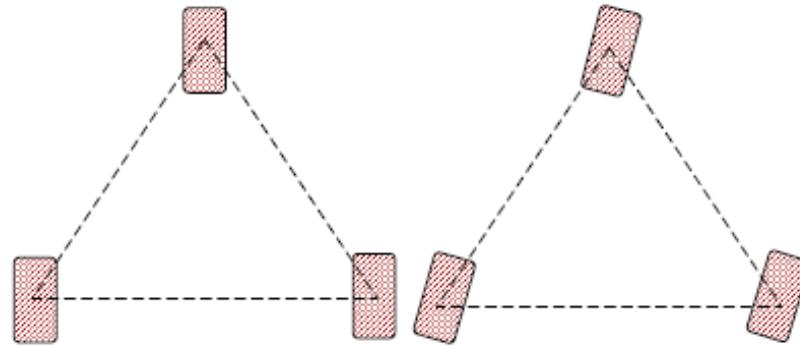


- Axis of all wheels intersects a common point ICR
- Kinematic model: tricycle drive

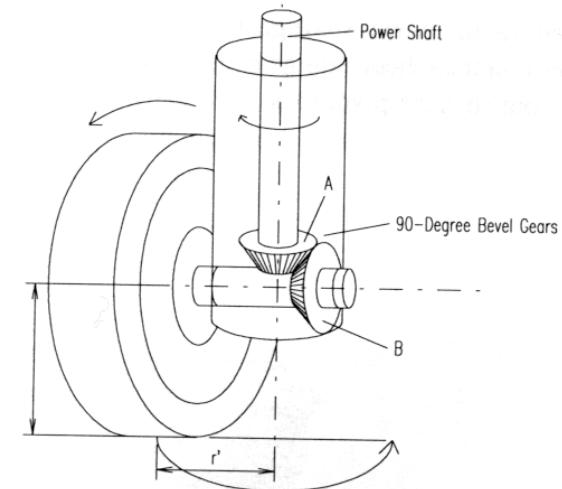
Ackerman steering (ROTA robot)



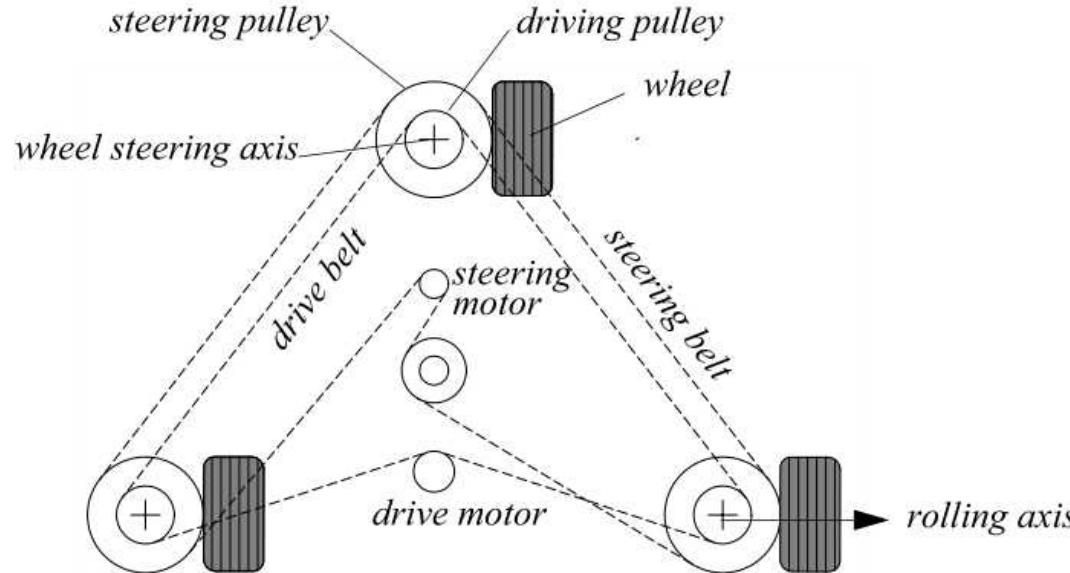
Synchro drive



- Three (or more) wheels
- Two motors:
 - Translation motor sets the speed of all three wheels together
 - Steering motor turns all the wheels together about each of their individual vertical steering axes

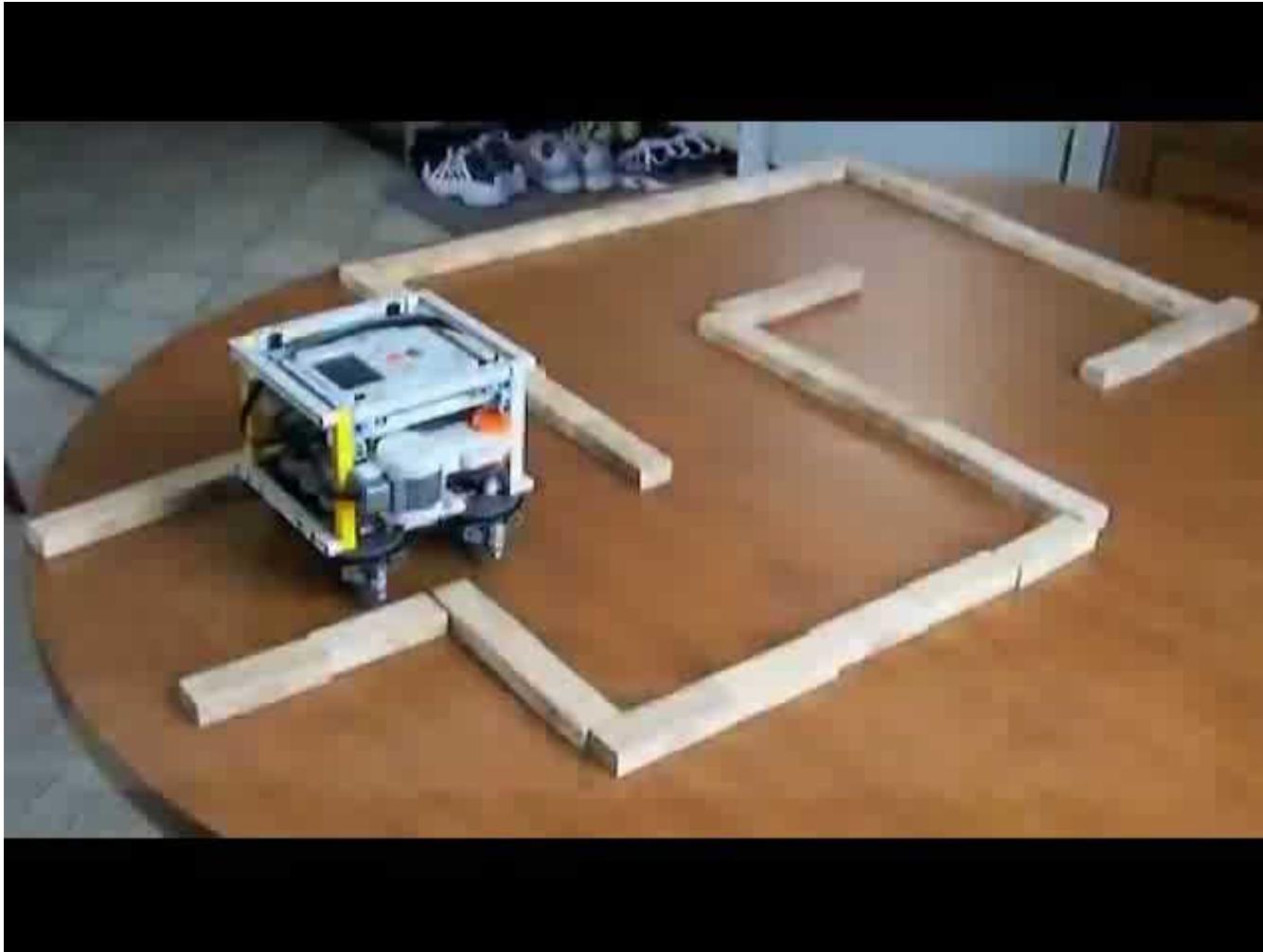


Synchro drive



- The robot **can move in any direction**
- The robot can always reorient its wheels and **move along a new trajectory without changing its footprint**
- However, the **orientation of the chassis is not controllable** (since the wheels are being steered with respect to the robot chassis)

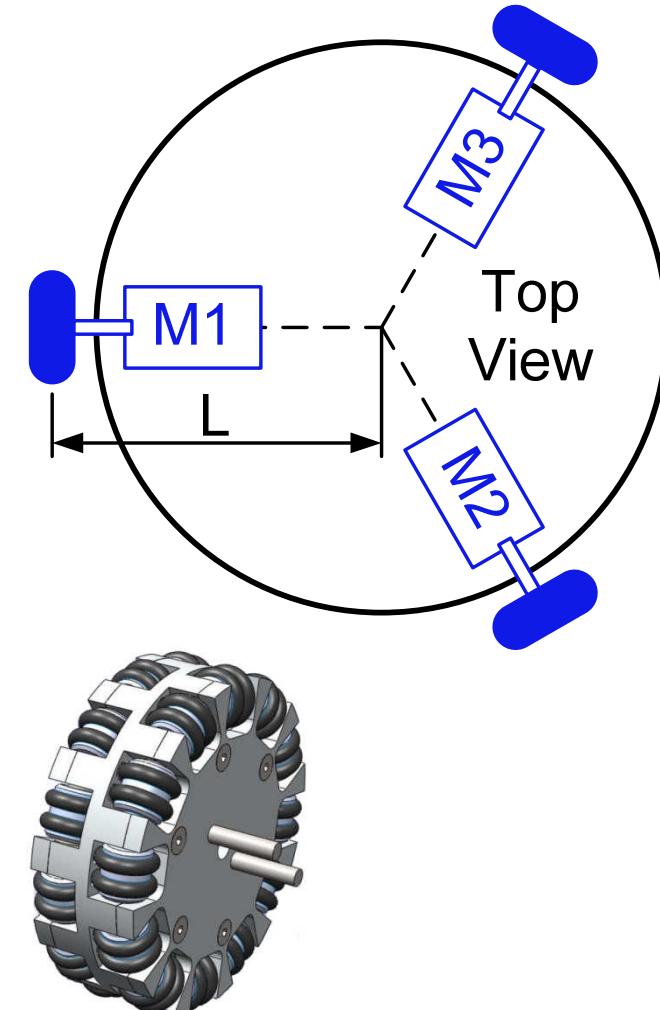
Synchro drive (LEGO)



<http://y2u.be/MFxjlthqXVs>

Omnidirectional drive

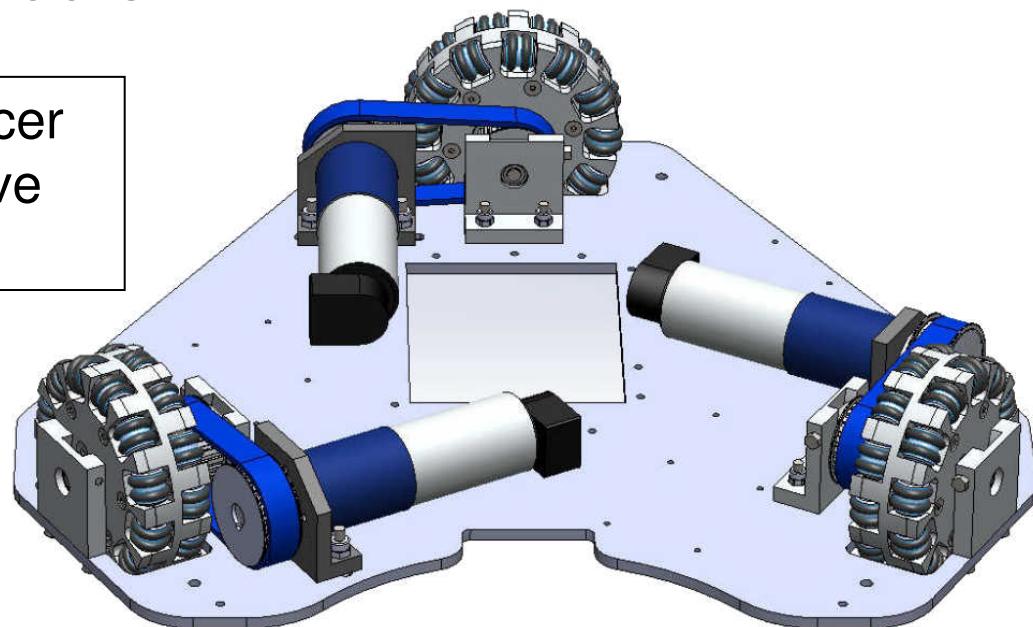
- Uses **Swedish wheels**
- **Each wheel has one independent drive motor**
- Allows **movement in any direction** by setting appropriate speeds in each of the three motors
- Allows complex movements (for instance translation combined with rotation)
- Three wheels configuration:
 - the wheels are spaced 120°



Omnidirectional drive

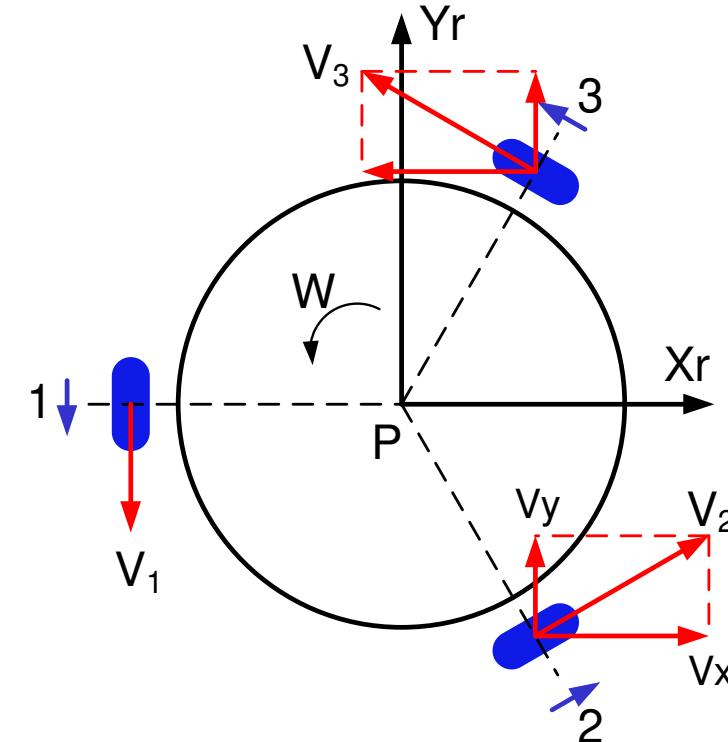
- Allows the generation of complex movements, such as to go straight changing, at the same time, the robot orientation
- **Excellent maneuverability**
- 4-wheel configuration: greater traction but more sensitive to uneven floors

CAMBADA (IRIS Lab) soccer robot – omnidirectional drive structure



Omnidirectional drive – kinematics

- The translation velocities of the wheels, V_1 , V_2 and V_3 , determine the global velocity of the robot on the environment
- The translation velocity of the wheel hub "i" (V_i) can be divided in two parts:
 - pure translation of the robot
 - pure rotation of the robot



$$V_i = V_{transl, i} + V_{rot}$$

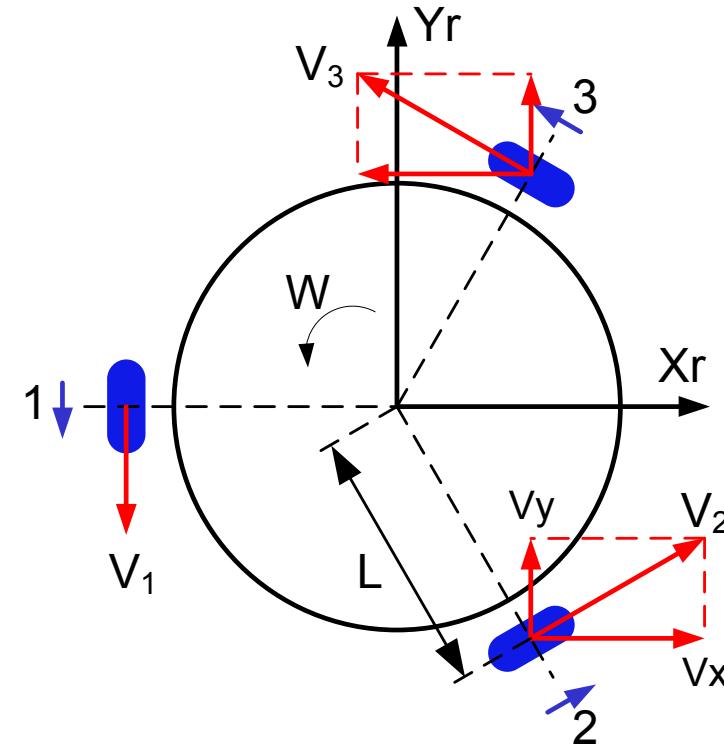
Omnidirectional drive – kinematics

- When the robot performs a pure rotation, the hub "i" velocity becomes

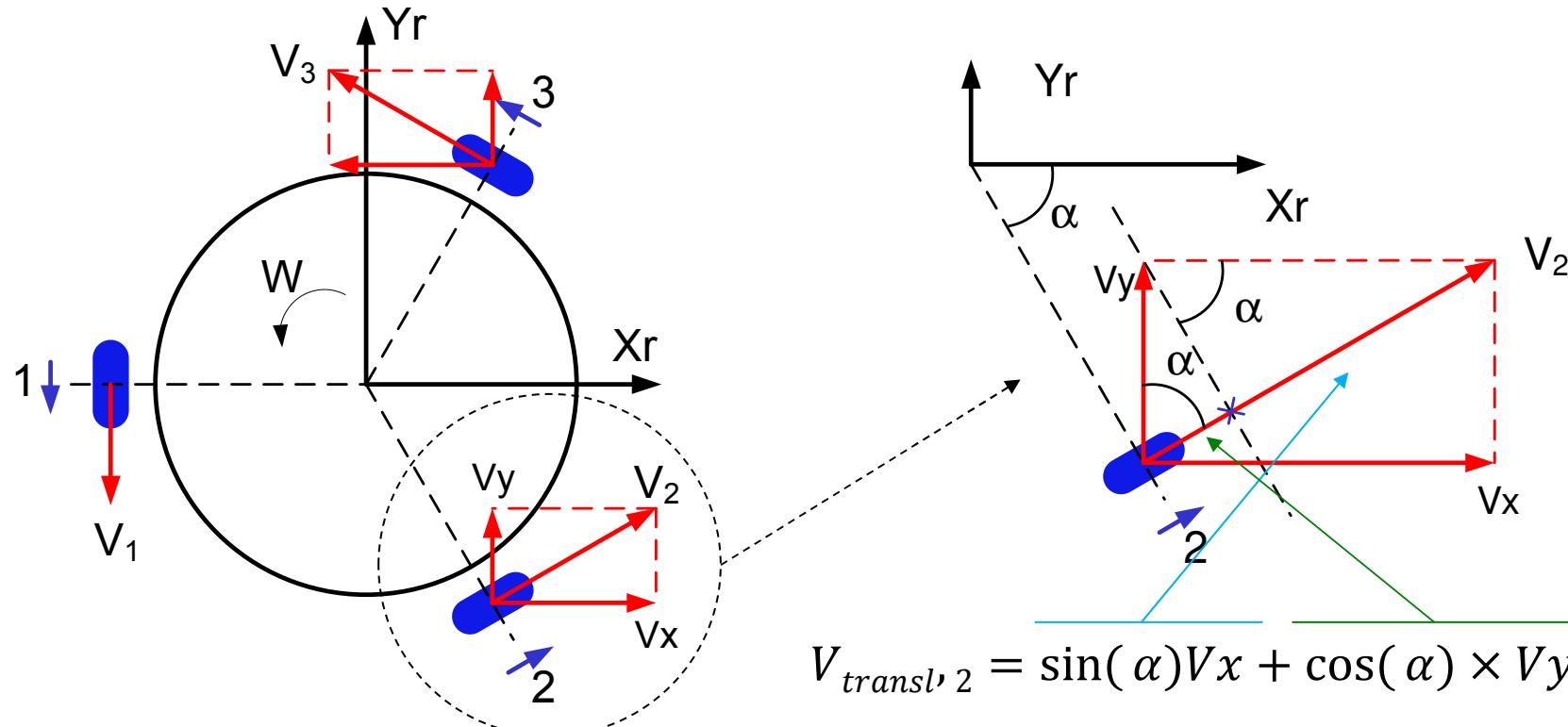
$$V_i = LW$$

where:

- L: is the distance from the geometric center of the robot to the wheel
- W: angular velocity of the robot



Omnidirectional drive – kinematics



$$V_{transl, 3} = -\sin(\alpha)Vx + \cos(\alpha)Vy$$

$$V_{transl, 1} = \sin(0)Vx - \cos(0)Vy = -Vy$$

Omnidirectional drive – kinematics

- Taking hub 1 as reference, the hub angles are:

$$\alpha_1 = 0^\circ, \alpha_2 = 120^\circ, \alpha_3 = 240^\circ$$

- The pure translation velocity at wheel hub "i" can then be generalized as:

$$V_{transl,i} = \sin(\alpha_i)Vx - \cos(\alpha_i)Vy$$

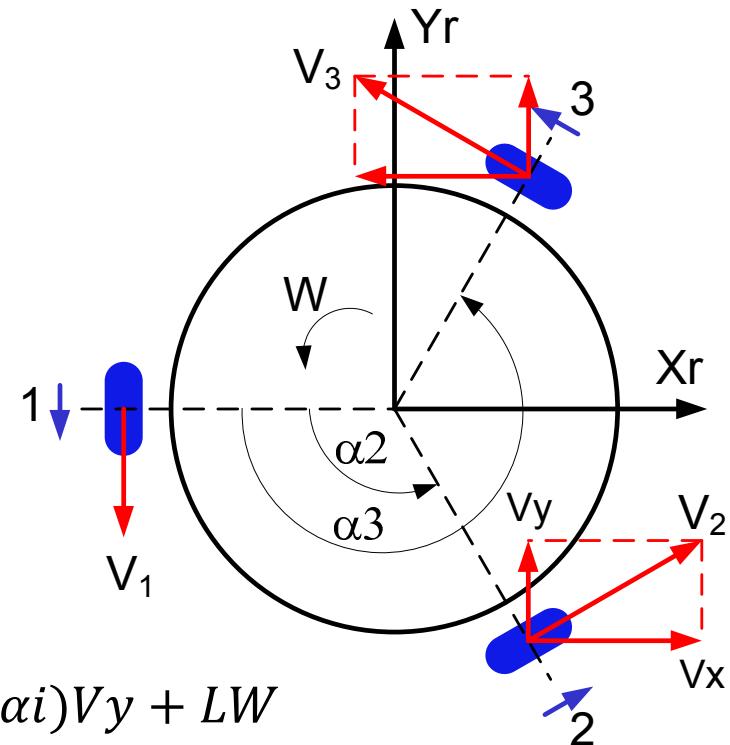
- And V_i becomes:

$$V_i = V_{transl,i} + V_{rot} = \sin(\alpha_i)Vx - \cos(\alpha_i)Vy + LW$$

- But, $V_i = r \cdot W_i$, (r is the wheel radius and W_i the wheel angular velocity)

$$rW_i = \sin(\alpha_i)Vx - \cos(\alpha_i)Vy + LW$$

$$W_i = (\sin(\alpha_i)Vx - \cos(\alpha_i)Vy + LW) / r$$



Omnidirectional drive – kinematics

- We can then write:

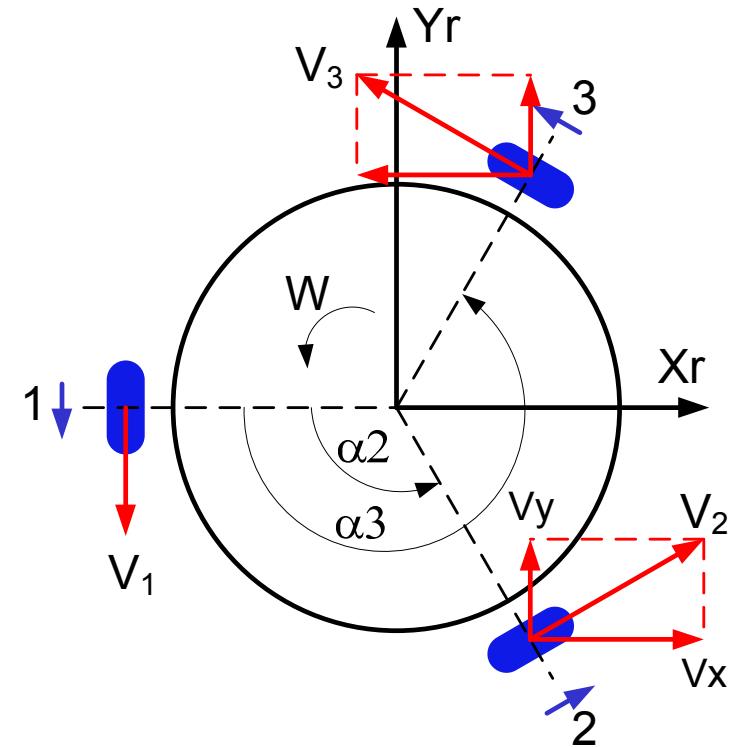
$$W_1 = \frac{1}{r}(-V_y + LW)$$

$$W_2 = \frac{1}{r}(\frac{\sqrt{3}}{2}V_x + 0.5V_y + LW)$$

$$W_3 = \frac{1}{r}(-\frac{\sqrt{3}}{2}V_x + 0.5V_y + LW)$$

- Solving for V_x , V_y and W :

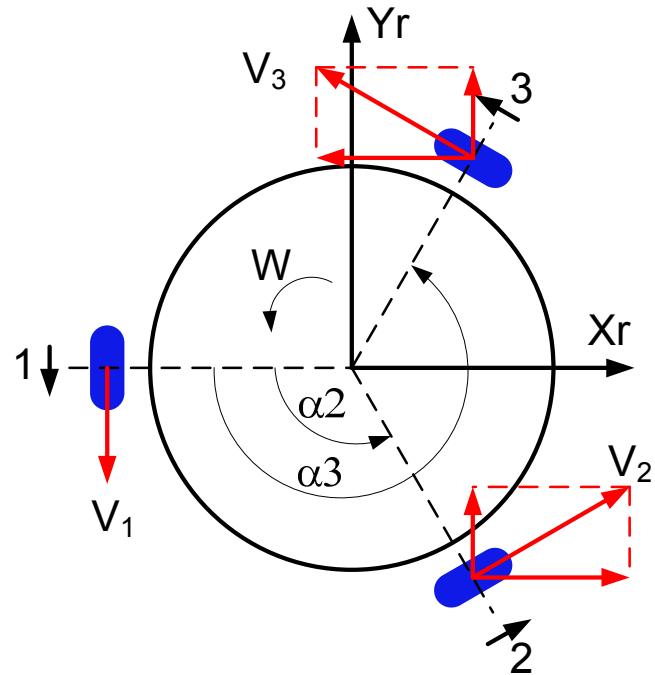
$$\begin{bmatrix} V_x \\ V_y \\ W \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{\sqrt{3}} \times r & -\frac{1}{\sqrt{3}} \times r \\ -\frac{2}{3}r & \frac{1}{3}r & \frac{1}{3}r \\ \frac{r}{3L} & \frac{r}{3L} & \frac{r}{3L} \end{bmatrix} \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix}$$



Kinematic model in local frame

Omnidirectional drive - kinematics

Kinematic model in global frame



$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ W \end{bmatrix}$$

Omnidirectional drive (CAMBADA)



<http://y2u.be/PXq89EONEz0>

Computer Vision basics: Digital Cameras Parameters

Institute of Electronics and Telematics Engineering of Aveiro
Dep. of Electronics, Telecommunications and Informatics
Universidade de Aveiro
Portugal

Outline

- Broad definition of computer vision & artificial vision areas
- Some basic definitions
- The typical image processing pipeline
- Image parameters (intrinsic and adjustable)
- Basic optics
- Camera parameters and perspective views
- Examples of simple image processing tasks

Broad definition

- Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images.
- Computer vision applications are increasing:
 - surveillance;
 - machine inspection;
 - medicine;
 - **robotics**;
 - entertainment;
 - media.
- One of the goals: make computer vision converge towards human vision (or make it even better).

Artificial vision areas

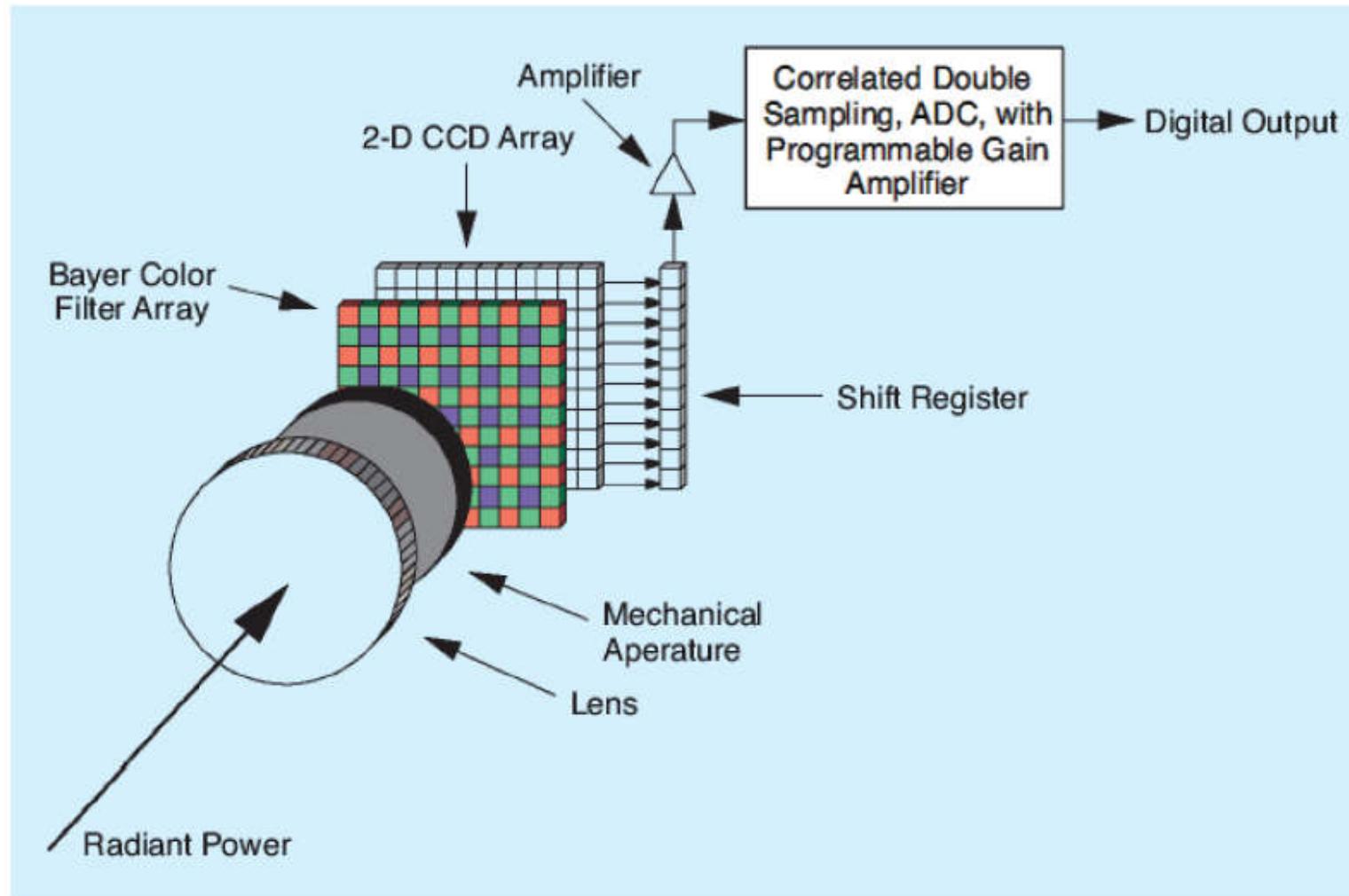
- Computer Vision
 - Techniques for image acquisition, extraction, characterization and interpretation of the information gathered from images of the 3D world.
 - **Machine Vision**
 - Computer Vision for automation and robotic applications.
- Image Processing
 - Signal processing where the input signal is an image (2 or 3 dimensional signal) and the output can be a transformation of this image or a set of characteristics associated to the image.
- Artificial Vision
 - Wide research field that includes all sciences and techniques that allow the study and application of all activities related to the use and interpretation of an image.

Computer vision

- Broad definition of computer vision & artificial vision areas
- **Some basic definitions**
- The typical image processing pipeline
- Image parameters (intrinsic and adjustable)
- Basic optics
- Camera parameters and perspective views
- Examples of simple image processing tasks

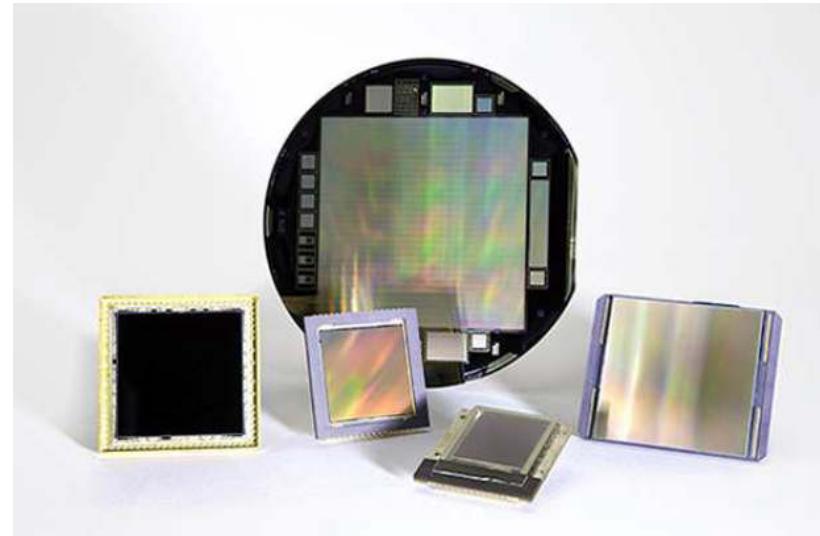
Digital Cameras

Image acquisition using a digital camera:
(IEEE SP Magazine, Jan 2005)



Digital Cameras (sensors)

- Some considerations: speed, resolution, cost, signal/noise ratio, . . .
- **CCD** - Charge Coupled Device
 - Higher dynamic range
 - High uniformity
 - Lower dark noise.
- **CMOS** - Complementary Metal Oxide Semiconductor
 - Lower voltage, lower power
 - Higher speed
 - Lower system complexity.
 - Very flexible in terms of in camera complementary hardware



Digital Cameras

- Several interfaces (Firewire, GigE, CameraLink, USB, . . .).
- Scientific usage (high resolution, long exposure time, . . .).
- High speed (ex. $>>1000$ fps).
- Linear (ex. 10000 lines/second).
- 3D
- Infrared (ex. 8 to 14 μm).
- Multispectral
- High dynamic range
(ex. using a prism and two sensors).



Definitions - Luminance

Luminance

Luminance is normally defined as a measurement of the photometric **luminous intensity per unit area of light travelling in a given direction.**

Therefore it is used to describe the amount of light that goes through, or is emitted from, a particular area, and falls within a given solid angle.

The SI unit for luminance is candela per square meter (cd/m^2).

The CGS unit of luminance is the *stilb*, which is equal to one candela per square centimeter or $10 \text{ kcd}/\text{m}^2$.

Definitions - Chrominance

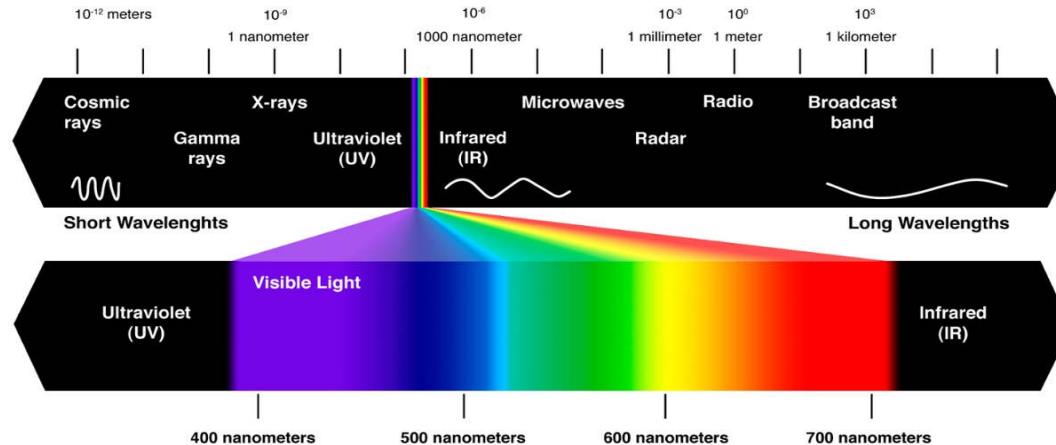
Chrominance

Chrominance is a numeral that describes the way a certain amount of light is distributed among the visible spectrum.

A black and white image has a balanced distribution of energy among the visible spectrum matched to the band pass characteristics of the human visual system. This means that when viewed by a human a B&W image has no color information which means that its color information is zero.

Therefore, chrominance has no luminance information but is used together with it to describe a colored image defined, for instance, by an RGB triplet.

Any RGB triplet in which the value of $R=G=B$ has no chrominance information.



RGB & YUV

Separating Luminance from Chrominance

Given an RGB triplet, we can define a derived triplet in which luminance and chrominance can be separated:

$$Y = W_r R + W_g G + W_b B \quad \text{Luminance}$$

$$U = U_{\max} \frac{B - Y}{1 - W_b} \approx 0.492(B - Y) \quad \text{Chrominance}$$

$$V = V_{\max} \frac{R - Y}{1 - W_r} \approx 0.877(R - Y) \quad \text{Chrominance}$$

where

$$W_r = 0.299$$

$$W_B = 0.114$$

$$W_G = 0.587$$

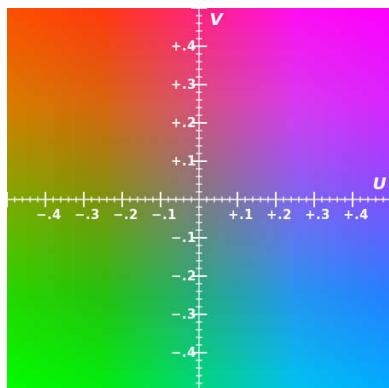
$$U_{\max} = 0.436$$

$$V_{\max} = 0.615$$

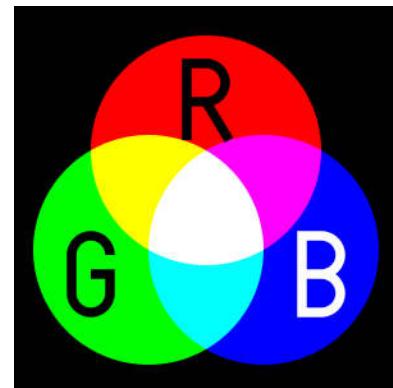
This values originally derives from the general model of the human visual system and had a significant impact on the ability to develop a television color system compatible with the previous B&W television systems.

A symmetric operation can be performed in order to recover the original RGB triple.

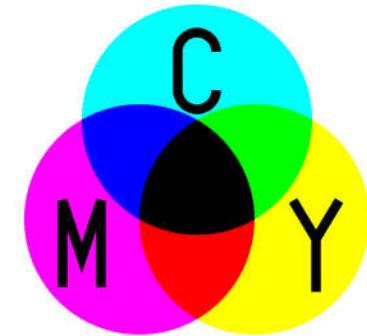
Examples of color spaces



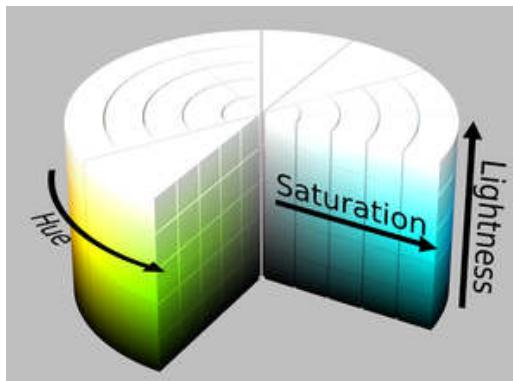
U-V Plane



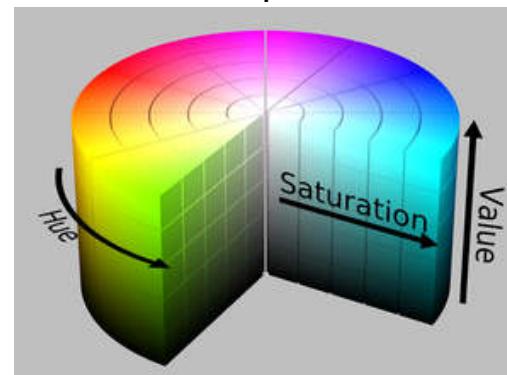
RGB Space



CYM(K) Space



HSL and
HSV
Spaces



HCL and
HCV
Spaces

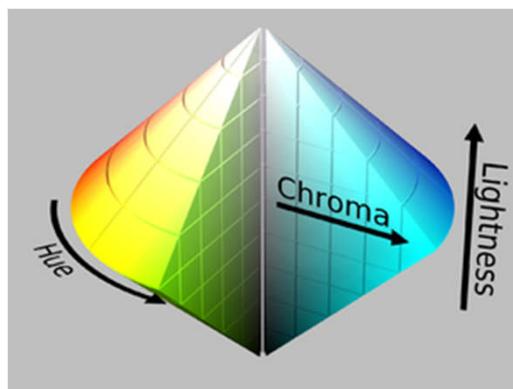


Image Sources: Wikipedia

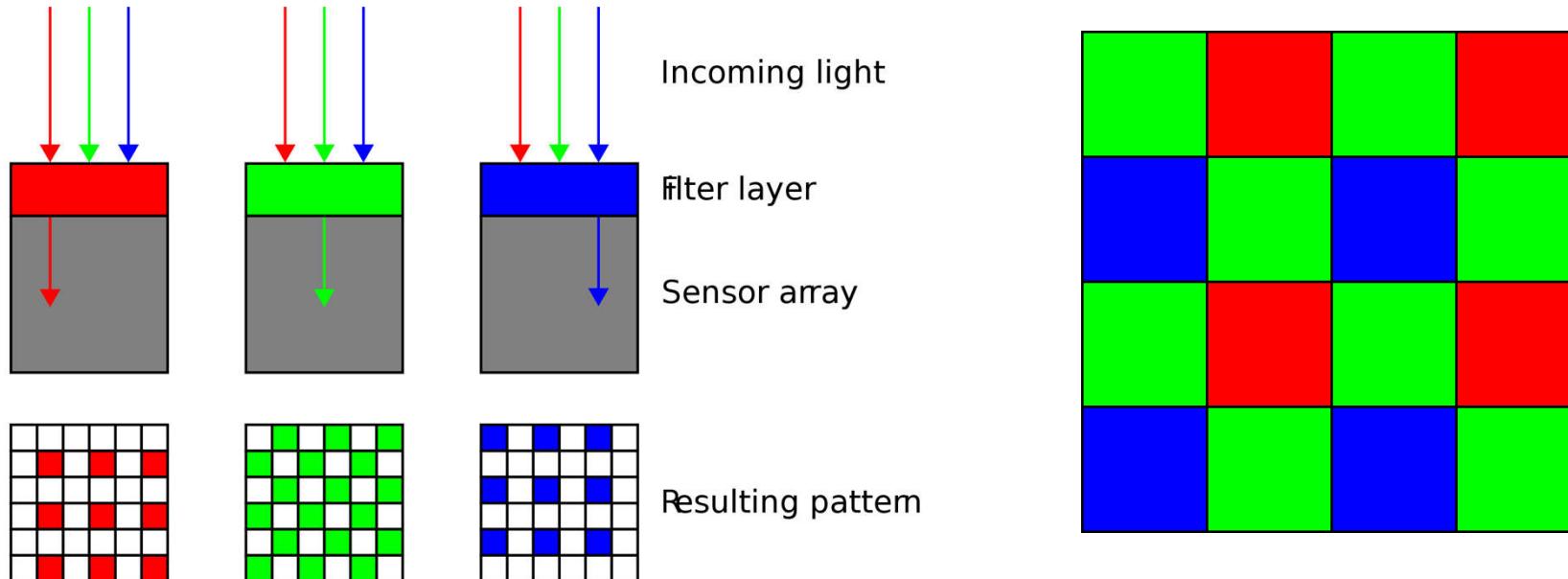
Image formation

Image formation

Most image sensors use either a CCD or CMOS technology that is able to “grab” light energy through a conversion process from photons to electrons which are then trapped either in an array of potential wells or an array of capacitors.

Only luminance is captured in this way.

To obtain a color image a set of filters must be used, from which the Bayer pattern configuration is the most common.



Computer vision

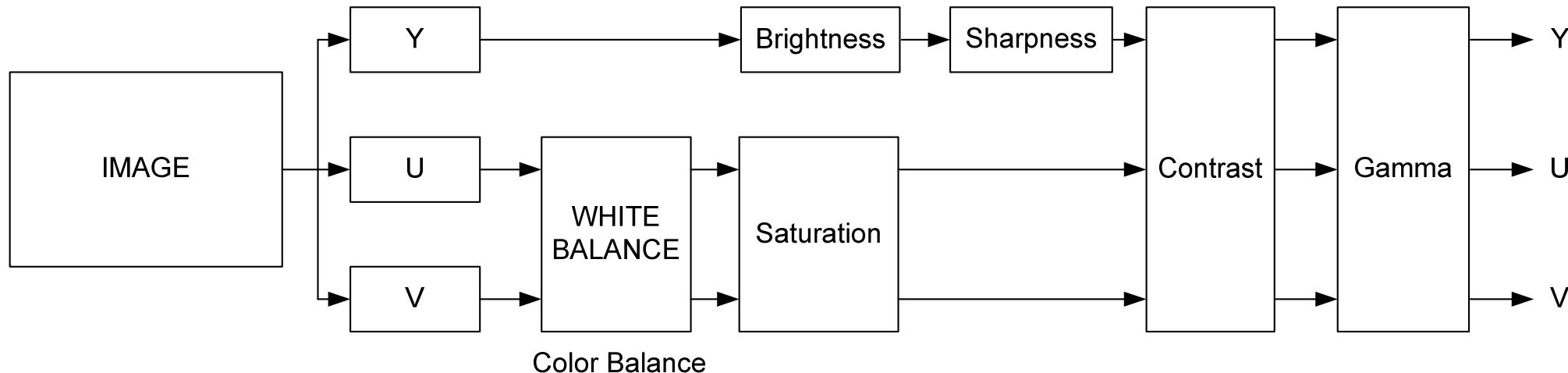
- Broad definition of computer vision & artificial vision areas
- Some basic definitions
- **The typical image processing pipeline**
 - Image parameters (intrinsic and adjustable)
- Basic optics
- Camera parameters and perspective views
- Examples of simple image processing tasks

The image processing pipeline

Image processing pipeline

A typical image processing pipeline (inside the image device) for a tri-stimulus system is shown below. This processing can be performed on the YUV or RGB components depending on the system.

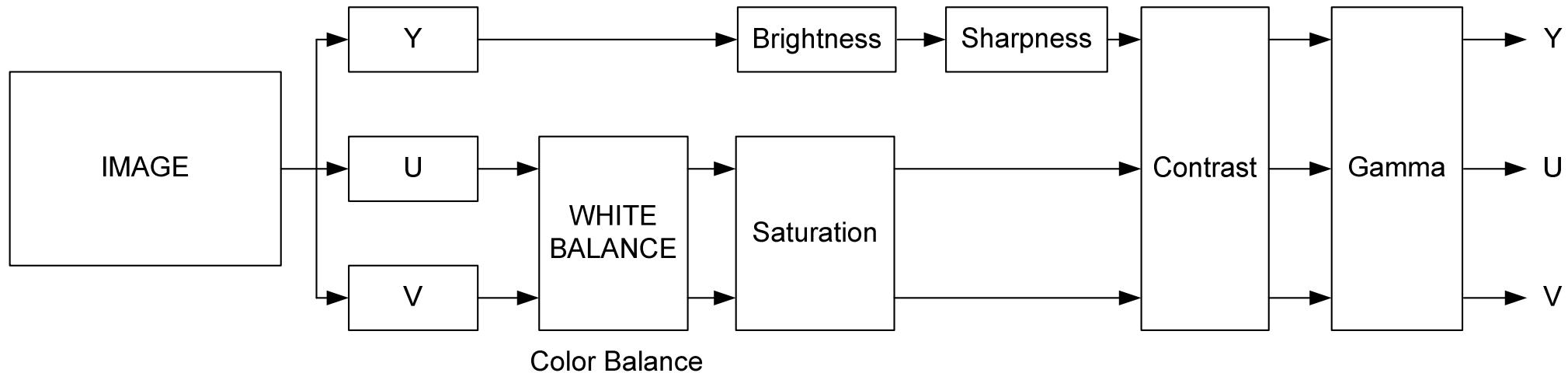
This should be understood as a mere example.



The image processing pipeline

Image processing pipeline

Depending on the system, more or less image parameters may be available for the user to control. Also, some of these parameters (namely brightness, contrast and saturation) are also intrinsic original image characteristics apart from being externally controllable parameters.



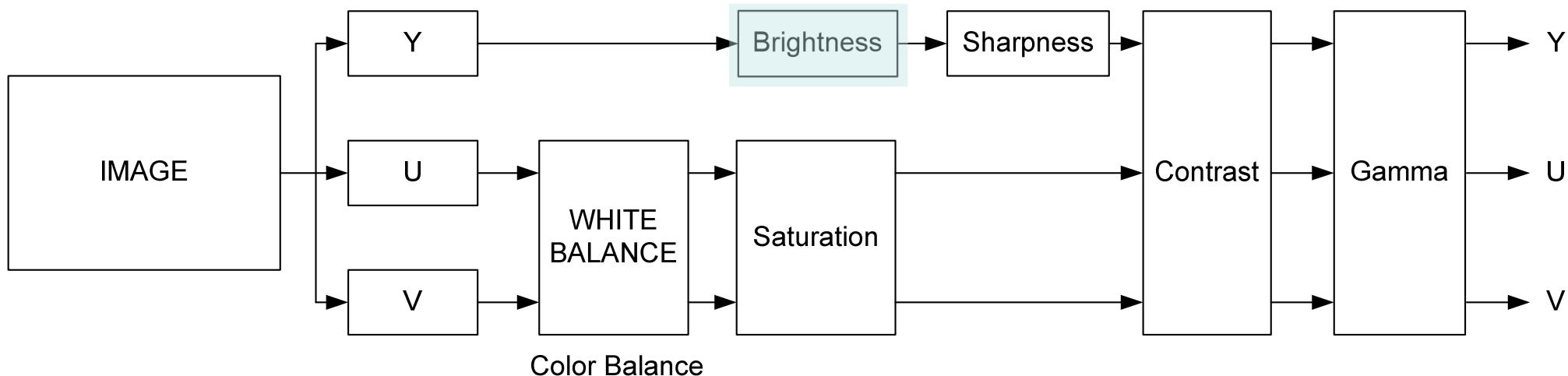
Brightness

Brightness (as an intrinsic image characteristic)

Brightness is one of the intrinsic original image characteristics. It represents a measure of the average amount of light that is integrated over the image during the exposure time.

Exposure time (that is, the period of time during which the sensor receives light while forming the image, may or may not be a controllable parameter of the image device).

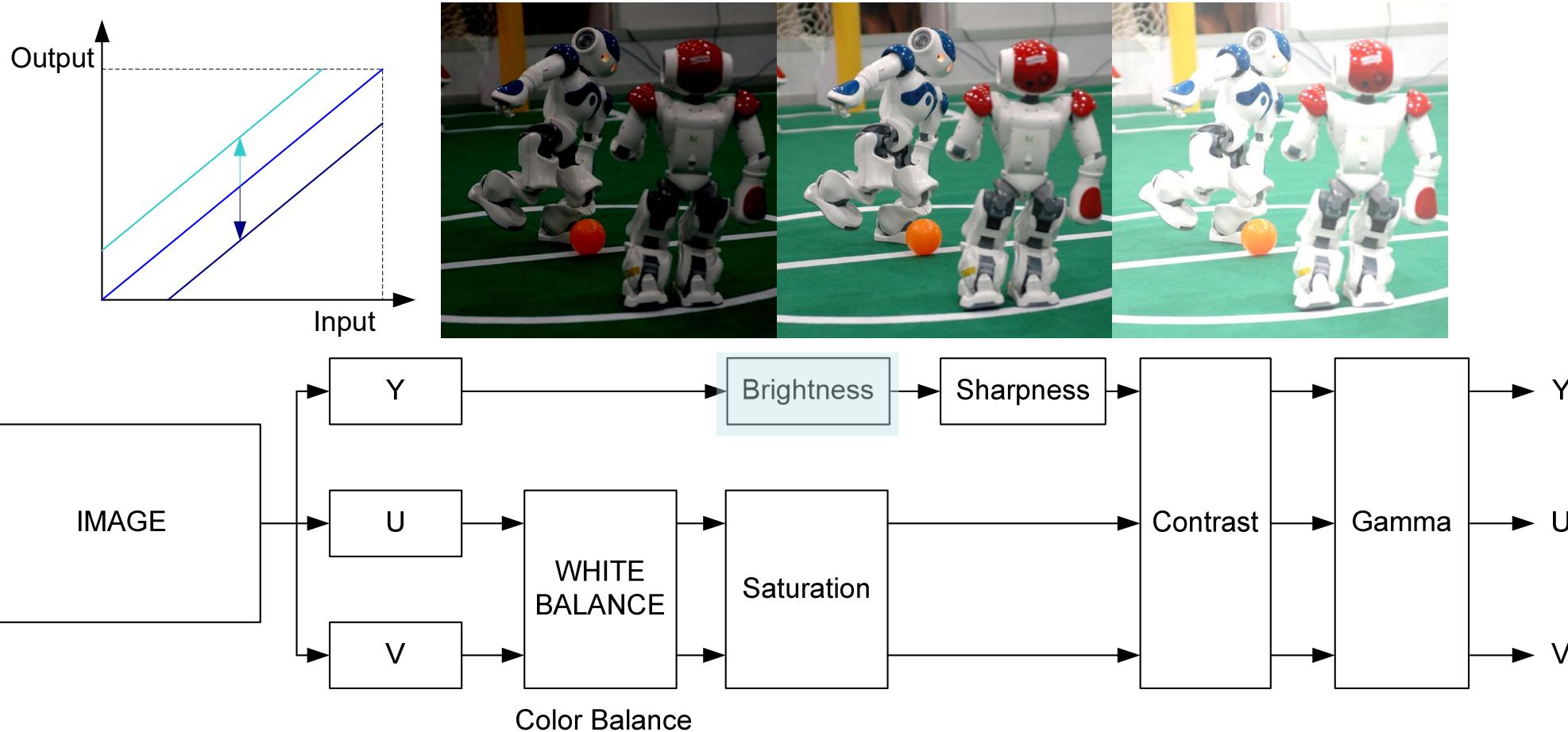
If the brightness is too high, overexposure may occur which will white saturate part or the totality of the image.



Brightness

Brightness (as a controllable parameter)

The brightness parameter is basically a constant (or offset) that can be added (subtracted) from the luminance component of the image.



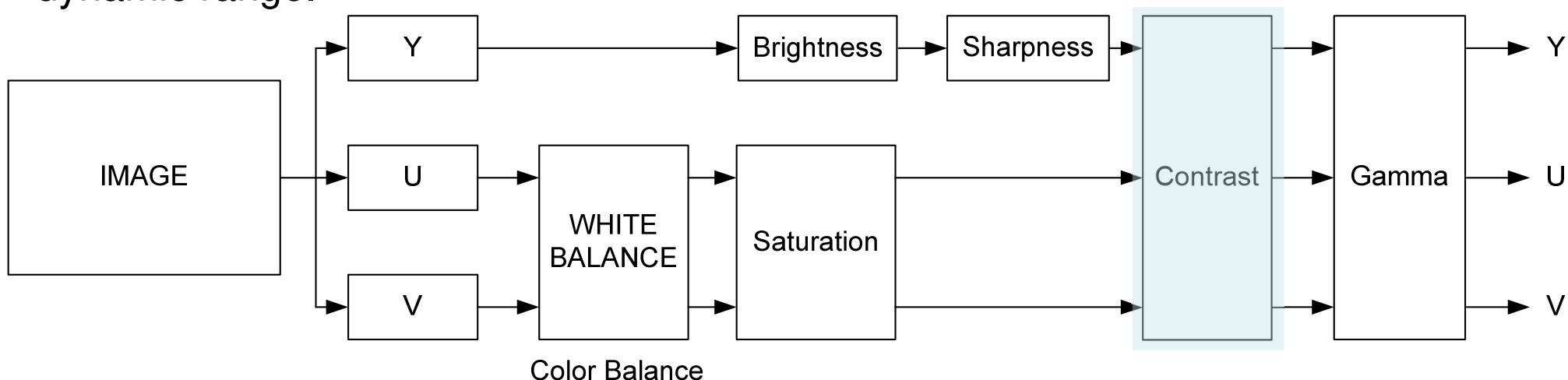
Contrast

Contrast (as an intrinsic image characteristic)

There is not a unique definition of contrast. One of the most used is that contrast is the difference in luminance (or color) along the 2D space that makes an object distinguishable. In visual perception of the real world, contrast is determined by the difference in the color and brightness of the object and other objects within the same field of view.

The faster and higher the luminance (or color) changes along the space the higher the contrast is.

The maximum possible contrast of an image is also denominated contrast ratio or dynamic range.



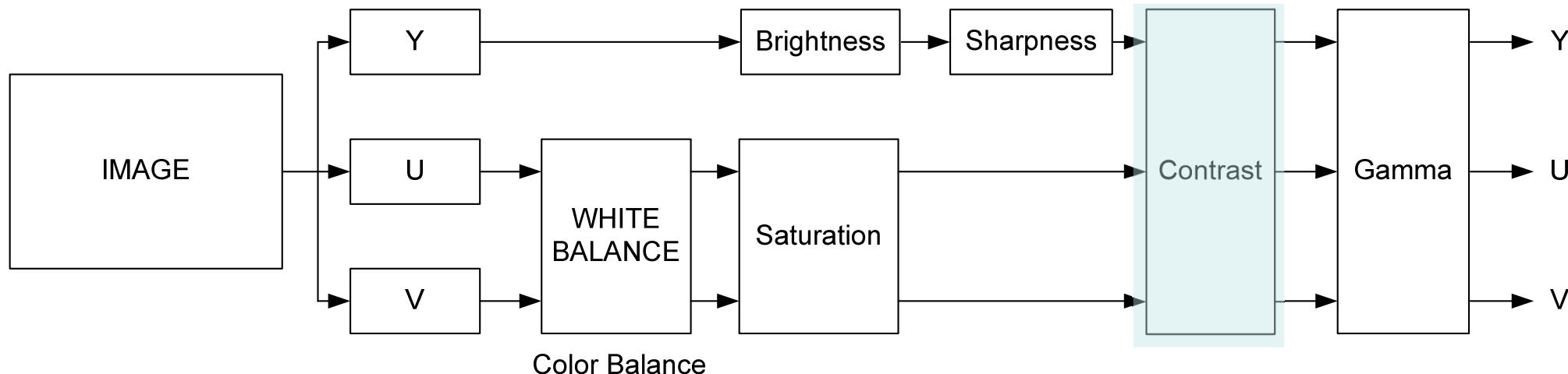
Contrast

Contrast (as an intrinsic image characteristic)

One of the possible definitions of contrast is given by the expression

$$\frac{\text{Luminance difference}}{\text{Average luminance}}$$

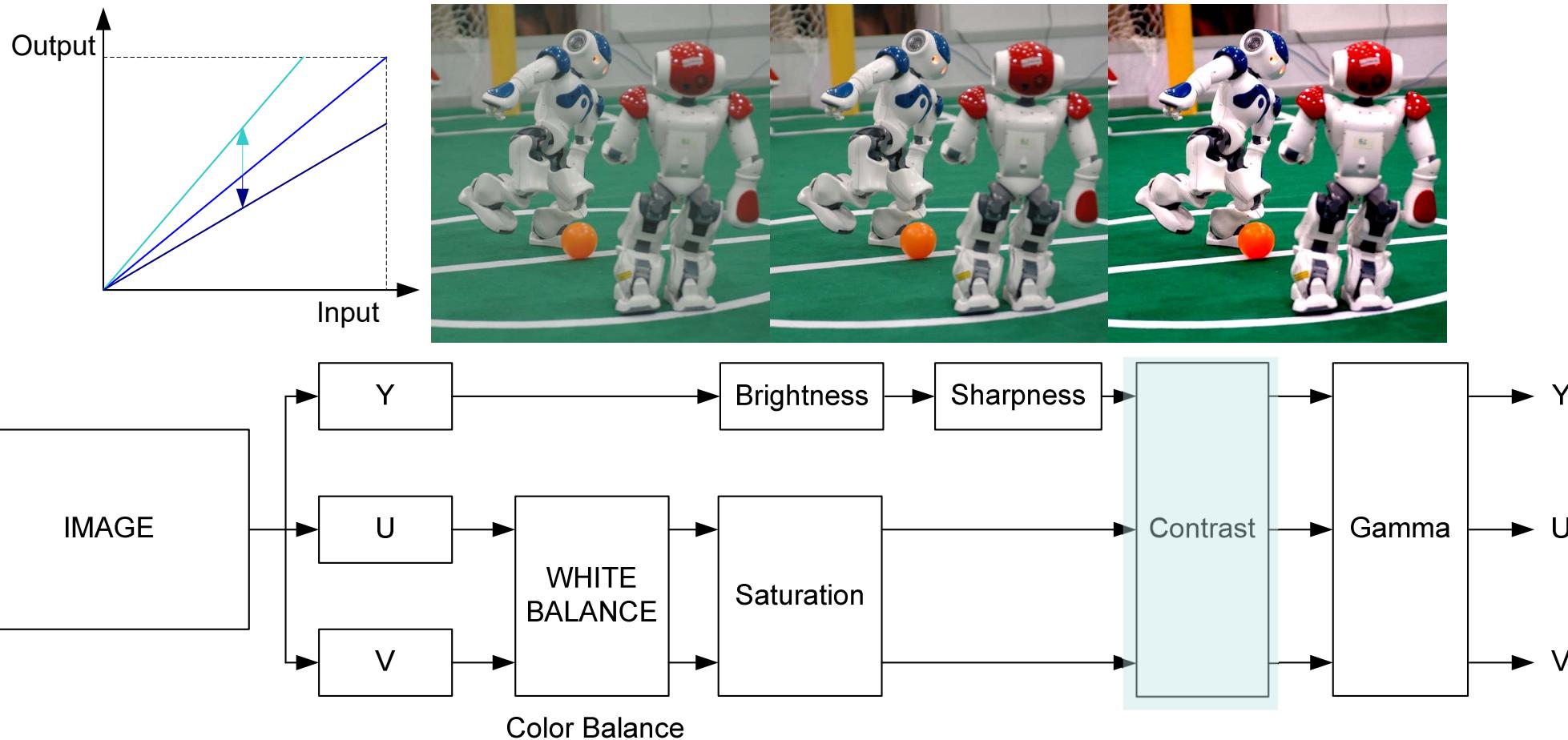
The human eye contrast sensitivity function is a typical band-pass filter with a maximum at around 4 cycles per degree with sensitivity reducing to both sides off that maximum. This means that the human visual system can detect lower contrast differences at 4 cycles per degree than at any other spatial frequency.



Contrast

Contrast (as a controllable parameter)

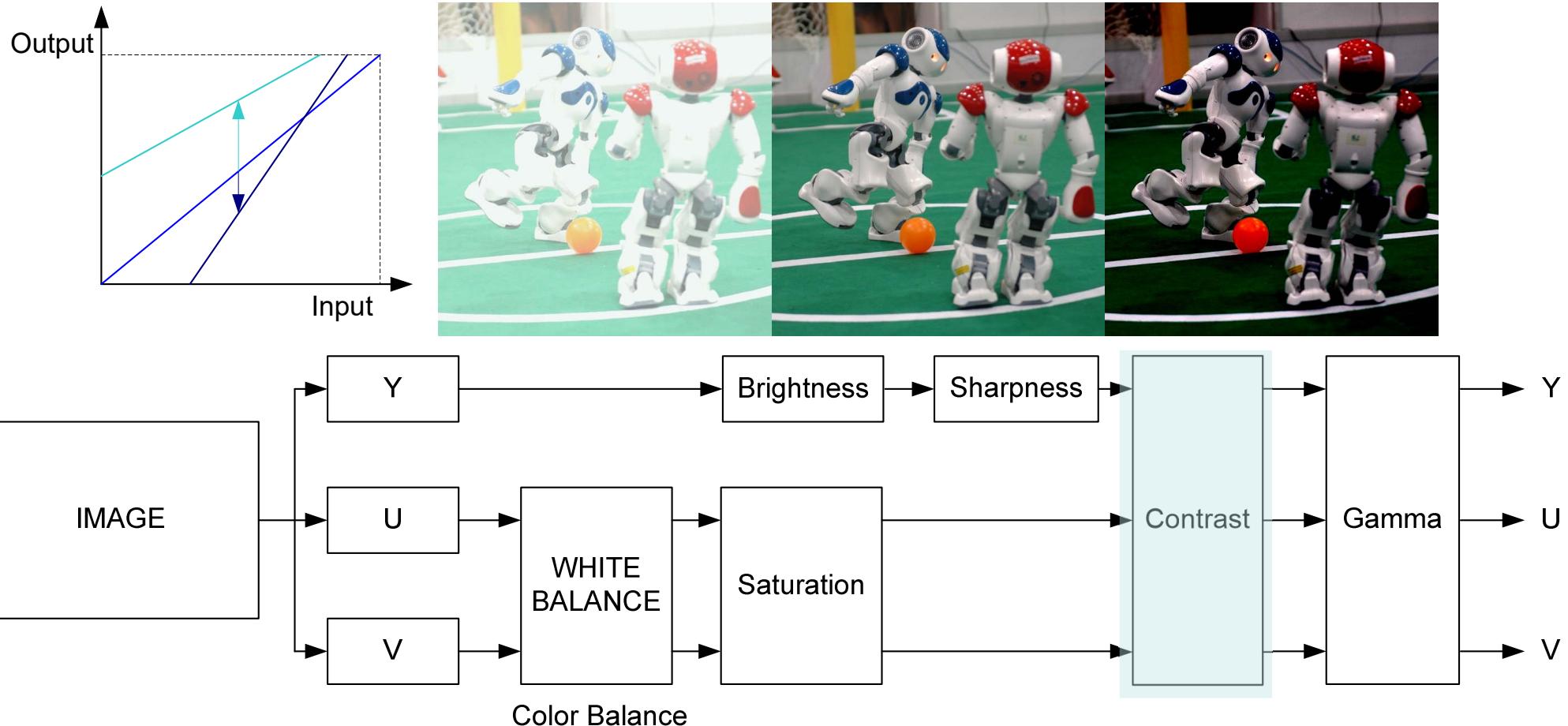
The contrast parameter is basically a variation in the gain control function of the luminance component of the image.



Contrast + Brightness

Contrast + Brightness(as controllable parameters)

It is common that contrast and brightness are actually a combined single transfer function.



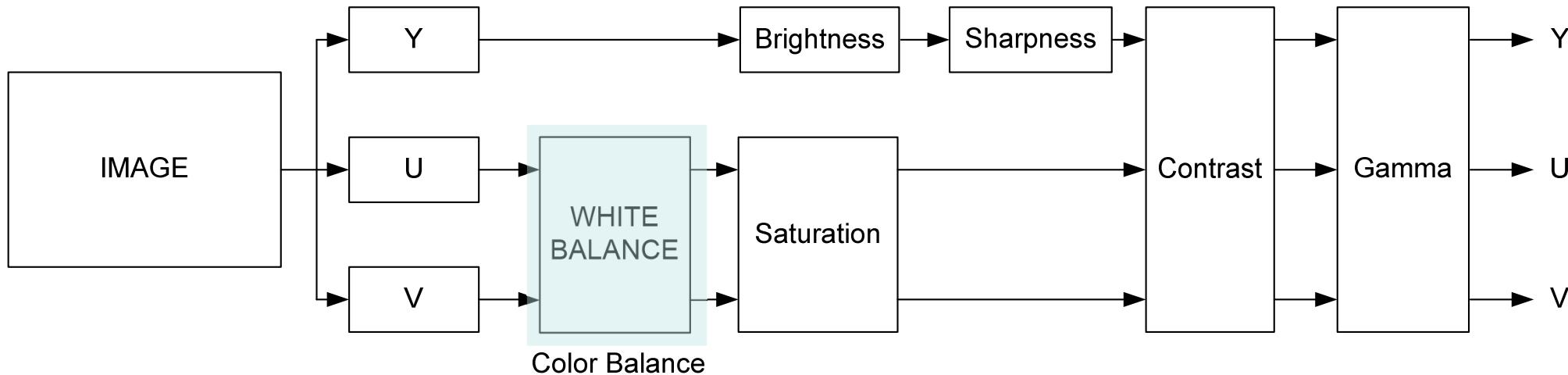
White Balance

White Balance (as controllable parameters)

White balance is the global adjustment of the intensities of the colors (typically red, green, and blue primary colors).

An important goal of this adjustment is to render specific colors – particularly neutral colors – correctly; hence, the general method is sometimes called gray balance, neutral balance, or white balance.

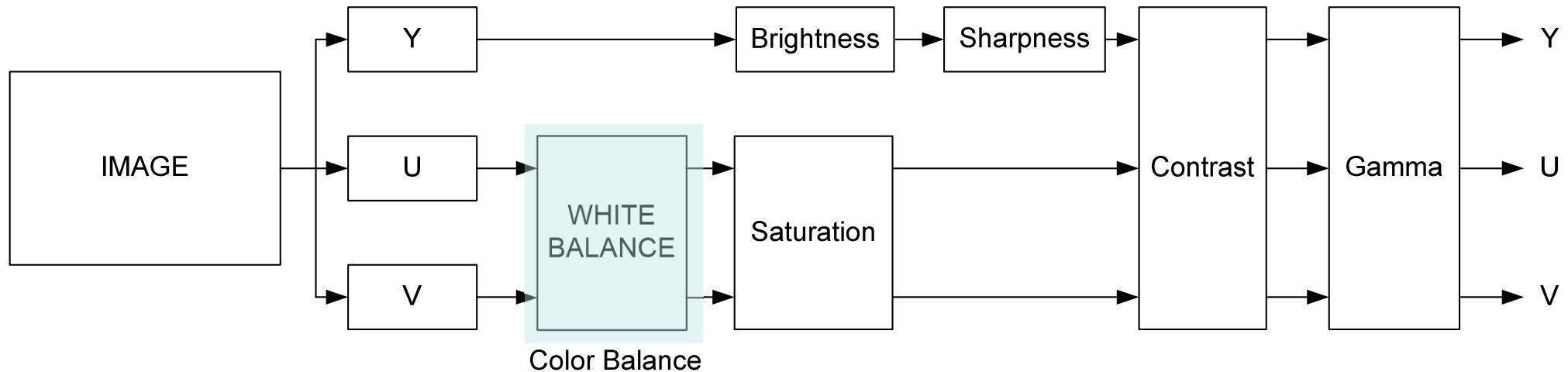
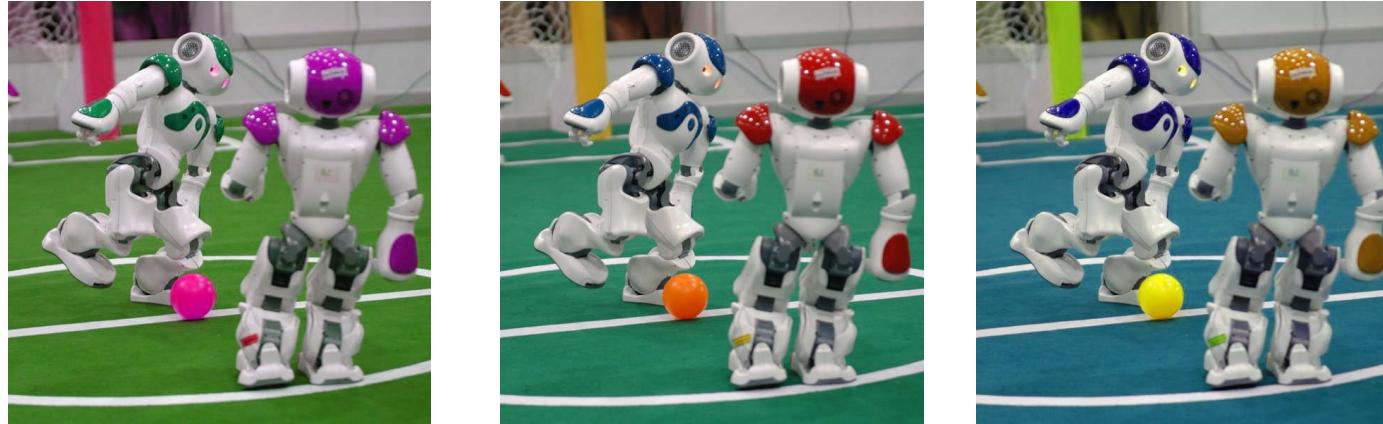
This balance is required because of different color spectrum energy distribution depending on the illumination source.



White Balance

White Balance

Examples



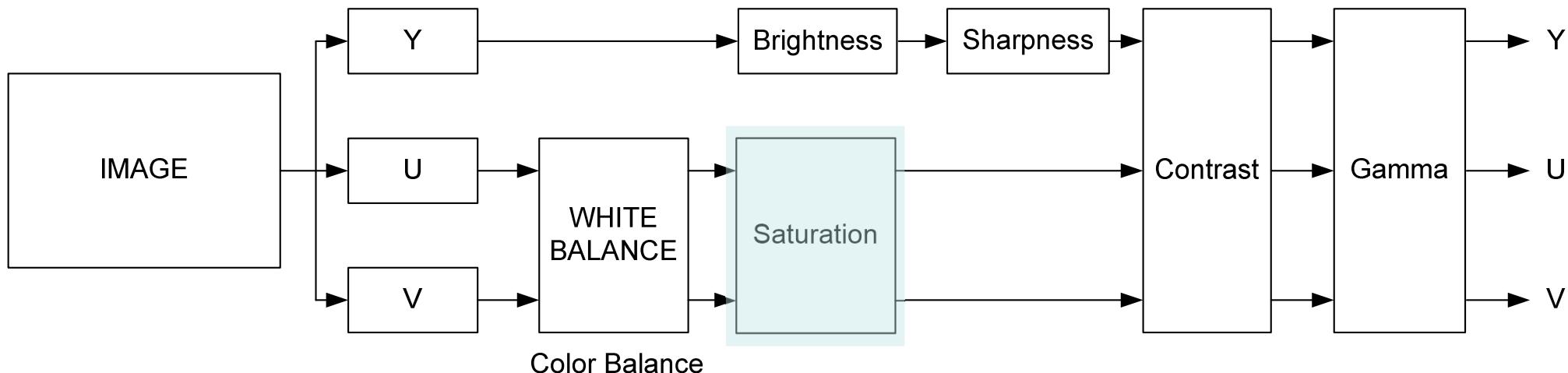
Saturation

Saturation (as an intrinsic image characteristic)

The saturation of a color is determined by a combination of light intensity that is acquired by a pixel and how much this light it is distributed across the spectrum of different wavelengths. The most purest (most saturated) color is obtained when using a single wavelength at a high intensity (laser light is a good example).

If the light intensity declines, then, as a result, the saturation also decline.

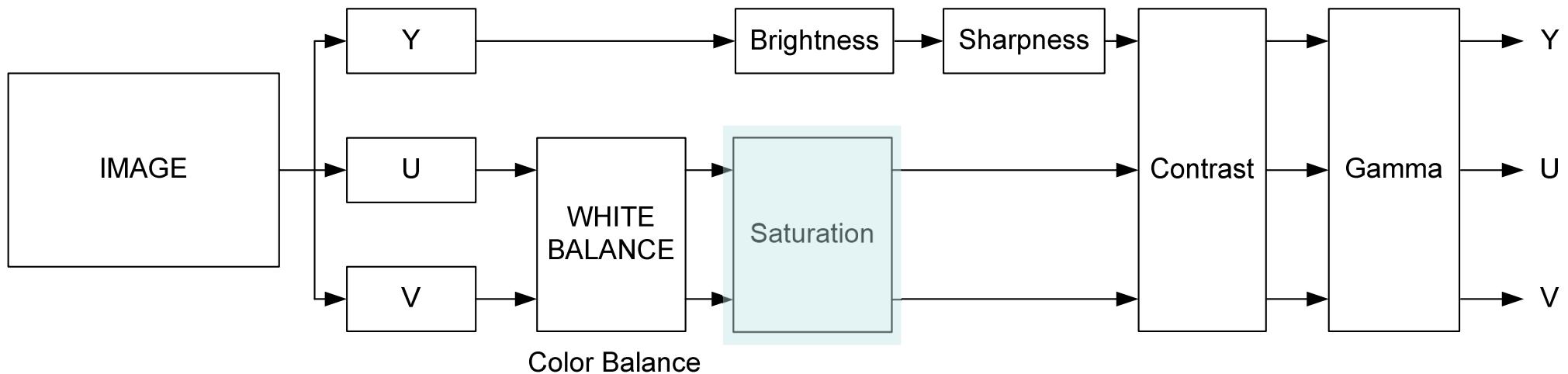
A non saturated image (B&W) has a spectrum distribution that matches the human eye spectrum sensibility. Saturation is sometimes also defined as the amount of white you have blended into a pure color.



Saturation

Saturation (as a controllable parameter)

To reduce the saturation of an image we can add white to the original colors. In fact this is the same as changing the gain of the U and V chromatic components.



Gama

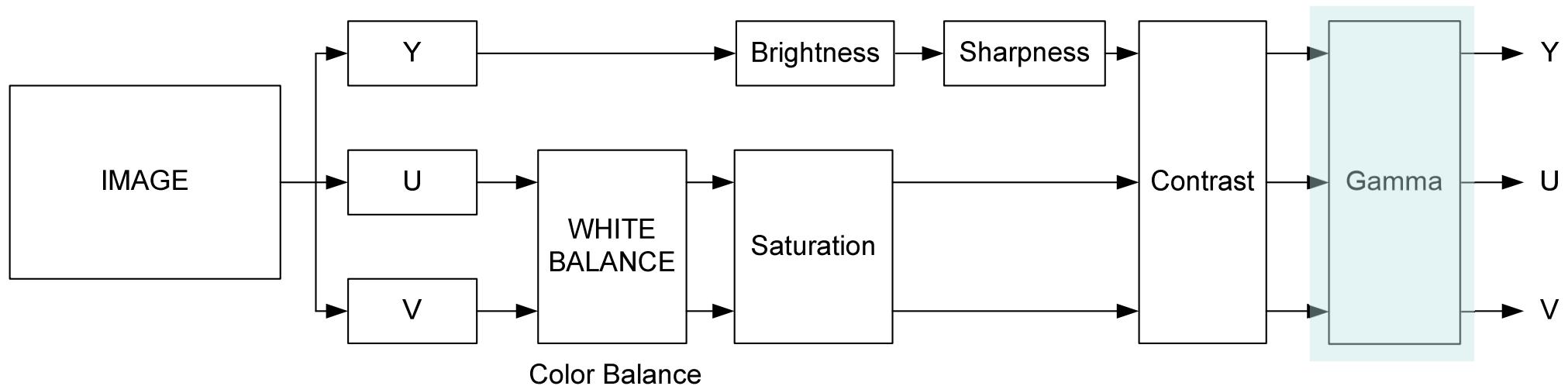
Gama

Gamma correction is the name of a nonlinear operation used to code and decode luminance or RGB tristimulus values. In the simplest cases gamma is defined by the power-law expression:

$$V_{out} = A V_{in}^{\delta}$$

where A is a constant and the input and output values are non-negative real values.

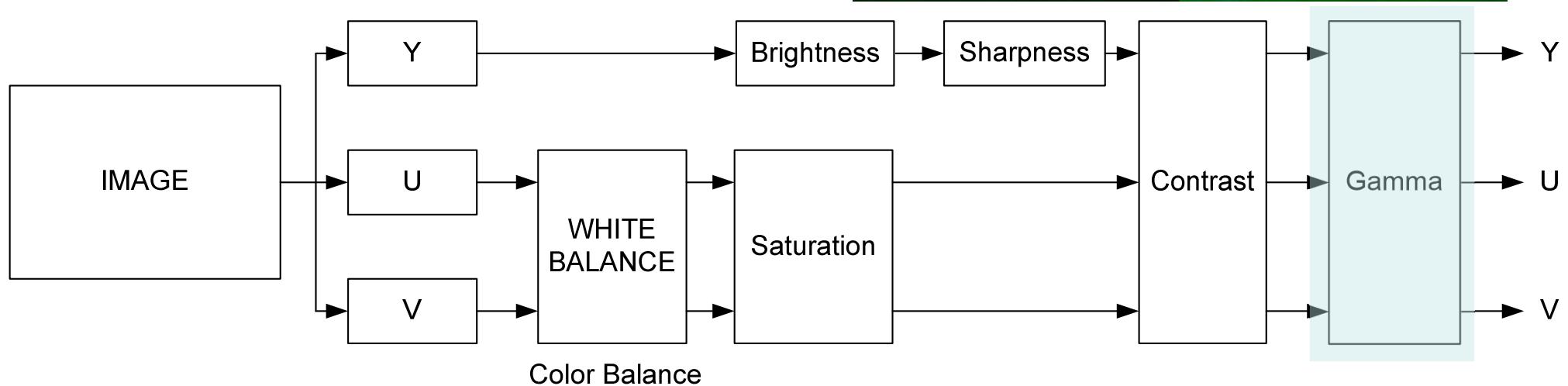
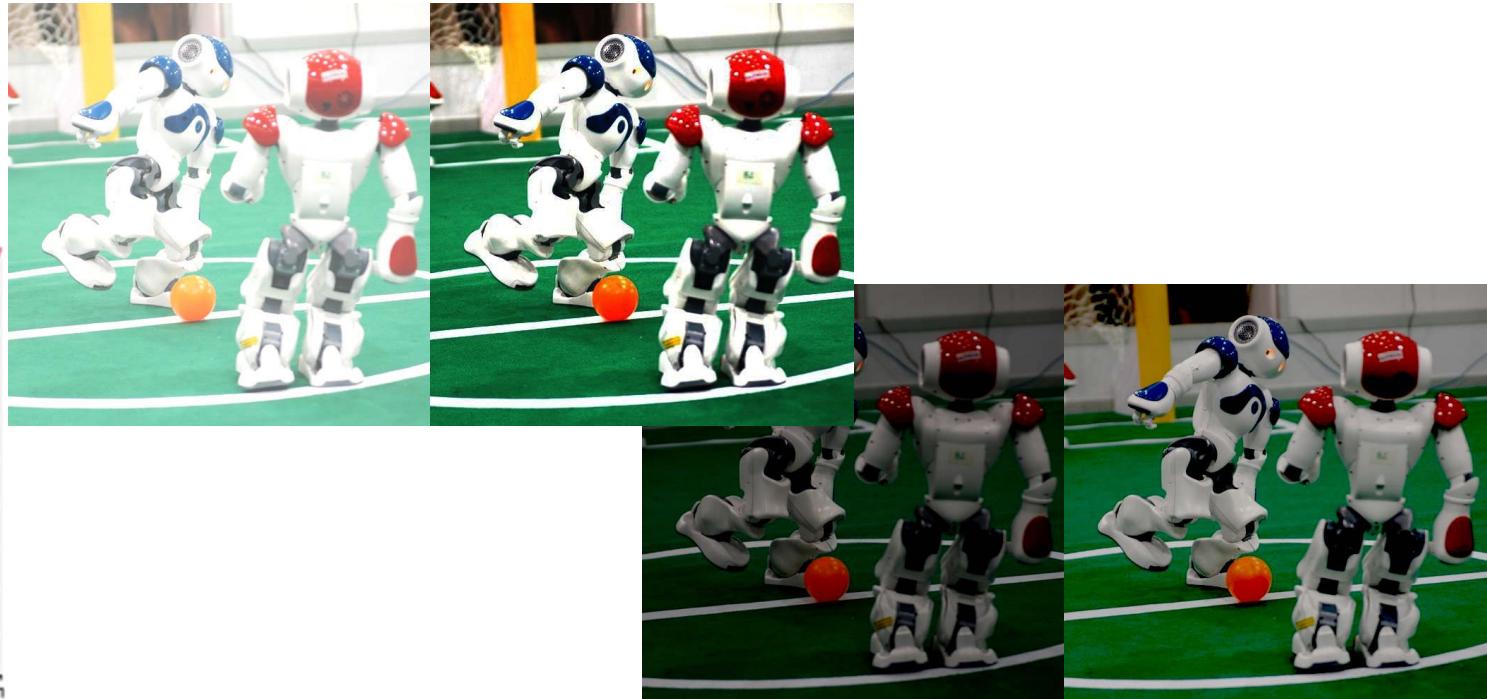
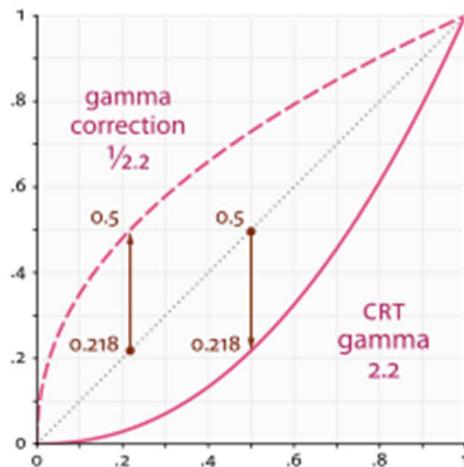
In most cases $A = 1$, and inputs and outputs are typically in the range 0–1.



Gamma

Gamma

Examples



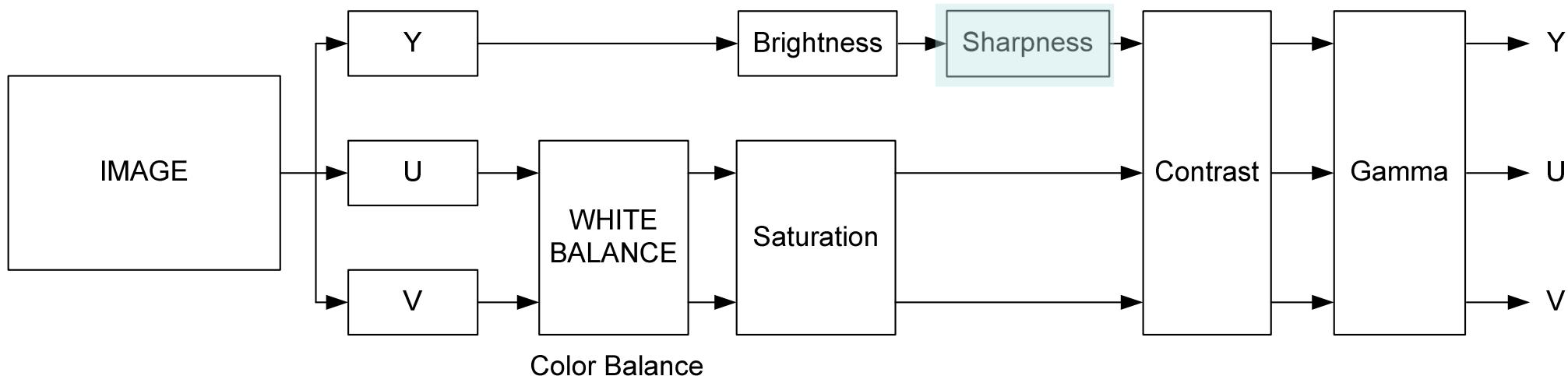
Sharpness

Sharpness (as a controllable parameter)

Sharpness is a measure of the energy frequency spatial distribution over the image.

Not all devices provide access to this parameter.

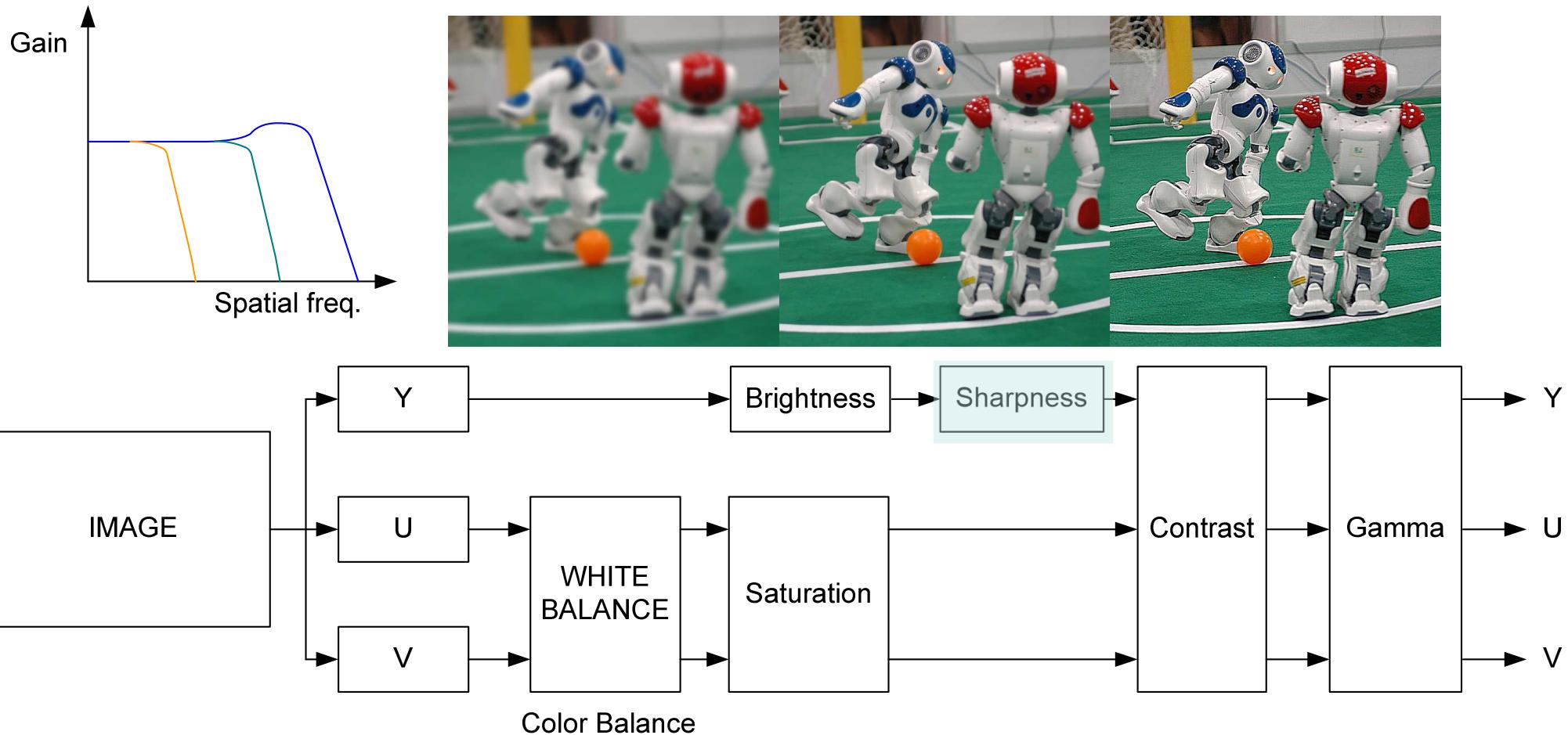
Sharpness basically allows the control of the cut-off frequency of a low pass spatial filter. This may be very useful if the image is afterward intended to be decimated, since it allows to prevent spatial aliases artifacts.



Sharpness

Sharpness (as a controllable parameter)

Examples.



Computer vision

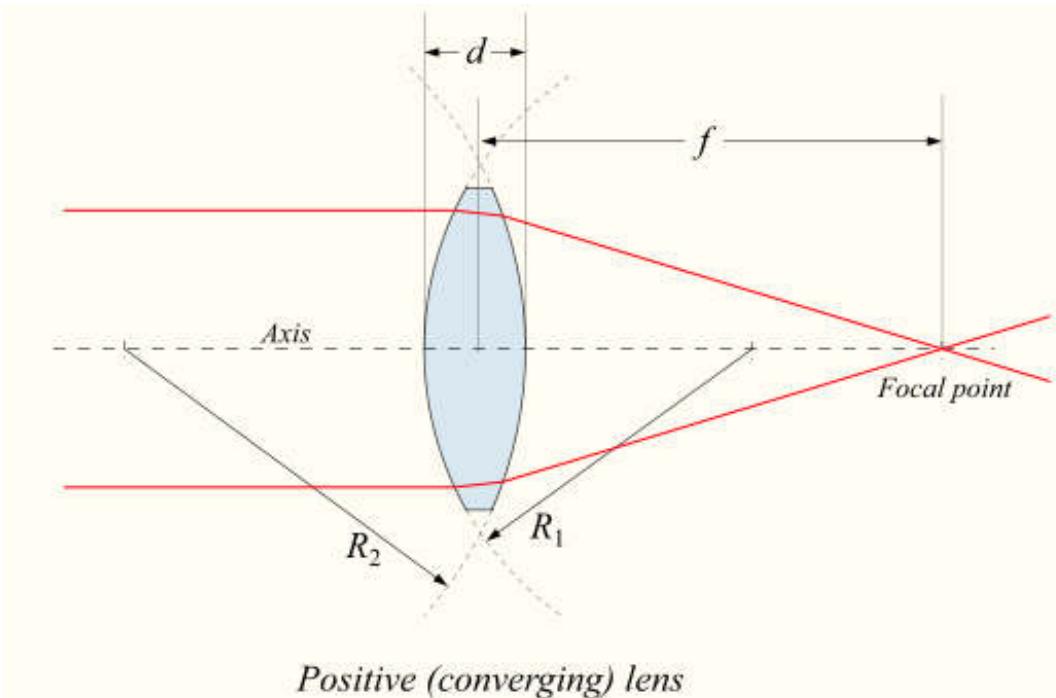
- Broad definition of computer vision & artificial vision areas
- Some basic definitions
- The typical image processing pipeline
- Image parameters (intrinsic and adjustable)
- **Basic optics**
- Camera parameters and perspective views
- Examples of simple image processing tasks

Optics basics

Optical lenses

The optical element of an image acquisition device is one of the most important elements of these devices. They can be made of complex groups of lenses, in particular when variable zooming is desirable. Optical component study goes far beyond the aim of this course.

Since most simple image acquisition systems, however, use a single converging lens as its optical interface, we will look simply into this case.



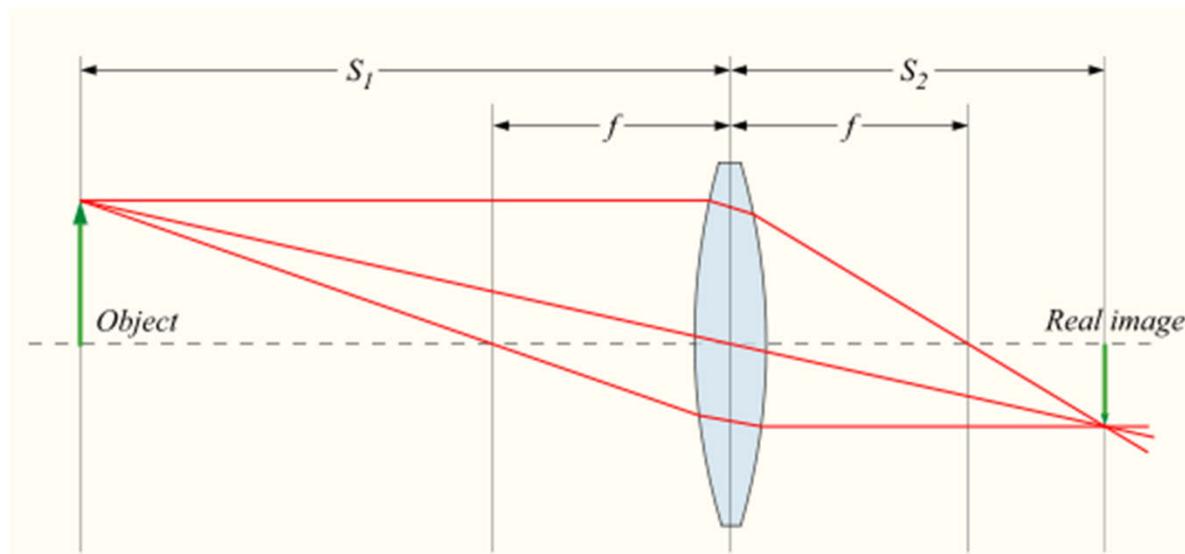
Optics basics

Formation of image

If the distances from the object to the lens and from the lens to the image are S_1 and S_2 respectively, for a lens of negligible thickness, in air, the distances are related by the thin lens formula which is an acceptable approximation to the full optical equation

$$\frac{1}{S_1} + \frac{1}{S_2} = \frac{1}{f}$$

If $S_1 \gg S_2$ then $S_2 \approx f$



Optics basics

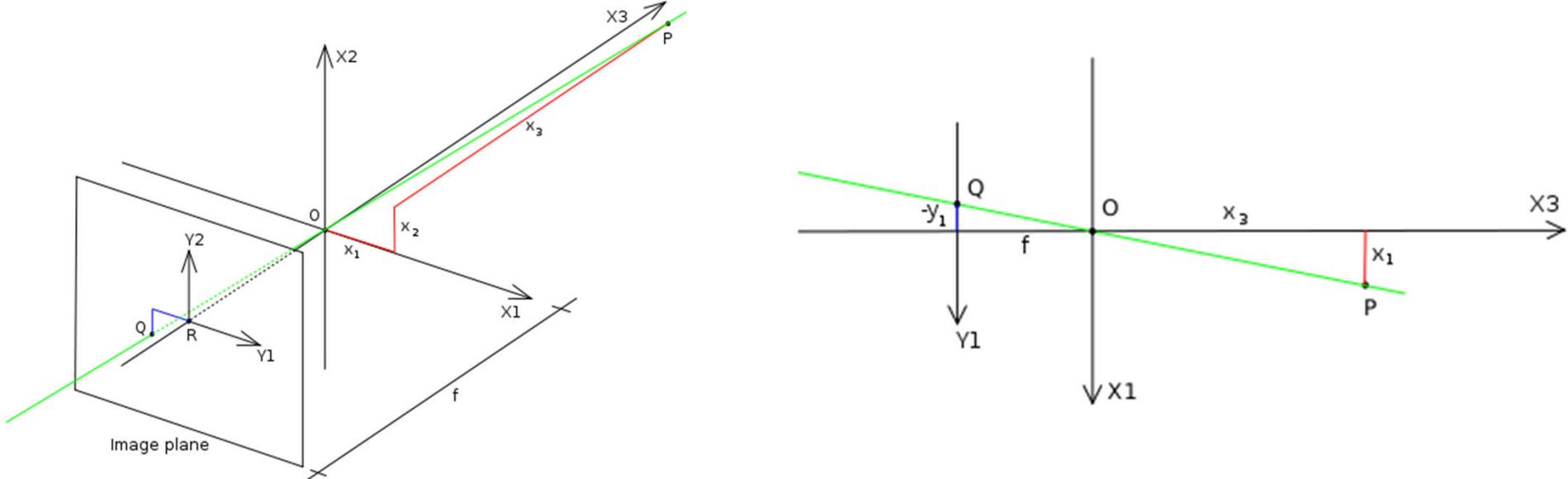
The pinhole model

For most practical digital image processing tasks, the optical system can be approximated by the pinhole model.

In this model, the lens is replaced by a very narrow opening (pinhole) through which lights go through directly into the image acquisition plane.

The point that is stroked by a light ray going through the pin hole in the direction of the lens main axis is called the image plane origin

In this model the acquisition plane lies at the focal distance from the pinhole.



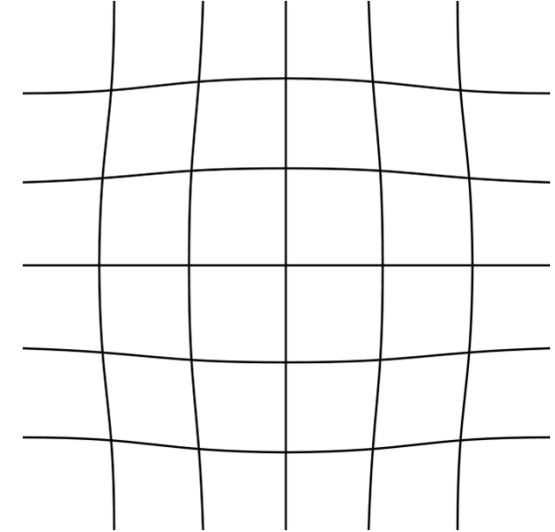
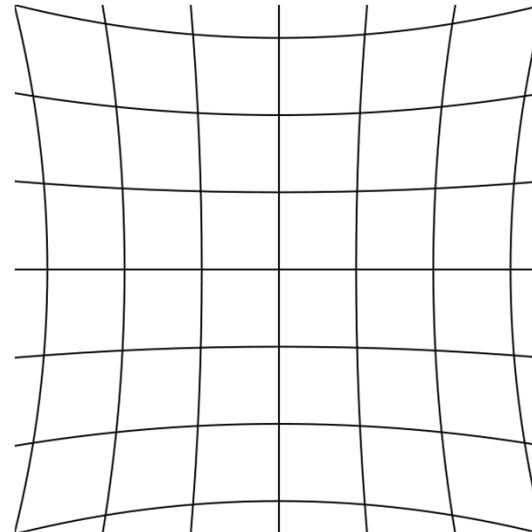
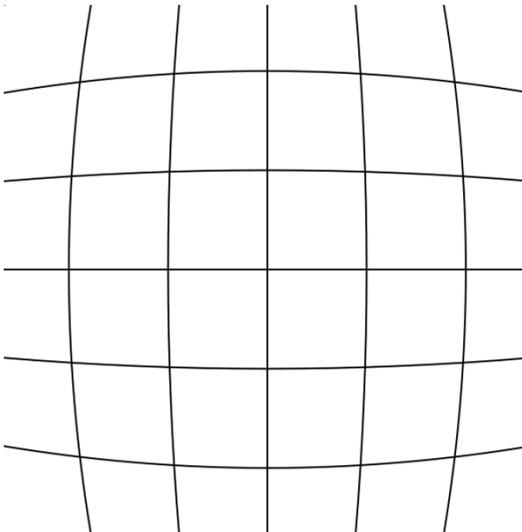
Lenses – Spherical Aberration

Spherical Aberration

Spherical Aberration is an image deformation resulting from the fact that most lenses have an external spherical surface cut which is easier to make than its proper shape.

Spherical Aberration can show as a barrel form (left side), pincushion effect (center) and Mustache distortion (right).

The barrel distortion is by far the most common one and normally increases with diminishing focal distance.



Lenses – Spherical Aberration

Spherical Aberration

Spherical Aberration results in multiple focus points depending on the distance at which each ray of light enters the lens when referred to its major axis, and can be corrected by using the simplified Brown's distortion model

$$x_u = (x_d - x_o)(1 + K_1 r^2 + K_2 r^4 + \dots)$$

$$y_u = (y_d - y_o)(1 + K_1 r^2 + K_2 r^4 + \dots)$$

where

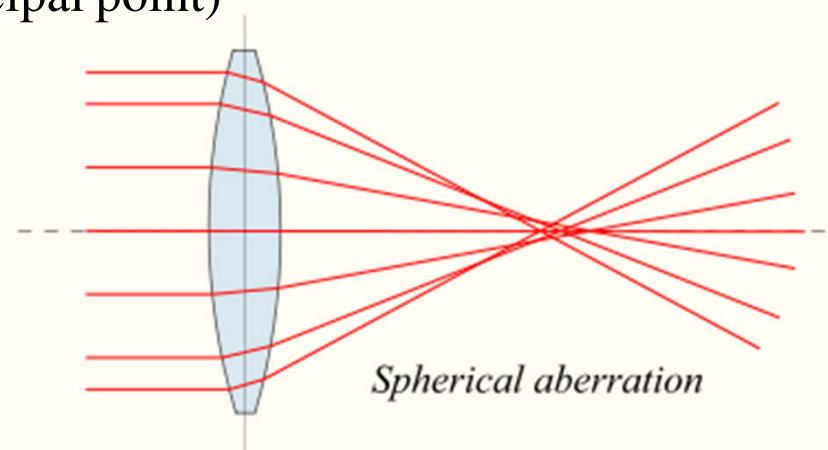
(x_d, y_d) - distorted image point as projected on image plane using specified lens

(x_u, y_u) - undistorted image point as projected by an ideal pin - hole camera

(x_o, y_o) - distortion center (assumed to be the principal point)

$K_n = n^{th}$ - radial distortion coefficient

$$r = \sqrt{(x_d - x_o)^2 + (y_d - y_o)^2}$$



Computer vision

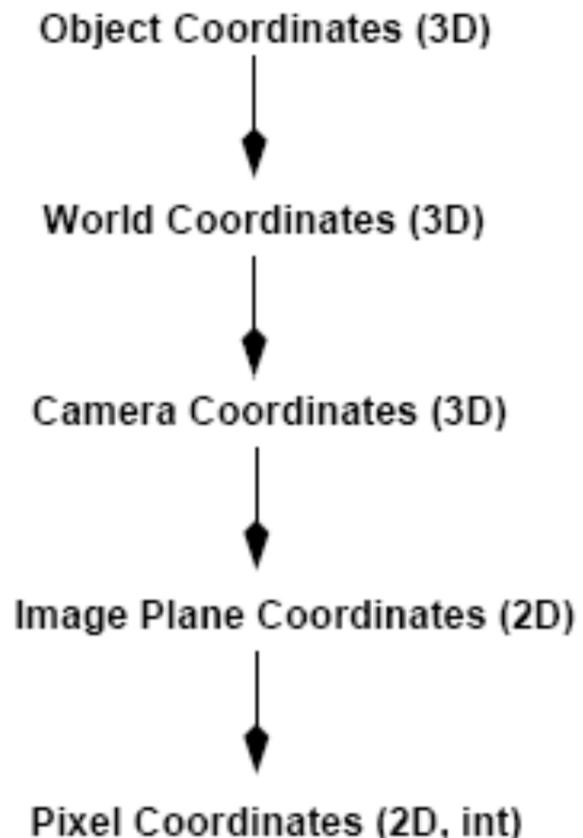
- Broad definition of computer vision & artificial vision areas
- Some basic definitions
- The typical image processing pipeline
- Image parameters (intrinsic and adjustable)
- Basic optics
- **Camera parameters and perspective views**
- Examples of simple image processing tasks

Camera parameters

One of the many applications of computer vision is to extract, from the frame image, information regarding the external world from which it has a limited perspective.

These applications may include, for instance, the determination of a certain distance of a point represented in the image when evaluated in real world coordinate system. This is only valid if we also know the plane in which this original point lies.

Therefore, geometrical transformations must be performed in order to accommodate the camera internal parameters and geometry and its position and posture in the real world.



Homogenous Coordinates

Homogenous Coordinates, in comparison to Cartesian coordinates, add an extra coordinate and define an equivalence relationship

$$(x, y) \rightarrow (kx, ky, k)$$

$$(X, Y, Z) \rightarrow (wX, wY, wZ, w)$$

This implies that any point in a 3D space can be represented by a multitude of equivalent matrixes, since the cartesian coordinates can be recovered from the homogenous coordinates by dividing each coordinate by the factor w .

In fact, this even allows us to represent any point in a plane which is at an infinite distance from the origin. Such representation will have its $w = 0$

Homogenous Coordinates

A rotation of a vector defined by two points in a Cartesian system can be obtained from (point a_n as the coordinates of the point to be projected, c_n as the pinhole coordinates and d_n as resulting rotated vector.

$$\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x \\ 0 & \sin \theta_x & \cos \theta_x \end{bmatrix} \begin{bmatrix} \cos \theta_y & 0 & \sin \theta_y \\ 0 & 1 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y \end{bmatrix} \begin{bmatrix} \cos \theta_z & -\sin \theta_z & 0 \\ \sin \theta_z & \cos \theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} \right)$$

The translation vector T_v , for the point p can, on the other hand, can be defined in a homogenous form by

$$T_v p = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix} = p + v$$

Homogenous Coordinates

One of the most interesting things in the use of homogenous coordinates is that we are allowed to combine a rotation matrix and a translation matrix into a single homogenous matrix.

$$\begin{array}{ccc}
 \text{Rotation} & & \text{Translation} \\
 \text{Matrix} & \xrightarrow{\quad} & \text{Matrix} \\
 \left[\begin{array}{c} X_c \\ Y_c \\ Z_c \\ 1 \end{array} \right] & = & \left[\begin{array}{c} - & - & - \\ - & R & - \\ - & - & - \\ 0 & 0 & 0 \end{array} \right] & \left[\begin{array}{c} X_w \\ Y_w \\ Z_w \\ 1 \end{array} \right]
 \end{array}$$

The diagram illustrates the combination of a rotation matrix and a translation matrix into a single homogenous matrix. On the left, a column vector $\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$ is labeled "Rotation Matrix". An arrow points from this vector to a 4x3 matrix $\begin{bmatrix} - & - & - \\ - & R & - \\ - & - & - \\ 0 & 0 & 0 \end{bmatrix}$. This matrix is labeled "R" in its center. An equals sign follows this. To the right of the equals sign is another 4x3 matrix, labeled "Translation Matrix" above it. This matrix has a column of ones at the bottom: $\begin{bmatrix} - & - & - \\ - & RT & - \\ - & - & - \\ 1 & 1 & 1 \end{bmatrix}$. Arrows point from the bottom-right corner of the first matrix to the top-left corner of the second matrix, indicating they are multiplied together.

Camera parameters

Camera parameters are normally divided into two big groups:

Extrinsic: the parameters that define the *location* and *orientation* of the camera reference frame with respect to a known world reference frame.

Intrinsic: the parameters necessary to link the pixel coordinates of an image point with the corresponding coordinates in the camera reference frame.

Camera parameters

Camera Intrinsic parameters

Intrinsic parameters are defined as a matrix of the form

$$A = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

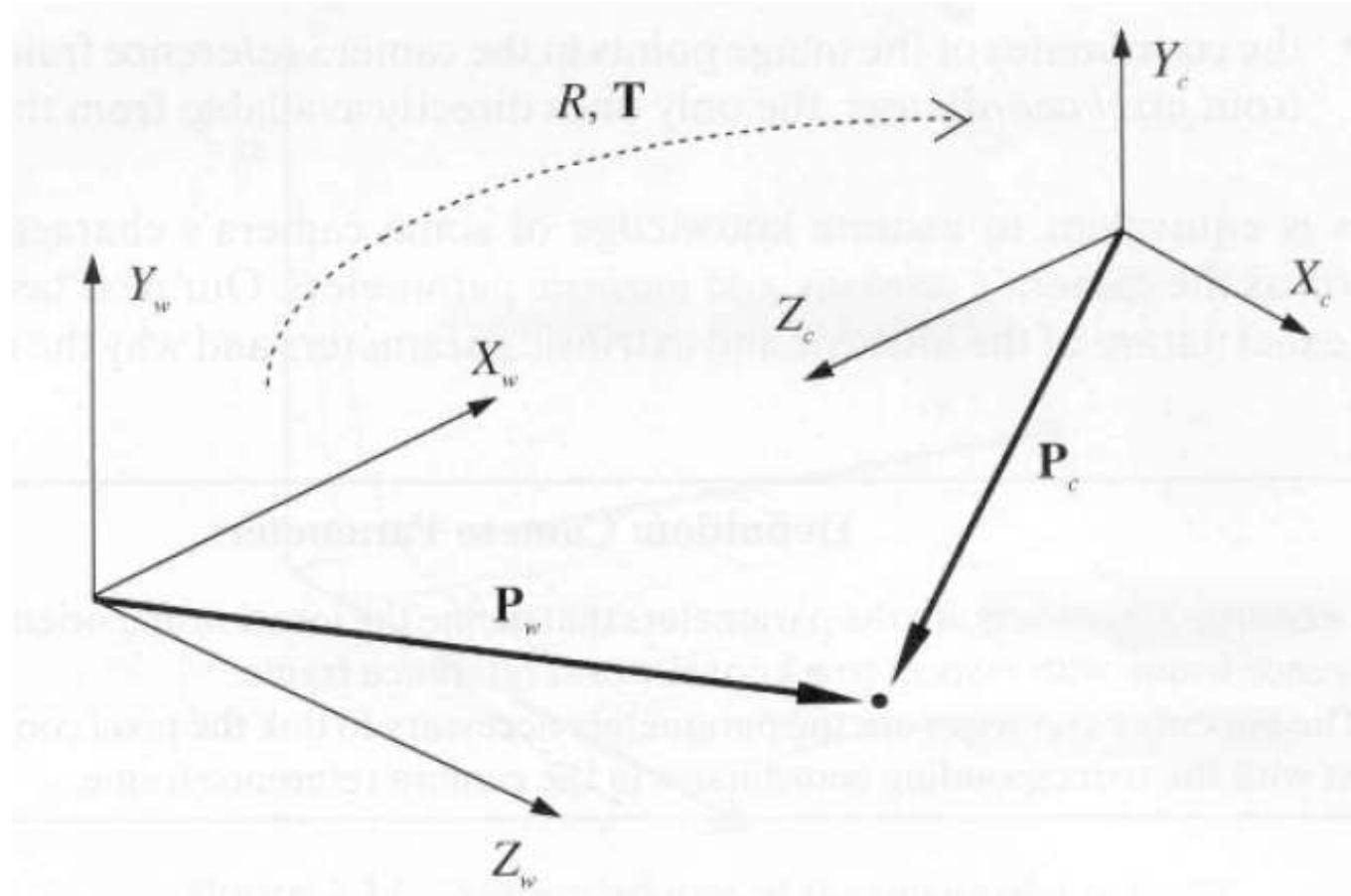
The intrinsic matrix containing 5 intrinsic parameters. These parameters include focal length, image format, and principal point. The parameters $\alpha_x = f \cdot m_x$ and $\alpha_y = f \cdot m_y$ represent focal length in terms of pixels, where m_x and m_y are the scale factors relating pixels to distance. γ represents the skew coefficient between the x and the y axis, and is normally 0. u_0 and v_0 represent the principal point, which would be ideally in the centre of the image.

Richard Hartley and Andrew Zisserman (2003). Multiple View Geometry in Computer Vision. Cambridge University Press. pp. 155–157. ISBN 0-521-54051-8

Camera parameters

Camera Extrinsic parameters

Extrinsic parameters denote the coordinate system transformations from 3D world coordinates to 3D camera coordinates.



Camera parameters

Camera Extrinsic parameters

In other words, **Extrinsic Parameters**, reflect the projection view of the camera taking into consideration its relative position to the world coordinates (translation operation) and its rotations when we consider the camera coordinate system three axis in relation to the world coordinate system three axis.

To note the **Extrinsic parameters** does not reflect the position of the camera in regard to the world coordinate system. It is, in fact, the position of the origin of the world coordinate system expressed in coordinates of the camera-centered coordinate system.

If R and T are the extrinsic parameters, then the position of the camera C expressed in world coordinates is

$$C = -R^{-1}T$$

Parameters of the general model

Still referring to the pinhole camera model, a camera matrix can therefore be used to denote a general projective mapping from World coordinates.

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A [R \ T] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Where A represents the camera intrinsic parameters, R and T the rotation and translation matrix, \mathbf{k}_w each of the coordinates in 3D space and u and v the indexes of each pixel on the frame buffer.

Computer vision

- Broad definition of computer vision & artificial vision areas
- Some basic definitions
- The typical image processing pipeline
- Image parameters (intrinsic and adjustable)
- Basic optics
- Camera parameters and perspective views
- **Examples of simple image processing tasks**

Examples of simple image processing tasks

Spatial filtering: applying a convolutional matrix



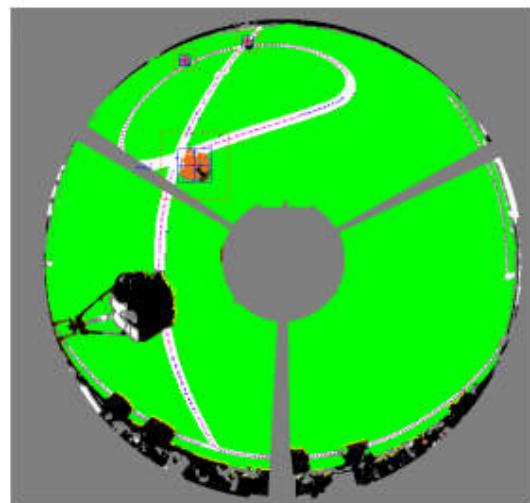
Low pass / High pass filtering



Edge detection
(Canny, Laplace, Sobel,
...)

Examples of simple image processing tasks

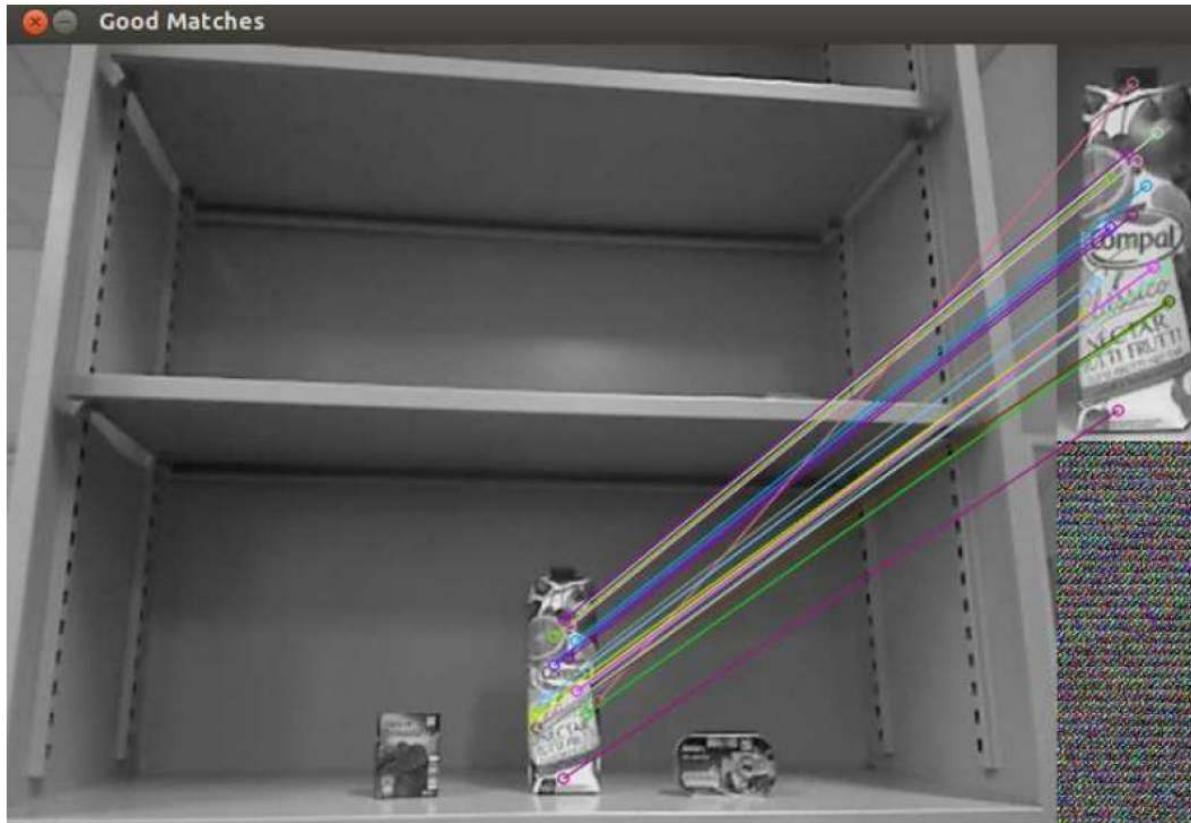
Segmentation:



Other image processing tasks

Other more complex processing techniques allow more complex results but can not be covered here. Examples are:

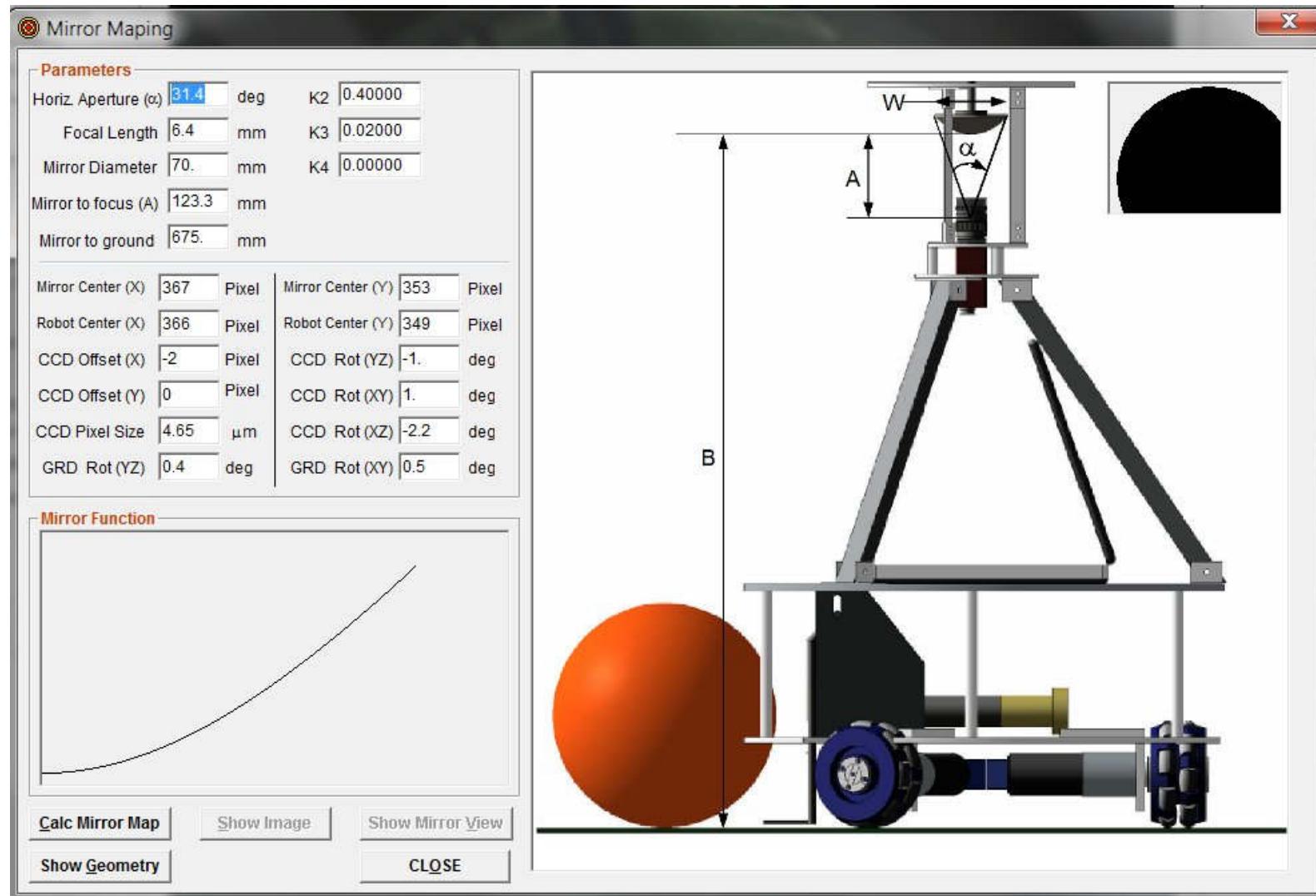
- Morphological operators
- Image descriptors
- Pattern recognition
- ...



Example of use of
image descriptors

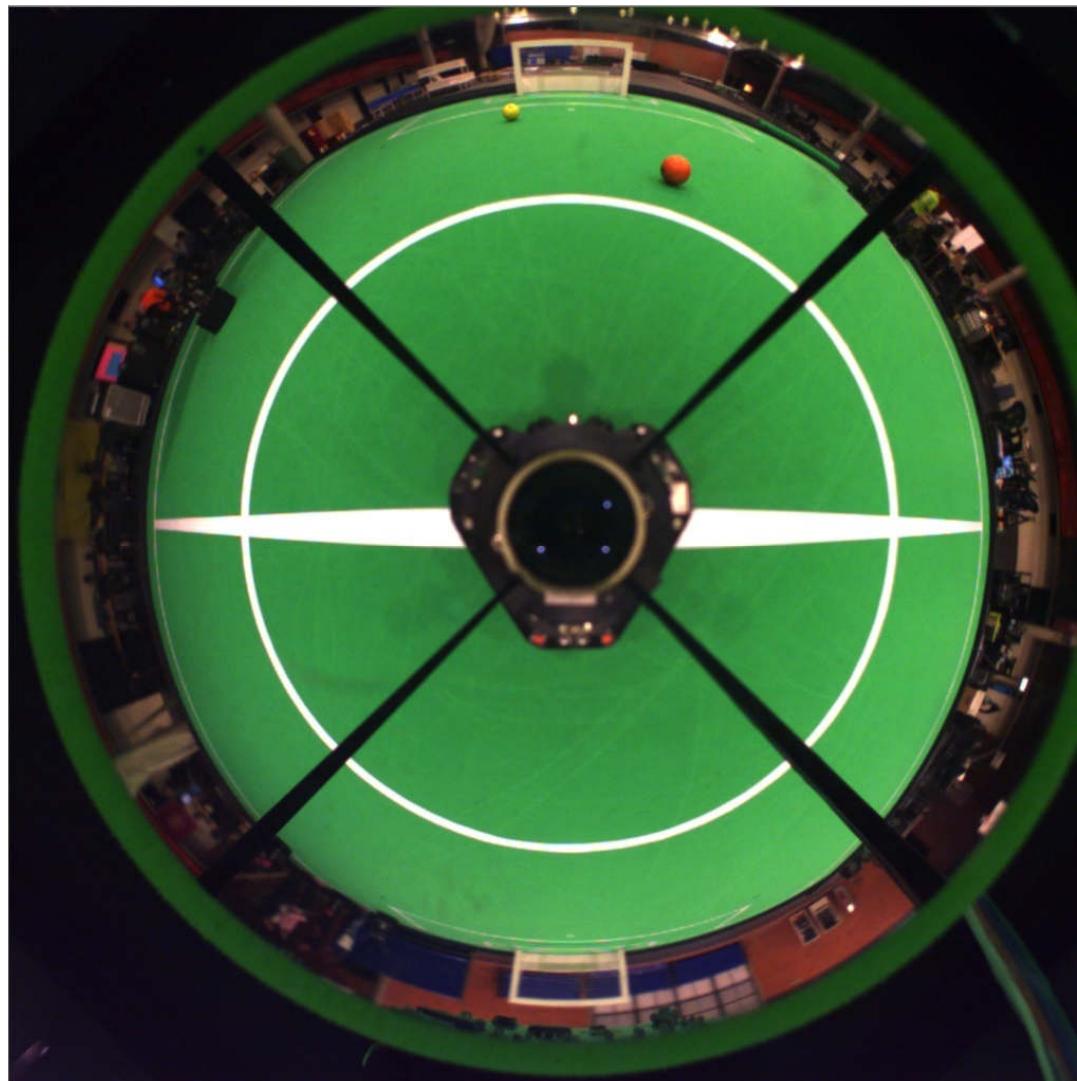
Vision System – a special example

- A catadioptric system is a vision system based on one camera and one mirror system



Vision System – a special example

- A catadioptric system is a vision system based on one camera and one mirror system



Vision System – a special example

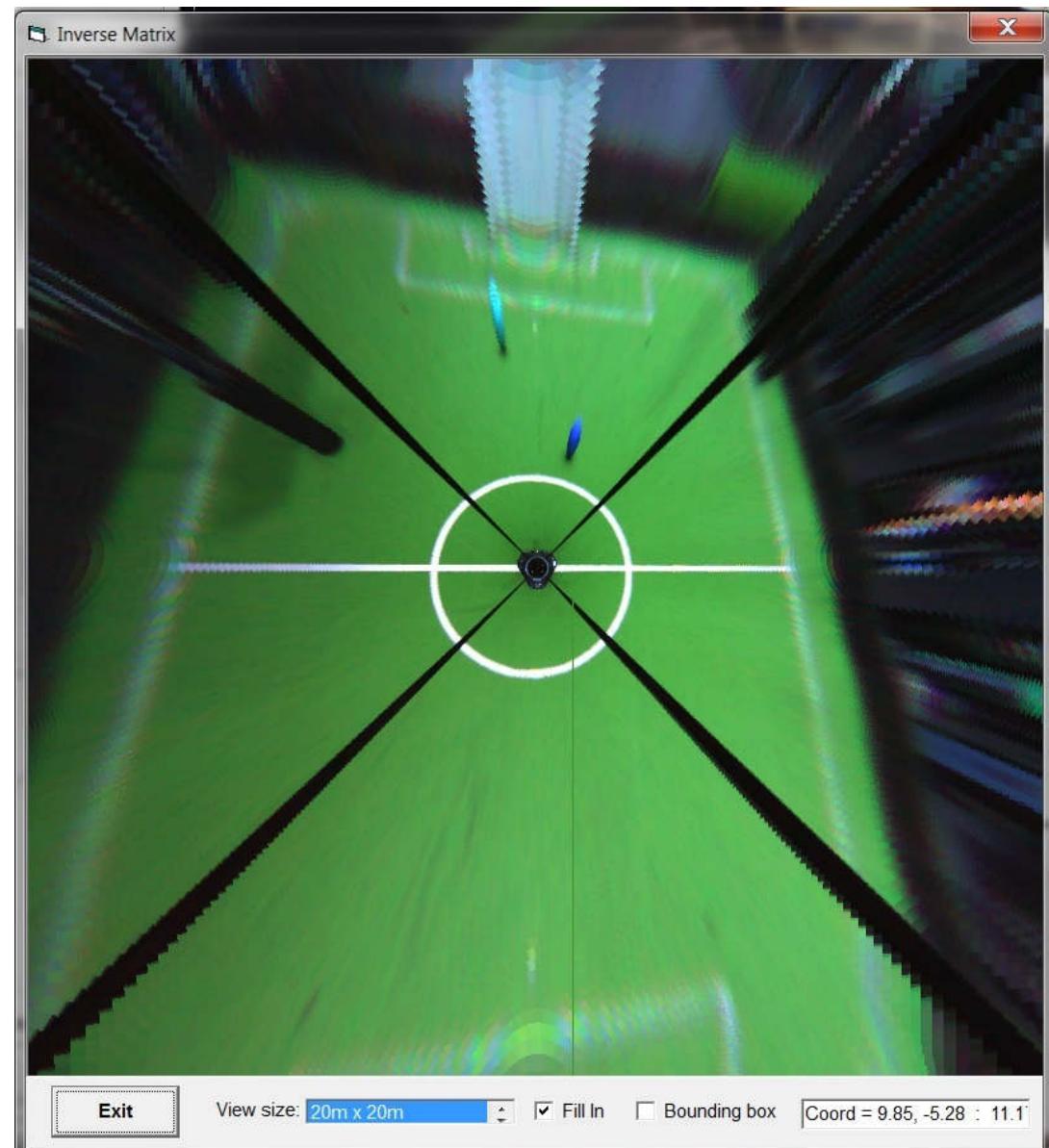
Such a system suffers from mechanical misalignments originating from several possible different reasons.

By using the camera model, the catadioptric model and estimating the geometry parameters that introduce the distortion, one can correct the original image to obtain a fairly accurate distance map at the ground level



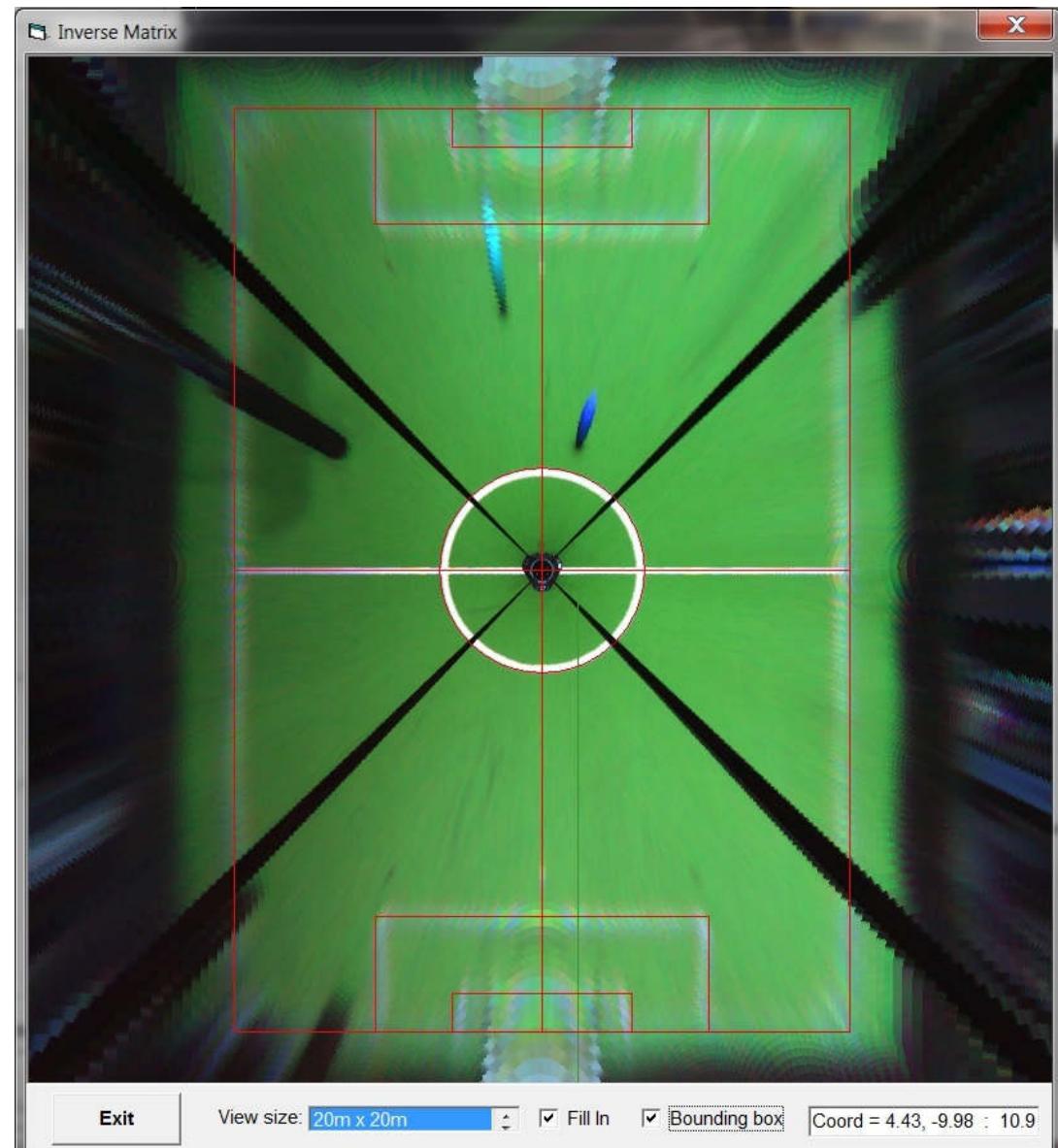
Vision System – a special example

- Setting all parameters to zero will normally result in a mapping distance matrix which is highly distorted
- the image on the right is an eagle eye view image obtained from the distance map in this case



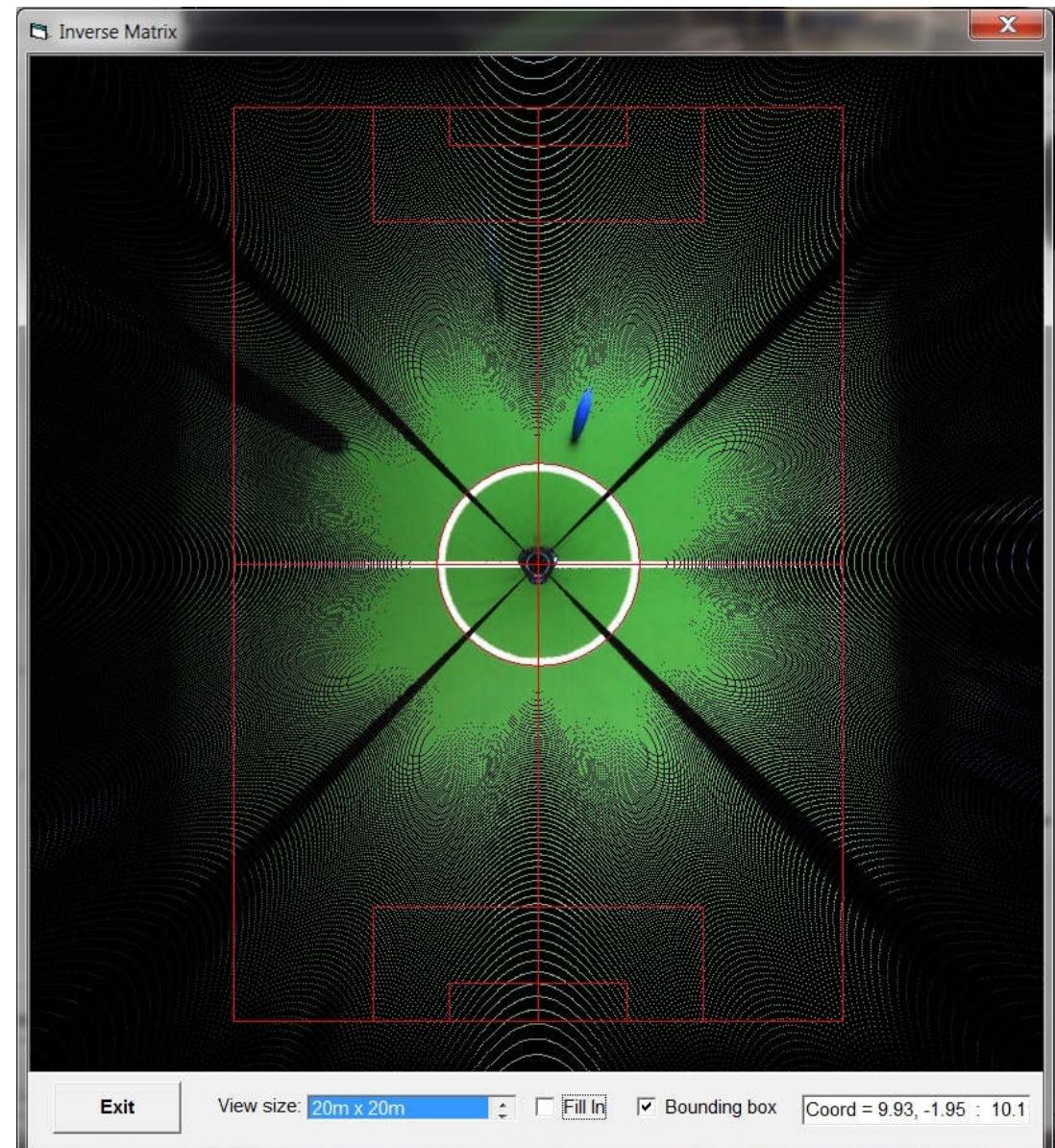
Vision System – a special example

- By adjusting the correction parameters one can obtain an “almost correct” distance matrix on the ground floor
- the image on the right is an eagle eye view image obtained from the distance map



Vision System – a special example

- The catadioptric effect highly affects the space resolution as we move from the center pixels to the peripheric ones
- the image on the previous slide is obtained by making a color integration among neighbor pixels according to its distance to the calculated resulting pixel coordinates



Some references

- Richard Hartley and Andrew Zisserman (2003). Multiple View Geometry in Computer Vision. Cambridge University Press. pp. 155–157. ISBN 0-521-54051-8.
- Z. Zhang, "A flexible new technique for camera calibration", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.22, No.11, pages 1330–1334, 2000
- Boris Peter Selby et al., "Patient positioning with X-ray detector self-calibration for image guided therapy", Australasian Physical & Engineering Science in Medicine, Vol.34, No.3, pages 391–400, 2011
- de Villiers, J. P.; Leuschner, F.W.; Geldenhuys, R. (17–19 November 2008). "Centi-pixel accurate real-time inverse distortion correction". 2008 International Symposium on Optomechatronic Technologies. SPIE. doi:10.1117/12.804771.
- Richard Hartley and Andrew Zisserman (2003). Multiple View Geometry in Computer Vision. Cambridge University Press. pp. 155–157. ISBN 0-521-54051-8.
- Sonka, M; Hlavac, V; Boyle, R (1995). Image Processing, Analysis & Machine Vision (2nd ed.). Chapman and Hall. pp. 14. ISBN 0-412-45570-6
- Ingrid Carlbom, Joseph Paciorek (1978). "Planar Geometric Projections and Viewing Transformations". ACM Computing Surveys 10 (4): 465–502. doi:10.1145/356744.356750
- Hecht, Eugene (1998). "5.7.6 The Camera". Optics (3rd ed.). ISBN 0-201-30425-2.

Fundamental Concepts of Computer Vision

Alina Trifan
António J. R. Neves

Electronics, Telecommunications and Informatics Department

University of Aveiro

alina.trifan@ua.pt
an@ua.pt
<http://sweet.ua.pt/an>

Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

Contents

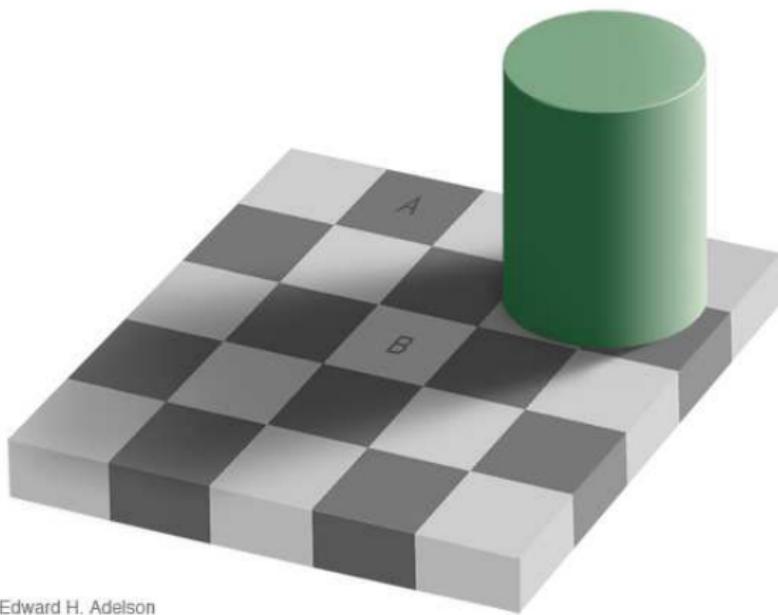
- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

- Vision is a complex physical and intellectual human task that stands as a primary interaction tool with the world.
- It is a complex process not completely understood, even after hundreds of years of research.
- The visualization of a physical process involves an almost simultaneous interaction of the eyes and the brain.
- This interaction is performed by a network of neurons, receptors and other specialized cells.

Introduction (2)

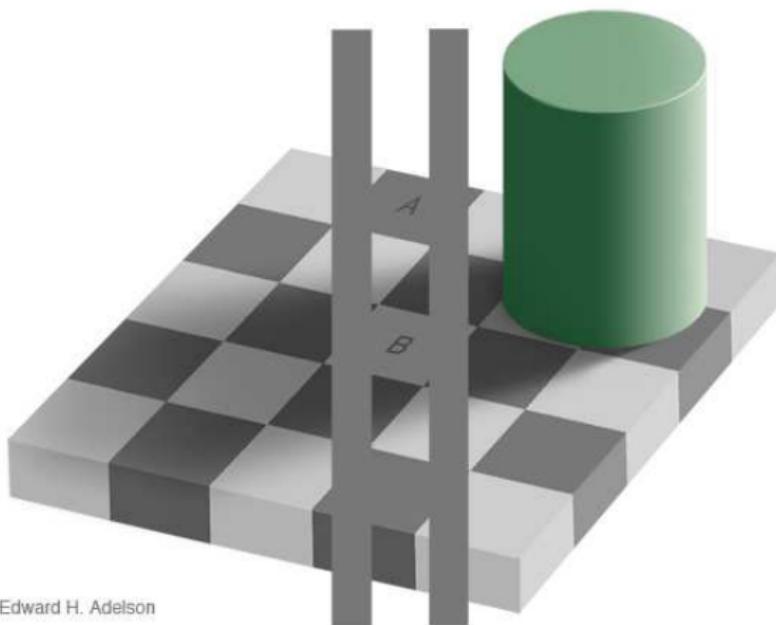
- The human eye is equipped with a variety of optical elements, including the cornea, iris, pupil, a variable lens and the retina.
- Can do amazing things like:
 - Recognize people and objects
 - Navigate through obstacles
 - Understand mood in the scene
 - Imagine stories
- But:
 - Suffers from illusions
 - Ignores many details
 - Ambiguous description of the world
 - Doesn't care about accuracy of world

Illusions (1)



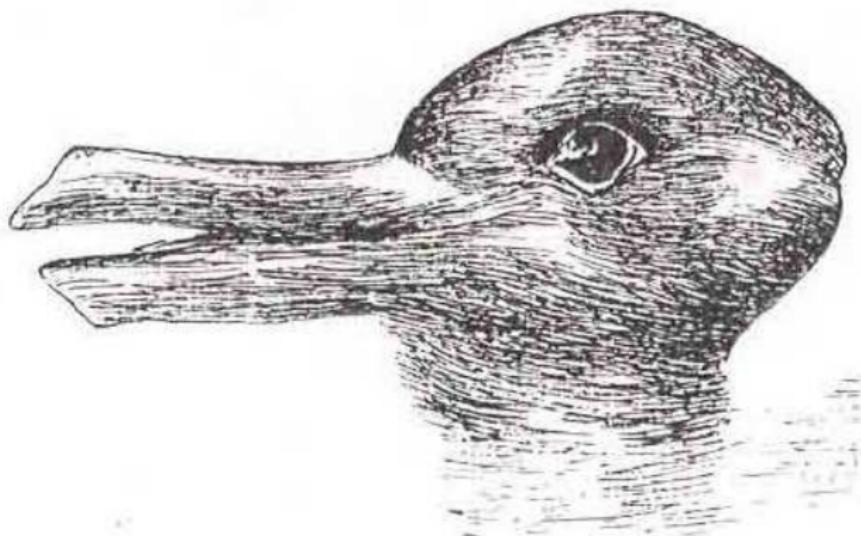
Edward H. Adelson

Illusions (2)



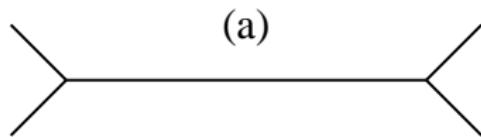
Edward H. Adelson

Illusions (3)

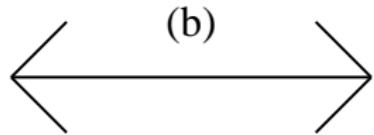


Other illusions...

- The human visual system exhibits a considerable cognitive component, influenced by memory, context, and intention:

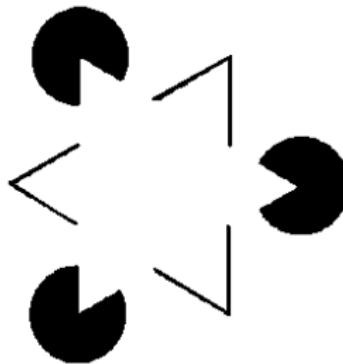


(a)



(b)

Which is the longer one?

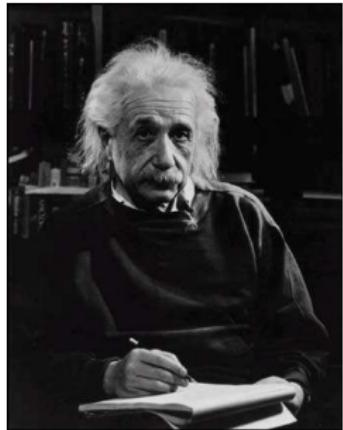


A triangle?

A picture is worth 1000 words



A picture is worth millions of words



- Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images ...
- Computer vision applications are increasing:
 - surveillance;
 - machine inspection;
 - medicine;
 - robotics;
 - entertainment;
 - media.
- The utopic goal: make computer vision converge towards human vision. Can we ever accomplish that?

- Computer vision seeks to develop algorithms that replicate one of the most amazing capabilities of the human brain - inferring properties of the external world purely by means of the light reflected from various objects to the eyes.
- With vision, it is possible to determine how far away objects are, how they are oriented with respect to the subject, and in relationship to various other objects.
- It is possible to guess their colors and textures and recognize them.
- It is possible to segment regions of space corresponding to particular objects and track them over time.
- In this class, we will see an overview of the concepts and algorithms used in Computer Vision to achieve the referred tasks ...

Some definitions (1)

- Image Processing
 - Signal processing where the input signal is an image (2 dimensional signal) and the output can be a transformation of this image or a set of characteristics associated to the image.
- Computer Vision
 - Techniques for image acquisition, extraction, characterization and interpretation of the information gathered from images of the 3D world.
- Machine Vision
 - Computer Vision for automation and robotic applications.
- Artificial Vision
 - Wide research field that includes all sciences and techniques that allow the study and application of all activities related to the use and interpretation of an image.

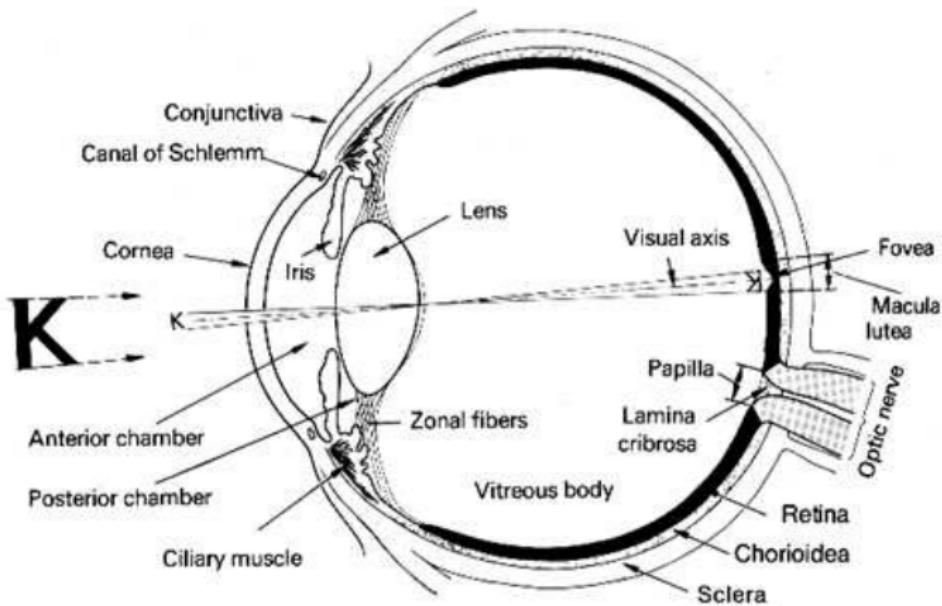
Some definitions (2)

- Perception
 - The process of acquiring an image.
- Pre-processing
 - Image correction(distortion, noise) or enhancement(filtering).
- Segmentation
 - Breaking an image into segments (areas of interest).
- Descriptors
 - Features of the image (shape, size . . .).
- Object Detection
 - **Where** is **this** object in the image?
- Object Recognition
 - **Which** object is depicted in the image?
- Image understanding
 - . . .

Contents

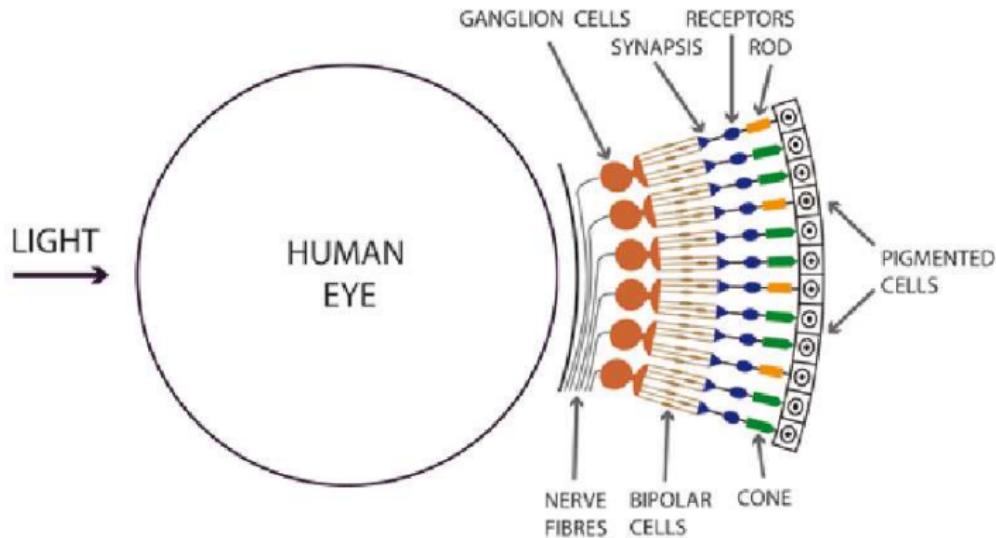
- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

Human vision

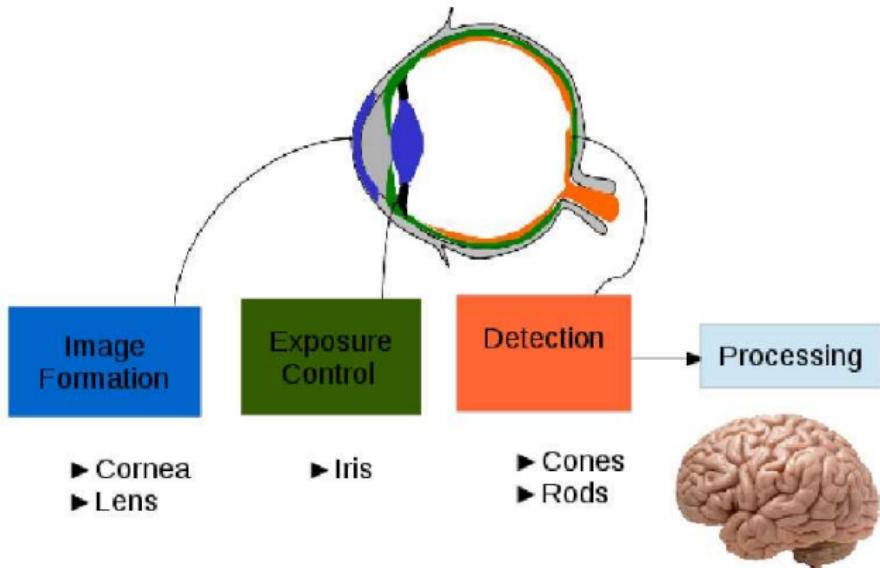


Human vision

Basic Cross section of the Eye - Showing the Rods and Cones



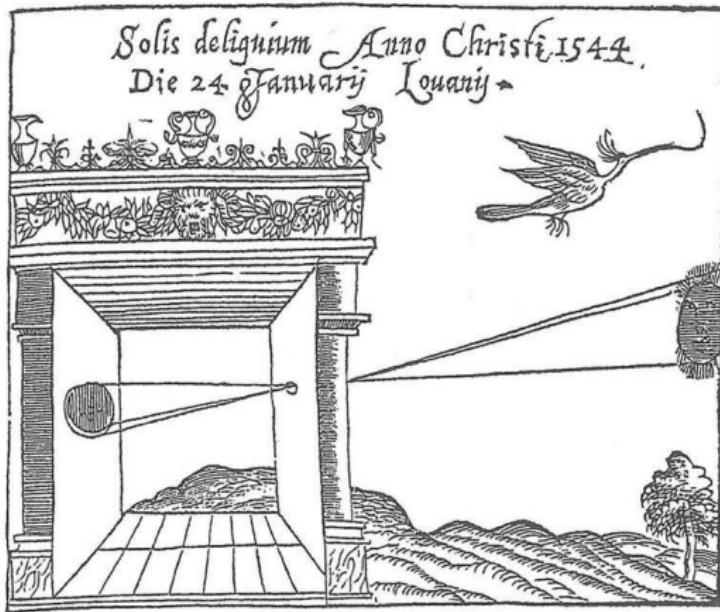
Human vision



Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation**
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

History of cameras (1544)



Camera Obscura, Gemma Frisius, 1544

History of cameras (1568)

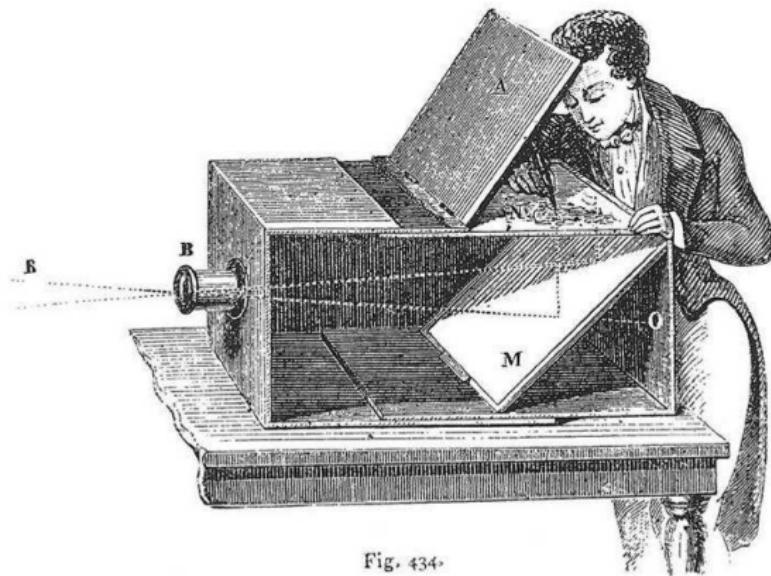


Fig. 434.

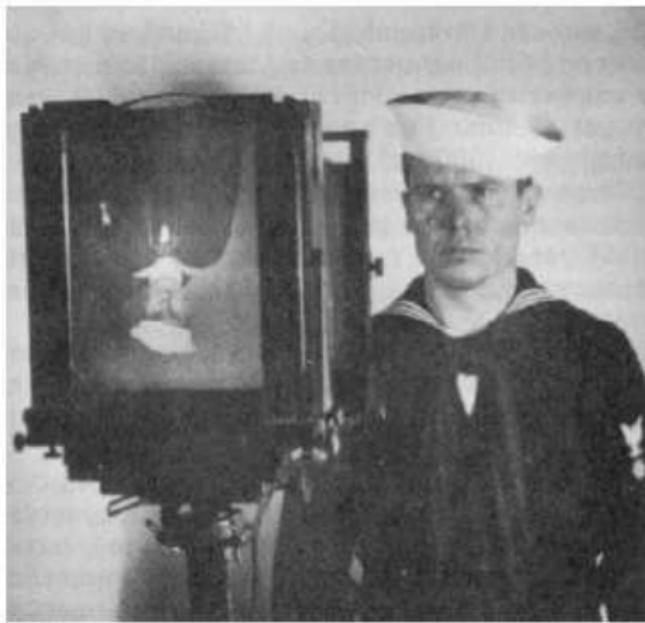
Lens Based Camera Obscura, 1568

History of cameras (1837)



Still Life, Louis Jaques Mande Daguerre, 1837

History of cameras (1930)

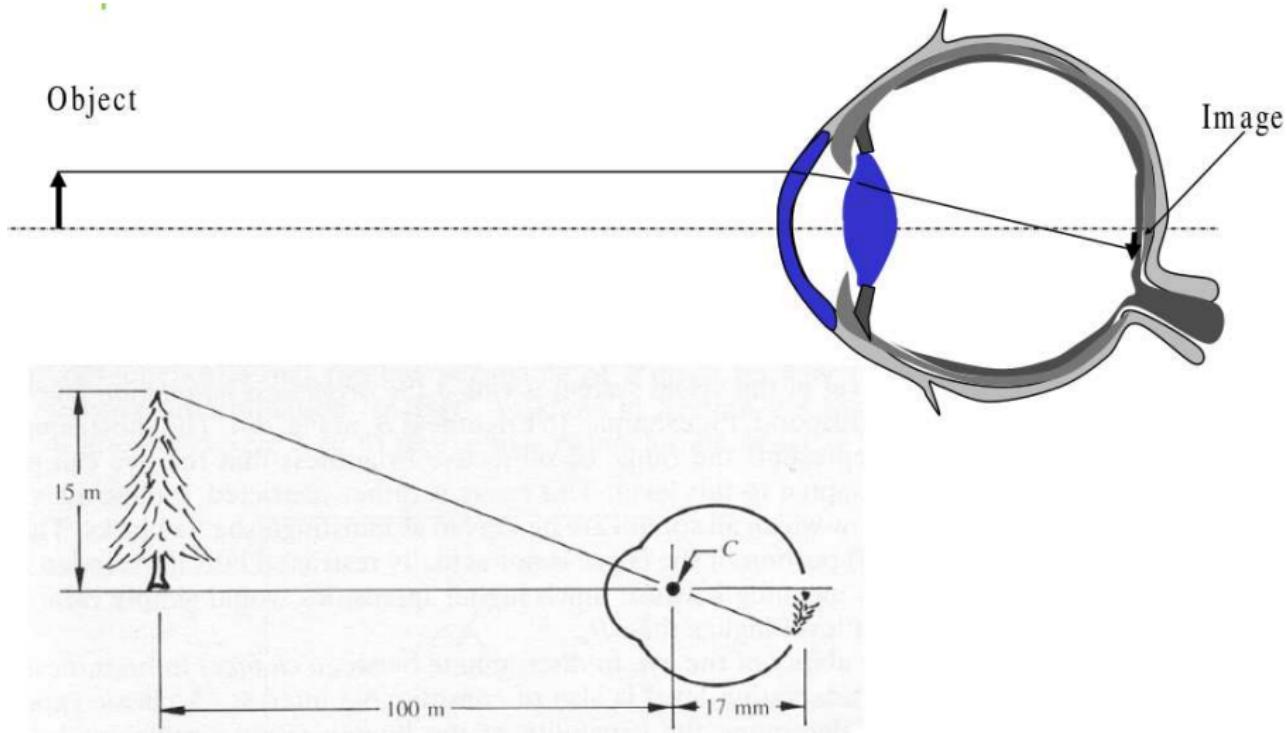


History of cameras (1970 - nowadays)



Silicon Image Detector, 1970 - digital cameras

Human eye



Pinhole Camera Model

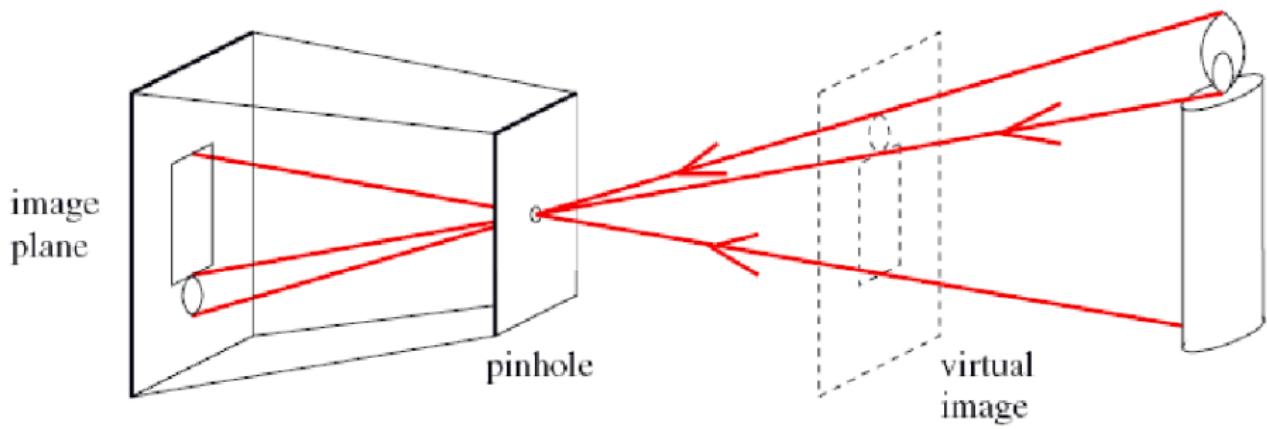
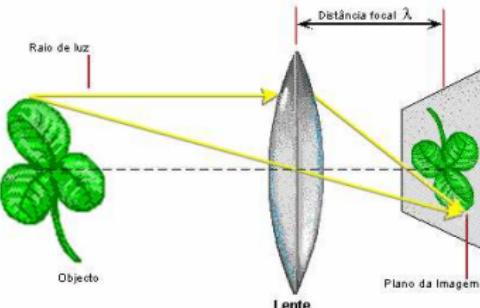


Image through a lens



- All the rays of light that come from an object in direction to the lens converge, on the other side, in another point at a certain distance from the lens. This distance is called **focal distance**.
- f smaller – wide-angle camera; f gets larger – more telescopic.
- All the points that verify this fact are denoted the **focal plane**.
- There are some other important parameters related to lens: Field of View, Depth of Field, ...

Basic Camera Geometry

- Far objects appear smaller.
- Lines project to lines.
- Lines in 3D project to lines in 2D.
- Distances and angles are not preserved.
- These geometric properties are “common sense”. Other properties can be inferred if we formalize the model using
... Mathematics, of course...

Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras**
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

Digital camera

- Image acquisition using a digital camera:

(IEEE SP Magazine, Jan 2005)

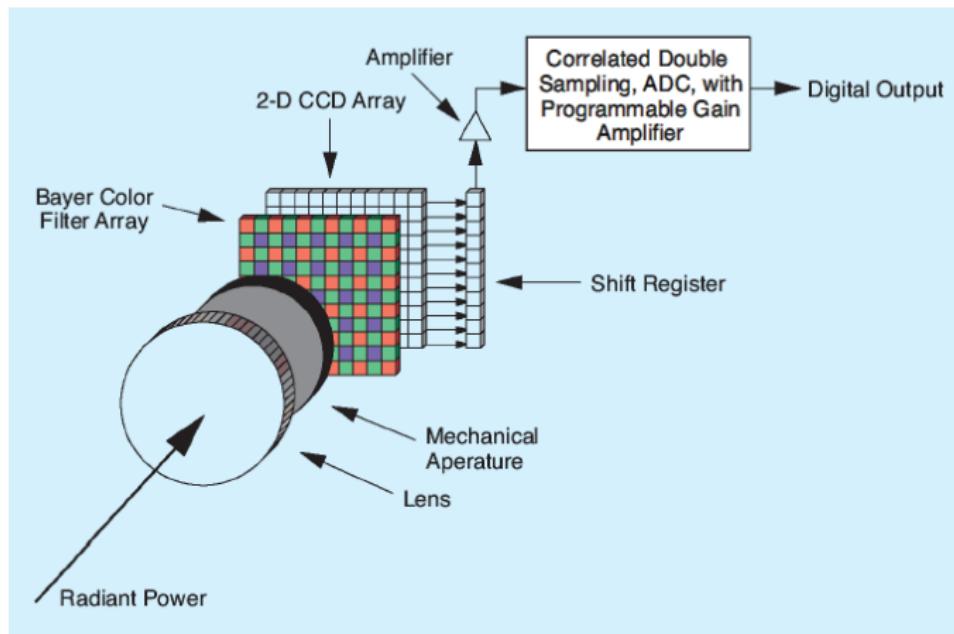
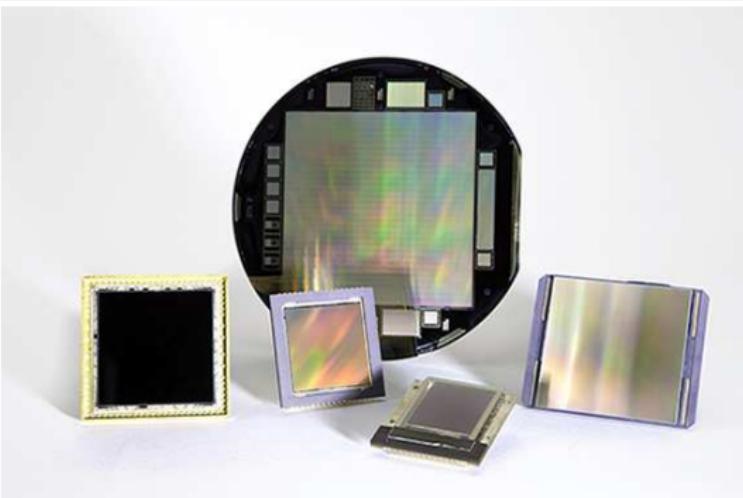
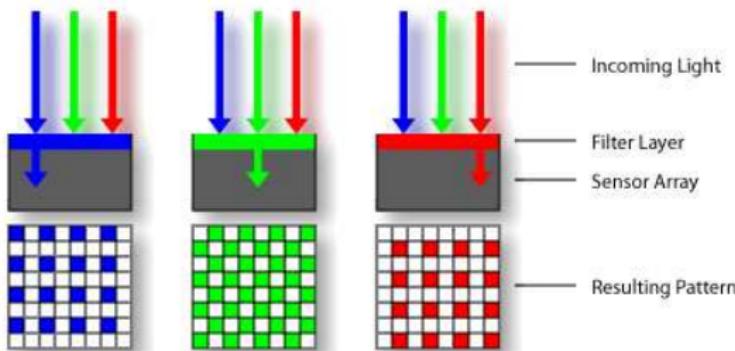
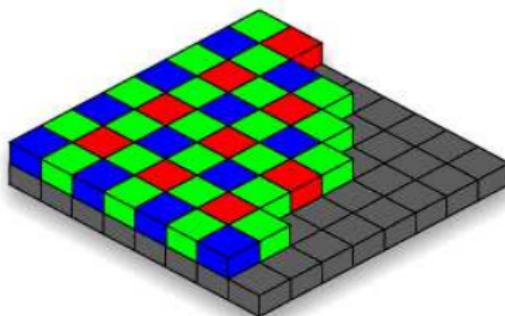


Image sensors



- Some considerations: speed, resolution, cost, signal/noise ratio, ...
- **CCD - charge coupled device** - Higher dynamic range, High uniformity, Lower noise.
- **CMOS - Complementary Metal Oxide Semiconductor** - Lower voltage, Higher speed, Lower system complexity.

The Bayer matrix



Digital cameras - several solutions



Digital cameras - several solutions

- Several interfaces (Firewire, GigE, CameraLink, USB, ...).
- Scientific usage (high resolution, long exposure time, ...).
- High speed (ex. 1000 fps).
- Linear (ex. 10000 lines per second).
- 3D
- Infrared (ex. 8 to 14 μm).
- High dynamic range (ex. using a prism and two sensors).
- Multispectral

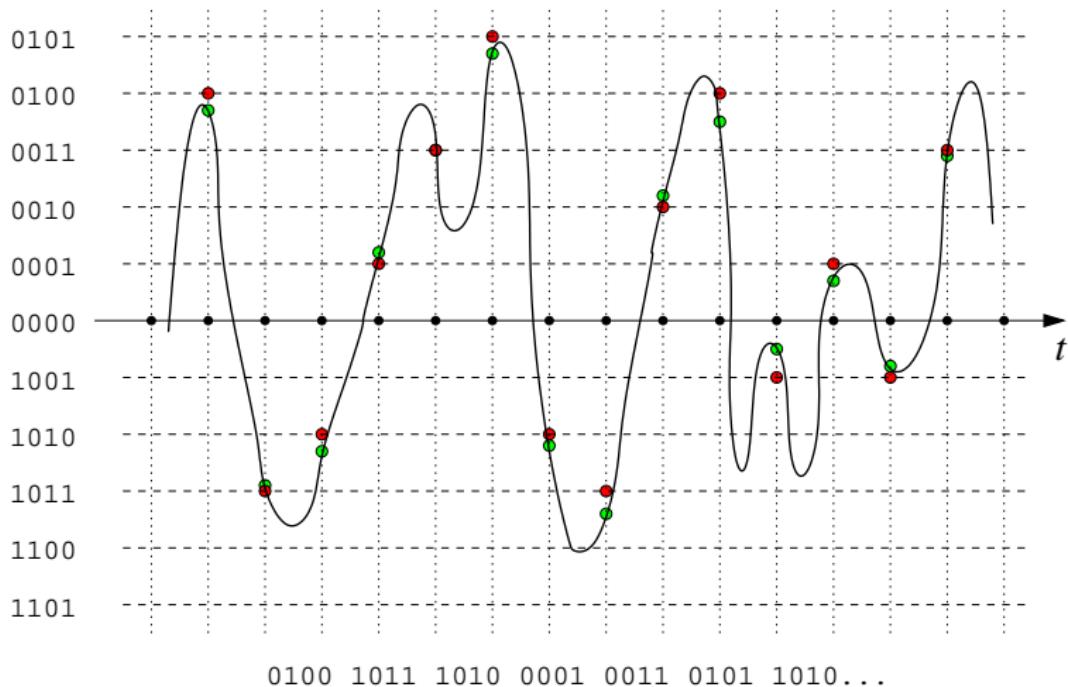
Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

Sampling and quantization

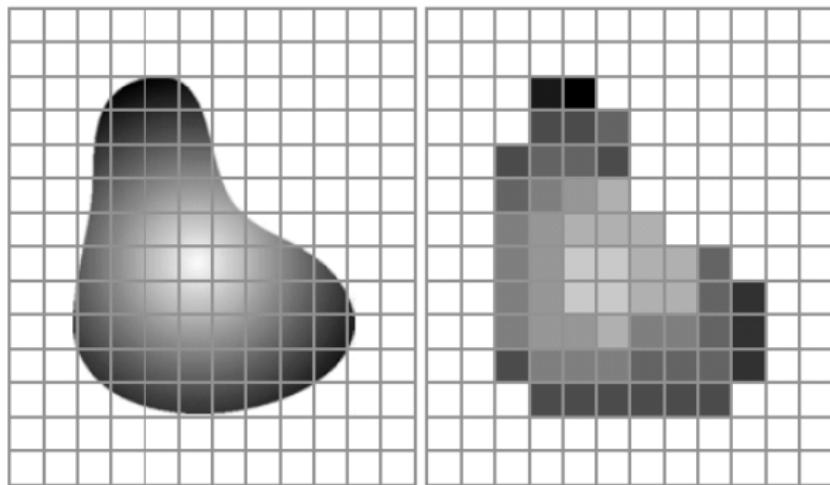
- Generally, an image can be represented by a two-dimensional function, $f(x, y)$, where x and y are spatial coordinates.
- The meaning of f in a given point in space, (x, y) , depends on the source that generated the image (visible light, x-rays, ultrasound, radar, . . .).
- Nevertheless, we generally assume that $f(x, y) \geq 0$.
- Moreover, both the spatial coordinates and the function values are continuous quantities.
- Therefore, to convert $f(x, y)$ into a digital image, it is necessary to perform **spatial sampling** and **amplitude quantization**.

Digitalization: sampling + quantization



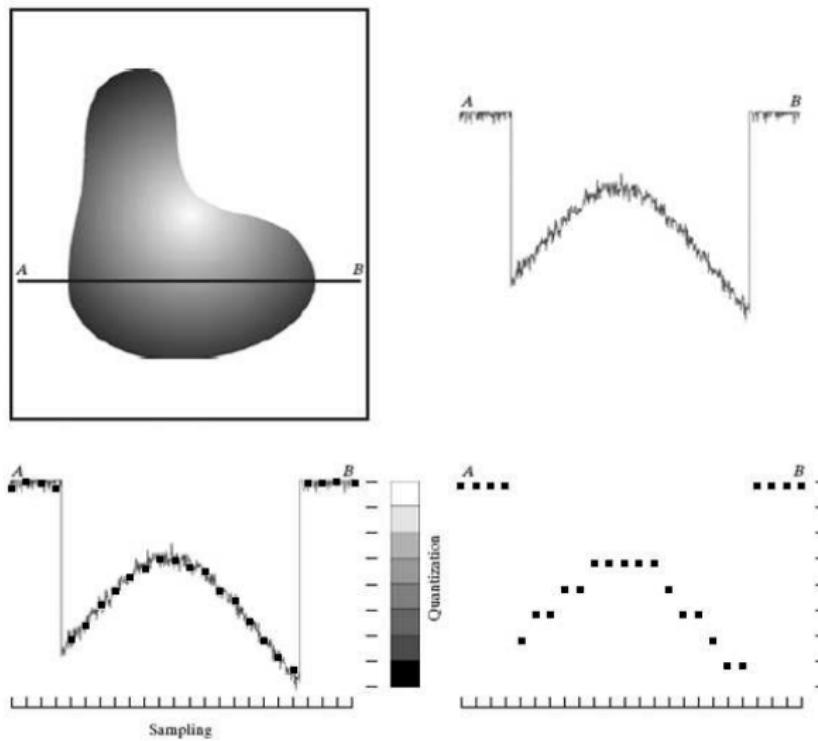
Sampling and quantization

- Sampling and quantization — example:
(Gonzalez & Woods)



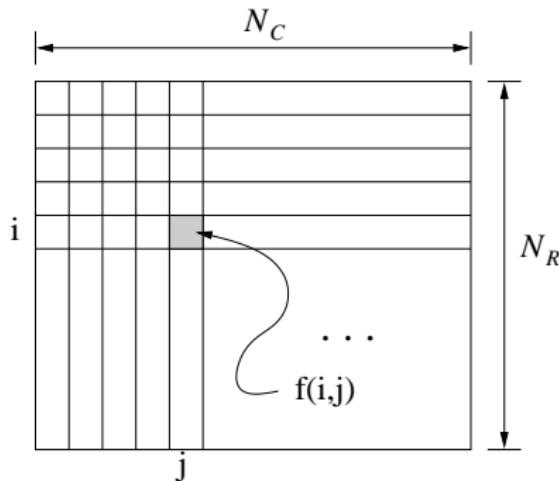
Sampling and quantization

- Sampling and quantization — example:
(Gonzalez & Woods)



Digital images

- Typically, a **digital image** is represented by a rectangular matrix of scalars or vectors.



- The $f(i,j)$ are named *pixels* and, usually, $f(i,j) \in \mathcal{I} \subset \mathbb{N}_0^n$.

- We will consider digital images of the following types:
 - Black and white (binary images).
 $f(i,j) \in \{0, 1\}$
 - Grayscale images.
 $f(i,j) \in \{0, 1, \dots, 2^b - 1\}$
 - Color-indexed images.
 $f(i,j) \in \{0, 1, \dots, 2^b - 1\} \xrightarrow{\alpha} \mathcal{I} \subset \{0, 1, \dots, 2^{b'} - 1\}^3$
 - Color images (for example, RGB images).
 $f(i,j) \in \{0, 1, \dots, 2^b - 1\}^3$

Examples



Color



Color-indexed (256)



Grayscale (256)



Black and white

Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces**
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

The visible spectrum



Spectral colors (pure colors)

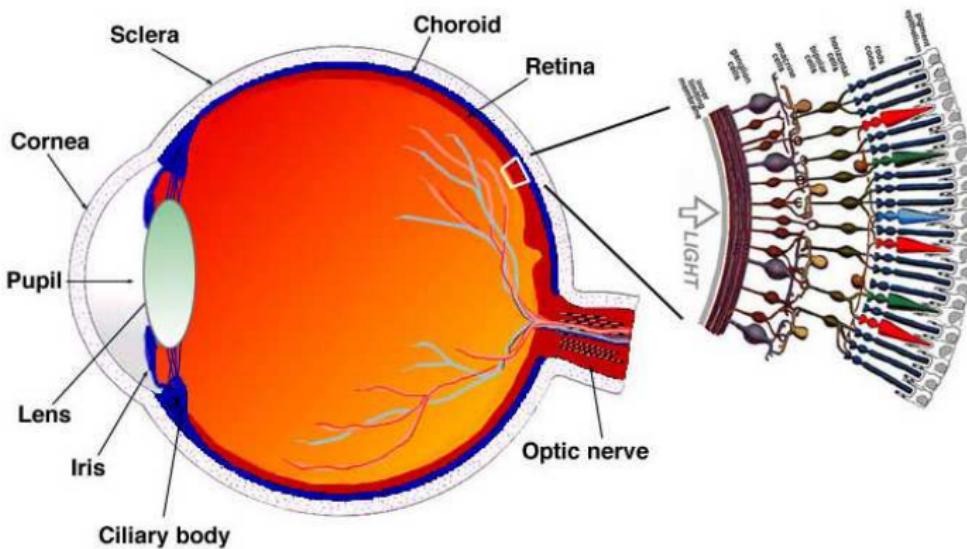
Cor	Wavelength
Violet	≈ 380–440 nm
Blue	≈ 440–485 nm
Cyan	≈ 485–500 nm
Green	≈ 500–565 nm
Yellow	≈ 565–590 nm
Orange	≈ 590–625 nm
Red	≈ 625–740 nm

The human perception of color

- Normally, the characteristics that allow colors to be distinguished are:
 - The **brightness** (how bright is the color).
 - The **hue** (the dominant color).
 - The **saturation** (how pure is the color).
- Together, the hue and the saturation define the **chromaticity**.
- Therefore, a color can be characterized by the brightness and the chromaticity.

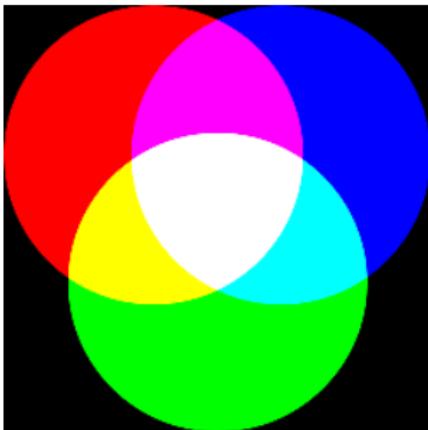
The human perception of color

- The human eye has **photoreceptors** that are sensitive to short wavelengths (*S*), medium wavelengths (*M*) and long wavelengths (*L*), also known as the blue, green and red photoreceptors.



Additive primaries

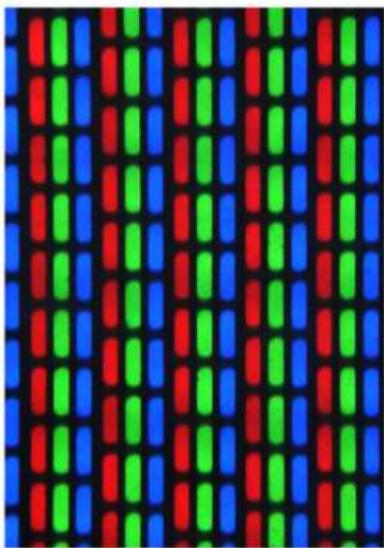
- The red, green and blue are the three additive primary colors.



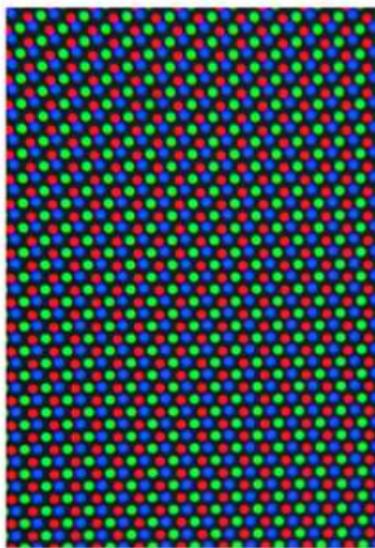
- Adding these three colors produces white.

The *RGB* color space

- Besides the use in acquisition on digital cameras, for example, the displays have pigments of these three colors...



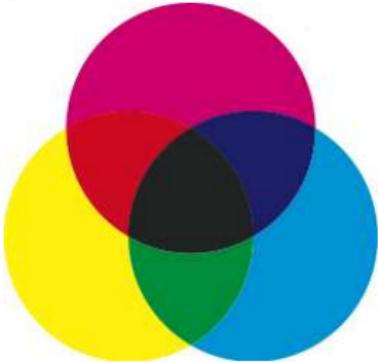
21" TV CRT Display



17" PC CRT Display

The CMY color space

- The **CMY** color space is based on the subtractive properties of inks.
- The cyan, magenta and yellow are the subtractive primaries. They are the complements, respectively, of the red, green and blue. For example, the cyan subtracts the red from the white.



- Conversion from *RGB* to *CMY*: $C = 1 - R$, $M = 1 - G$, $Y = 1 - B$.

The CMY color space



C component



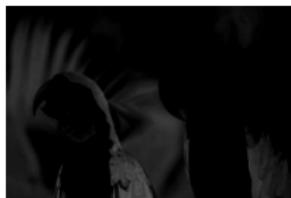
M component



Y component

The CMYK color space

- Due to technological difficulties regarding the reproduction of black, the **CMYK** color space is generally used for printing.



C component



M component



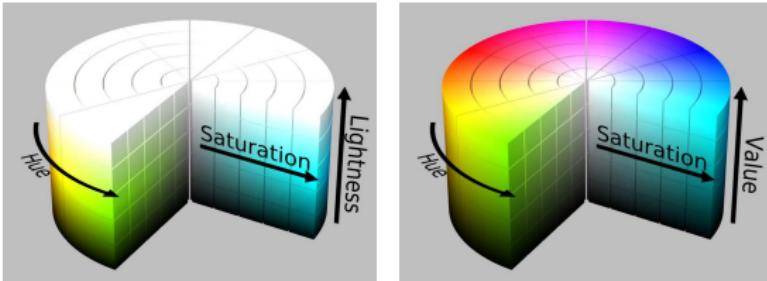
Y component



K component

The *HLS* and *HSV* color spaces

- The **HSL** and **HSV** are the two most common cylindrical coordinate representations of colors.
- They rearrange the geometry of RGB colors in an attempt to be more intuitive and perceptually relevant than the cartesian (cube) representation.
- They were developed in the 1970s for computer graphics applications, and are used for color pickers, in color-modification tools in image editing software, and commonly for image analysis and computer vision.



The *HLS* and *HSV* color spaces

- RGB to HSV:

$$V = \max(R, G, B)$$

$$S = \begin{cases} \frac{V - \min R, G, B}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H = \begin{cases} 60(G - B)/S & \text{if } V = R \\ 120 + 60(B - R)/S & \text{if } V = G \\ 240 + 60(R - G)/S & \text{if } V = B \end{cases}$$

The *HLS* and *HSV* color spaces

- RGB to HSL:

$$V_{max} = \max R, G, B$$

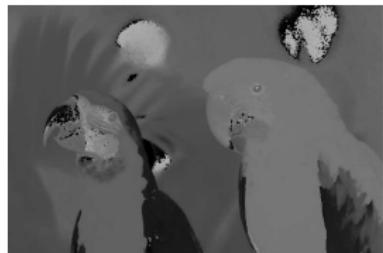
$$V_{min} = \min R, G, B$$

$$L = \frac{V_{max} + V_{min}}{2}$$

$$S = \begin{cases} \frac{V_{max} - V_{min}}{V_{max} + V_{min}} & \text{if } L < 0.5 \\ \frac{V_{max} - V_{min}}{2 - (V_{max} + V_{min})} & \text{if } L \geq 0.5 \end{cases}$$

$$H = \begin{cases} 60(G - B)/S & \text{if } V_{max} = R \\ 120 + 60(B - R)/S & \text{if } V_{max} = G \\ 240 + 60(R - G)/S & \text{if } V_{max} = B \end{cases}$$

The *HLS* and *HSV* color spaces



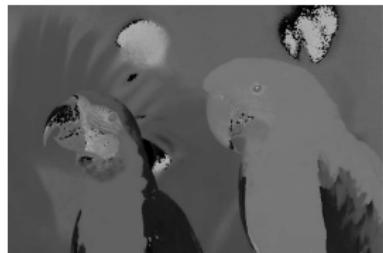
H component



S component



V component



H component



S component



L component

The YUV color space

- The YUV color space is used in some television standards.
- Y is the luminance component:

$$Y = 0.299R + 0.587G + 0.114B$$

- Components U and V represent the chrominance:

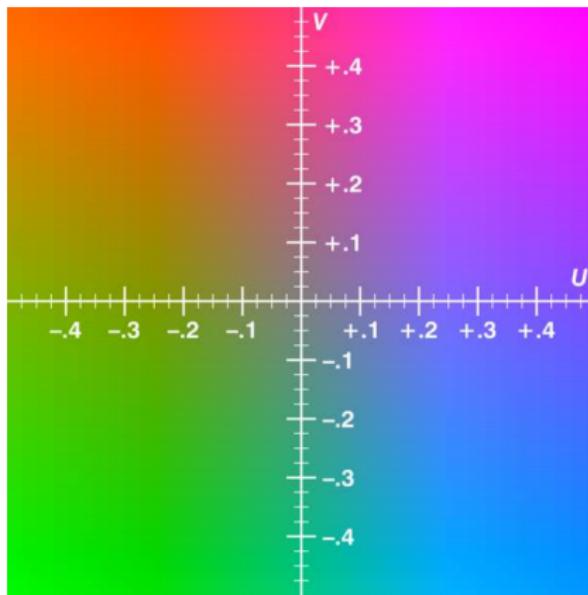
$$U = -0.147R - 0.289G + 0.436B = 0.492(B - Y)$$

$$V = 0.615R - 0.515G - 0.100B = 0.877(R - Y)$$

- For $R, G, B \in [0, 1]$, we have $Y \in [0, 1]$,
 $U \in [-0.436, 0.436]$ and $V \in [-0.615, 0.615]$.

The YUV color space

- $U - V$ plane, for a constant value of Y , equal to 0.5:



Advantages of the YUV color space

- The YUV color space allowed to maintain the compatibility with the old “black and white” television receivers.
- The human eye is more sensitive to the green color, which is represented mainly by the Y component.
- The U and V components are related to the blue and red.
- Since the human eye is less sensitive to the blue and red, it is possible to reduce the bandwidth used to represent the U and V components, without introducing significant perceptual degradation.

The YC_bC_r color space

- This is usually designated the digital version of YUV .
- The JPEG standard, as well as some other MPEG video standards, allows all 256 values in an 8 bits per component representation.
- In this case, considering $R, G, B \in \{0, \dots, 255\}$, we have:

$$Y = 0.299R + 0.587G + 0.114B$$

$$C_b = 128 - 0.168736R - 0.331264G + 0.5B$$

$$C_r = 128 + 0.5R - 0.418688G - 0.081312B$$

- After the conversion, $Y, C_b, C_r \in \{0, \dots, 255\}$.
- Besides its use in image and video coding, this color space is also used in some computer vision applications.

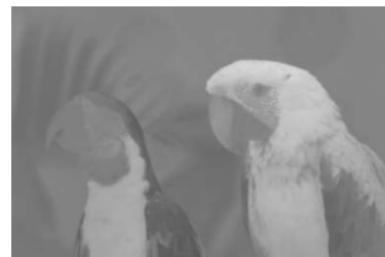
The YC_bC_r color space



Y component



C_b component



C_r component

Contents

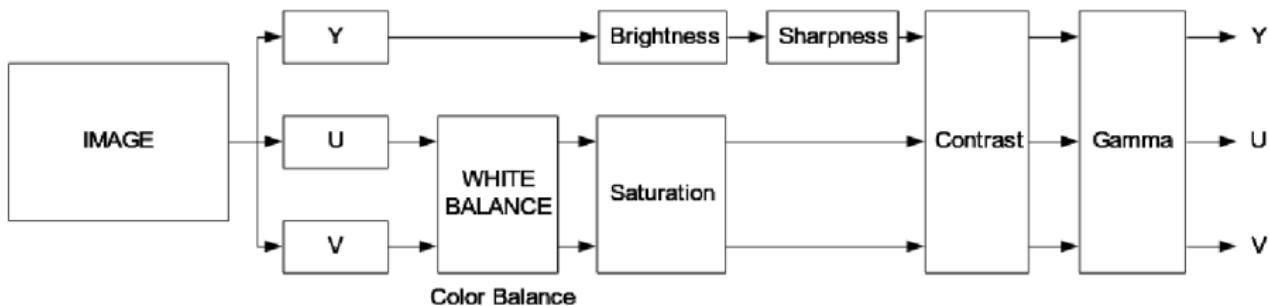
- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration**
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

Camera calibration

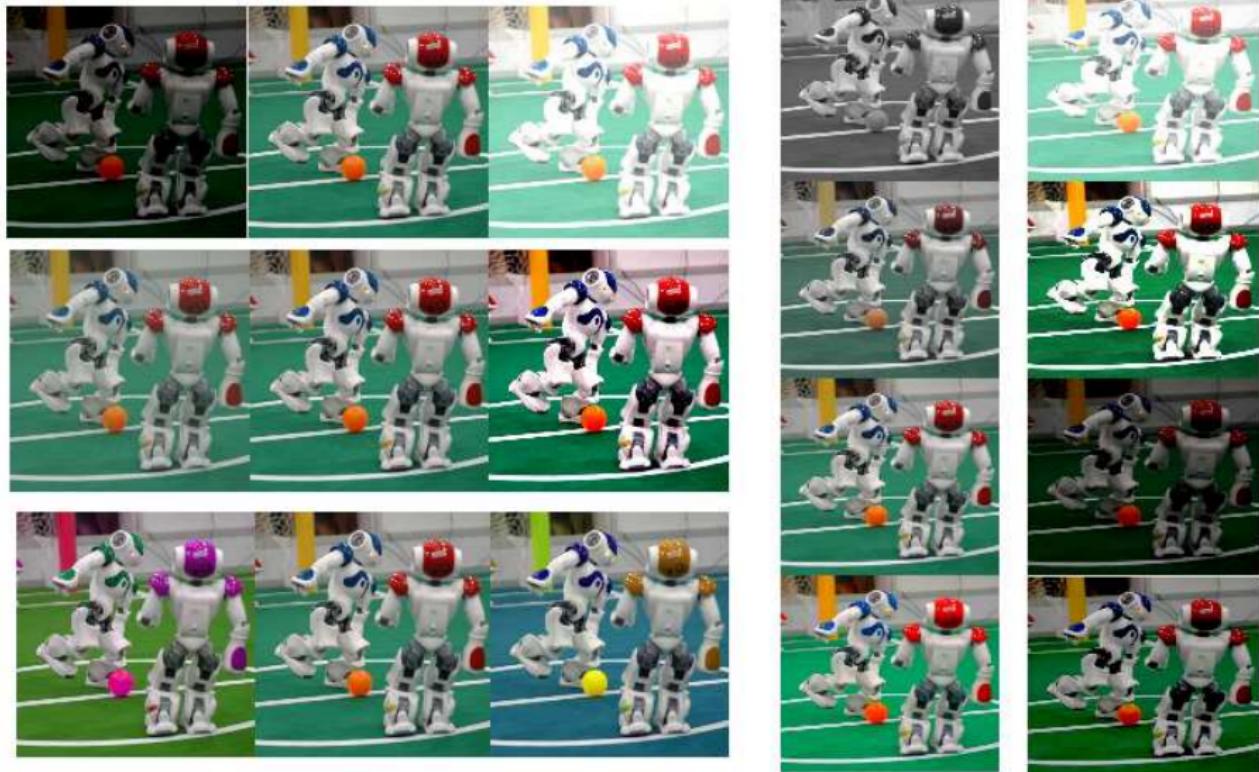
- In order to use a digital camera in some applications, it is necessary to calibrate some parameters.
- Colormetric parameters: the parameters that are related to color and intensity of the acquired image (gain, white-balance, brightness, sharpness, ...). The available parameters depends on the image processing pipeline of each camera.
- Extrinsic parameters: the parameters that define the location and orientation of the camera reference frame with respect to a known world reference frame.
- Intrinsic parameters: the parameters necessary to link the pixel coordinates of an image point with the corresponding coordinates in the camera reference frame.

Colormetric parameters (1)

A typical image processing pipeline (inside the image device) for a tri-stimulus system is shown below. This processing can be performed on the YUV or RGB components depending on the system. This should be understood as a mere example.



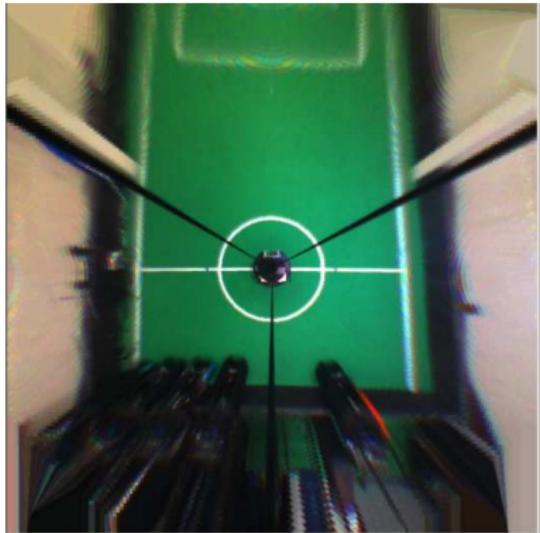
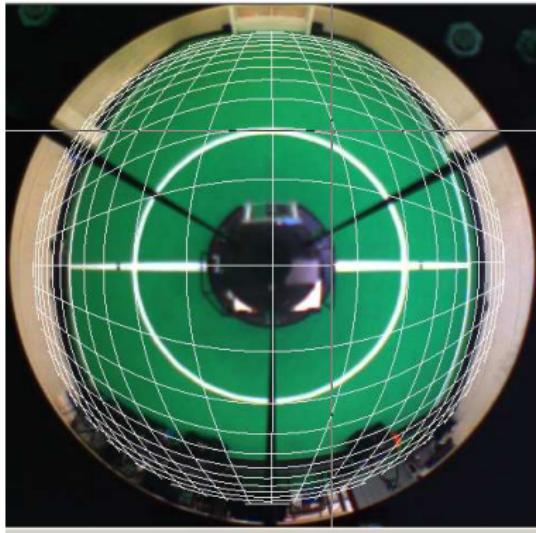
Colormetric parameters (2)



Extrinsic and Intrinsic parameters

- **Extrinsic parameters** denote the coordinate system transformations from 3D world coordinates to 3D camera coordinates.
- They reflect the projection view of the camera taking into consideration its relative position to the world coordinates (translation operation) and its rotations when we consider the camera coordinate system three axis in relation to the world coordinate system three axis.
- **Intrinsic parameters** include focal length, image format, principal point, skew.

Example of intrinsic and extrinsic calibration

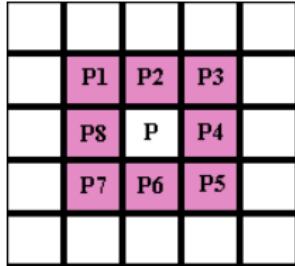
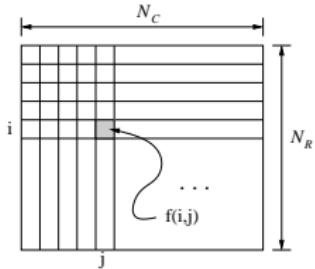


Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering**
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

Pixel Neighbours

- Many image processing operations make use of spatial relationships between pixels.
- A number of methods have been devised to specify pixel neighbors and calculate distance.
- The 4-neighbors of a pixel (x,y) are the closest pixels in horizontal and vertical directions (D4).
- The 8-neighbors are the 4-neighbors plus the four closest pixels in diagonal direction (D8).
- Diagonal only (DN).



- A group of pixels is said to be 4-connected if every pixel is 4-connected to the group.
- A group of pixels is said to be 8-connected every pixel is 8-connected to the group.

- The distance between pixels (x,y) and (u,v) can be calculated in several ways:
 - Euclidean (L2): $D = [(x - u)^2 + (y - v)^2]^{1/2}$
 - City-block (L1): $D = |x - u| + |y - v|$
 - Chessboard (Linf): $D = \max(|x - u|, |y - v|)$
- Although Euclidean distance is more accurate, the sqrt makes it expensive to calculate.

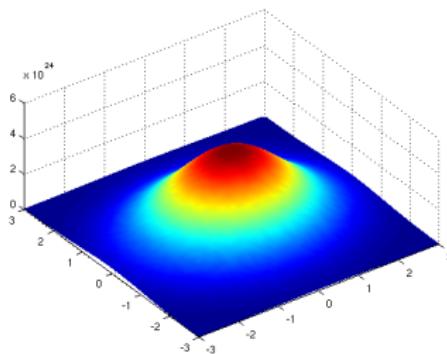
Spacial filtering

- Spatial filters make use of a fixed sized neighborhood in an input image to calculate output intensities.
- Linear filters use a weighted sum of pixels in the input image $f(i, j)$ to calculate the output pixel $g(i, j)$. In most cases, the sum of weights is one, so the output brightness = input brightness.
- Nonlinear filters can not be calculated using just a weighted sum (sqrt, log, sorting, selection).
- We can formalize the phrase “weighted sum of pixels” using correlation and convolution.
- The mathematical model is the discrete convolution operator based on the kernel h :

$$g(i, j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} h(i - m, j - n) f(i, j)$$

Examples of filters (1)

- Average - the easiest spatial filter to implement. The kernel is a matrix with all the values equals to one (the pixel is replaced by an average of the $N \times M$ neighbors). This filter smooths an image and removes noise and small details.
- Binomial - uses Binomial coefficients as weights to give more emphasis to pixels near the center of the $N \times M$ neighborhood.
- Gaussian - uses the Gaussian function to define the neighborhood weights.

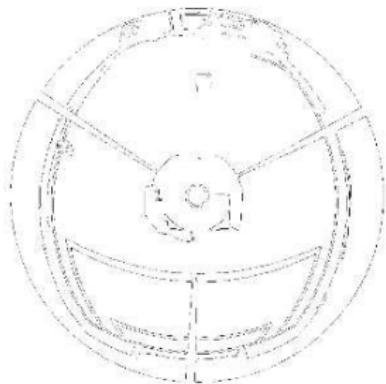


Example of filters (2)

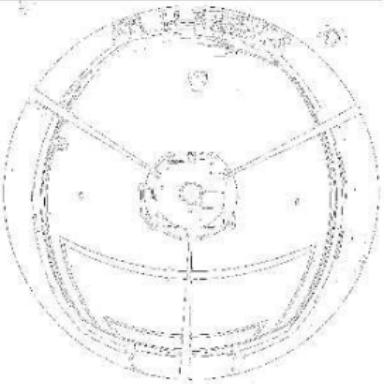


Edge detection

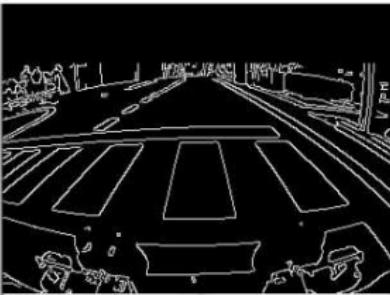
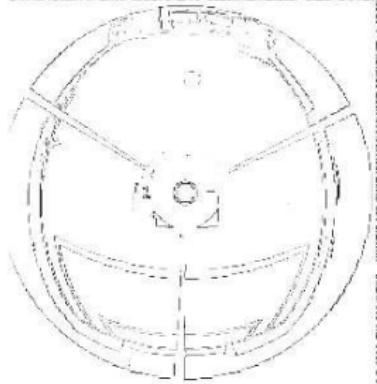
Canny



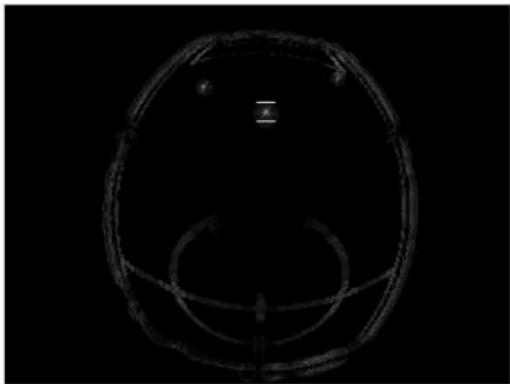
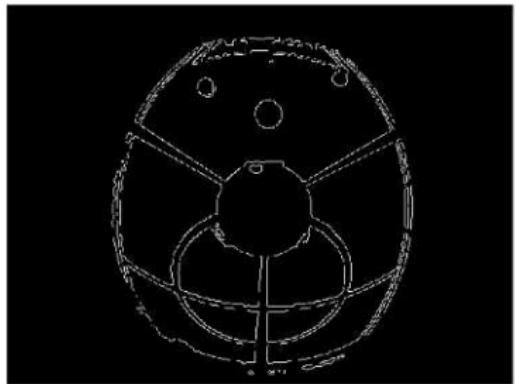
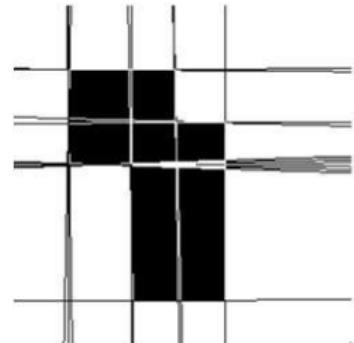
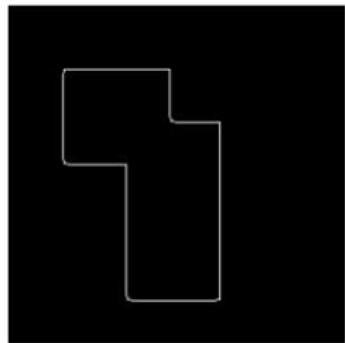
Laplace



Sobel



Hough Transform



Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms**
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

Histograms: definition

- In statistics, a histogram is a graphical display of tabulated frequencies.
- Typically represented as a bar chart.

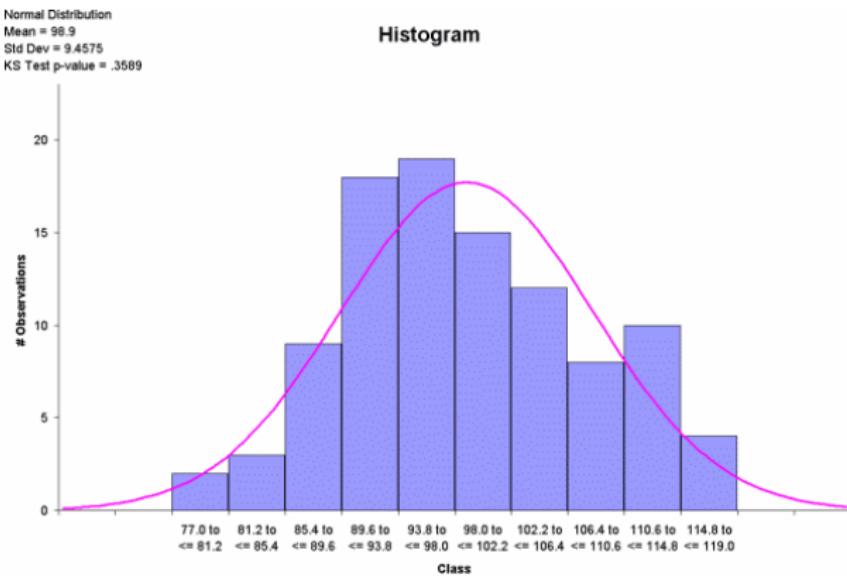
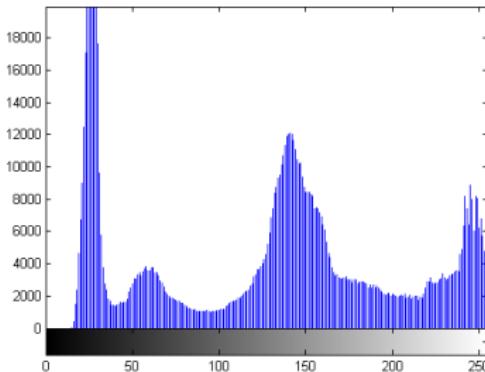
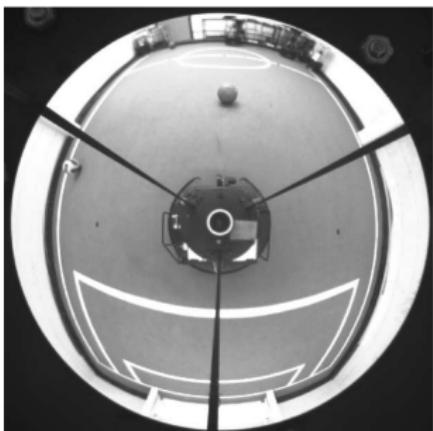


Image histograms

- In images, allow us to see the color or intensity distribution.
- The collected counts of data can be organized into a set of predefined bins.
- It is also possible to count image features that we want to measure (i.e. gradients, directions, etc).
- Some important parts of an histogram:
 - dims: The number of parameters you want to collect data.
 - bins: The number of subdivisions in each dim.
 - range: The limits for the values to be measured.
- If we want to count two features, the resulting histogram would be a 3D plot (in which x and y would be bin_x and bin_y for each feature and z would be the number of counts for each combination of (bin_x, bin_y)).

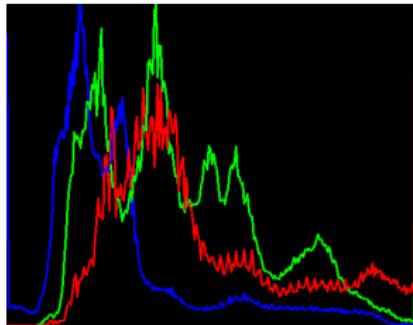
Histograms: example (1)

- Example of an histogram obtained from a grayscale image.
- Each bin shows the number of times each one of the gray values are present in the image.



Histograms: example (2)

- Example of an histogram showing the distribution of the colors on an image.



Histograms: operations

- Histogram operations are designed to enhance the visibility of objects of interest in an image.
- Histogram Equalization - improves the contrast in an image, in order to stretch out the intensity range.
- Local Histogram Equalization - increase the amount of enhancement by looking at local intensity properties (dividing an image into regions and perform histogram equalization on each sub-image or using local statistics).
- Histogram Comparison - get a numerical parameter that expresses how well two histograms match each other (ex. Correlation, Chi-Square, Intersection, . . .).
- Sum, subtract, . . .

Histograms: equalization

- Goal of histogram equalization is to reshape the image histogram to make it flat and wide.
- One of the solutions is to use the cumulative histogram (integral of intensity histogram) as the intensity mapping function.



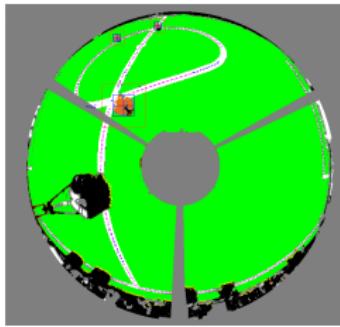
Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

Segmentation: concept

- Intermediate processing towards object recognition.
- Localize regions with common properties.
- Make a partition over the pixel ensemble.
- Usual grouping properties (Gray level, Color, Texture).
- Often requires preprocessing.
- Segmentation of non-trivial images is a difficult task.
- Segmentation accuracy determines the eventual success/failure of computerized image analysis.

Applications of segmentation



- The basis of many region based segmentation algorithms.
- The most immediate and computationally appealing step.
- Direct image partition based on intensity properties.
- Several approaches:
 - Global Thresholding
 - Variable Thresholding
 - Local - $T(x, y)$ depends on properties of the neighborhood of (x, y) .
 - Adaptive - $T(x, y)$ depends on the spatial coordinates, x and y .
 - The Otsu's method - Optimal global thresholding based on probabilistic estimates obtained from the histogram.

Region Growing

- Region growing is a procedure that groups pixels or subregions into larger regions based on a predefined criteria.
 - Start with a set of "seed" points and from these, grow regions by appending to each seed those neighboring pixels that have properties similar to the seed (intensity, color, ...).
- Selection of seeds
 - Often interactive
 - Automated
- Centroids of pixel clusters
- Additional criteria: size and shape of region grown so far
- Stopping rules
 - Ideally, growing a region should stop when no more pixels satisfy the criteria for inclusion in that region.

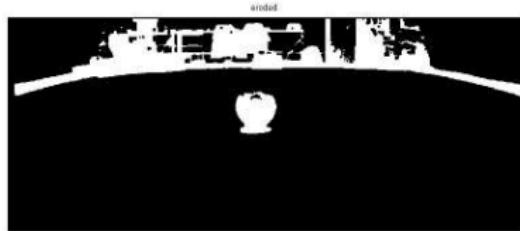
Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators**
- 12 Image Descriptors
- 13 Video Processing

Morphological operators



Green channel of the RGB image



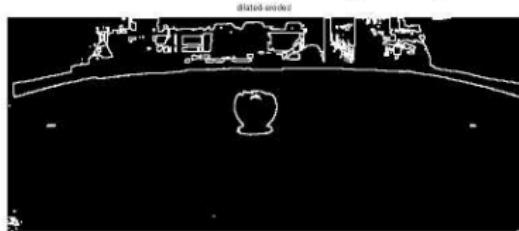
Binary image (Otsu method)



Dilation of the binary image



Erosion of the binary image



Difference between dilation and erosion

Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors**
- 13 Video Processing

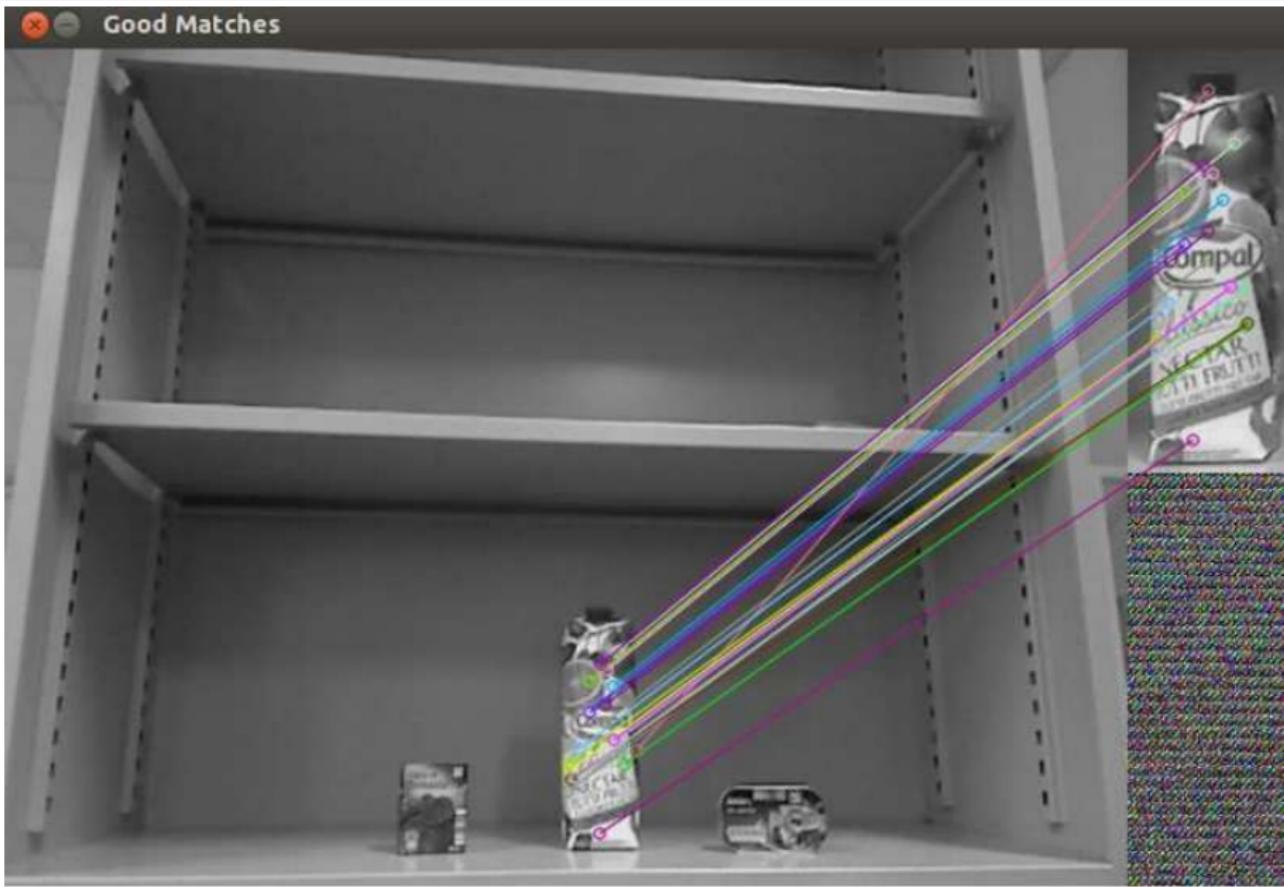
- **SIFT - Scale Invariant Features**

- SIFT features are generated by transforming an image into a collection of feature vectors, similar to neurons in inferior temporal cortex used in primate vision.
- SIFT keypoints of objects are extracted from a set of reference images and stored in a database.
- Recognition of an object in a new image is done by comparing each feature from this new image to the database.
- Best candidate is found based on the Euclidean distance of the feature vectors.

- **SURF - Speeded Up Robust Features**

- Inspired by SIFT . . . faster, more robust to image transformations.
- Relies on integral images for image convolutions and uses a Hessian matrix-based measure for the detector and a distribution-based descriptor, simplified to the essential.

Image Descriptors



Contents

- 1 Introduction
- 2 Human Vision
- 3 Image Formation
- 4 Digital Cameras
- 5 Digital Images
- 6 Color Spaces
- 7 Camera Calibration
- 8 Filtering
- 9 Histograms
- 10 Segmentation
- 11 Morphological Operators
- 12 Image Descriptors
- 13 Video Processing

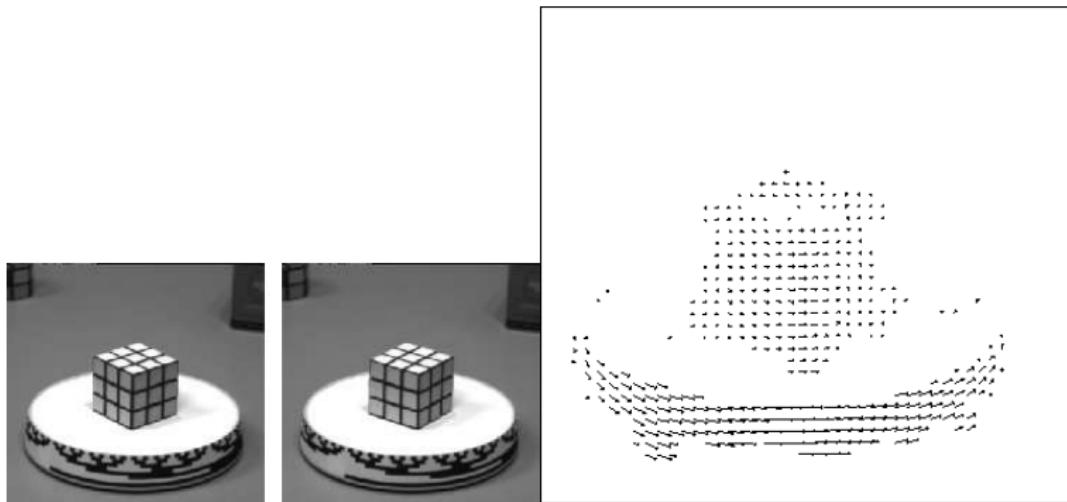
- A video signal can be represented by a 3D-function, $v(x, y, t)$, where x and y are spatial coordinates and t denotes time.
- The process of converting analog video into digital video requires both **spatial and temporal sampling**, besides **amplitude quantization**.
- Therefore, a **digital video** is a temporal sequence of digital images which we represent by $v(i, j, k)$, with $k = t/T, k \in \mathbb{N}_0$.
- $T \in \mathbb{R}$ indicates the period of time between two consecutive images (we call them frames). Therefore, $1/T$ (Hz) is the frame rate.
- Sometimes we will refer to video **fields**. They occur in interlaced video and are made of the even (odd) lines of a frame.

- Several information can be extracted from time varying sequences of images:
 - Camouflaged objects are only easily seen when they move
 - The relative sizes and position of objects are more easily determined when the objects move
 - Even simple image differencing provides an edge detector for the silhouettes of texture-free objects moving over any static background.

- The analysis of visual motion can be divided into two stages:
 - the measurement of the motion
 - the use of motion data to segment the scene into distinct objects and to extract three dimensional information about the shape and motion of the objects.
- There are two types of motion to consider:
 - movement in the scene with a static camera,
 - and movement of the camera, or ego motion.
- Since motion is relative, these types of motion should be the same. However, this is not always the case, since if the scene moves relative to the illumination, shadow and specularities effects need to be dealt with.

Motion field

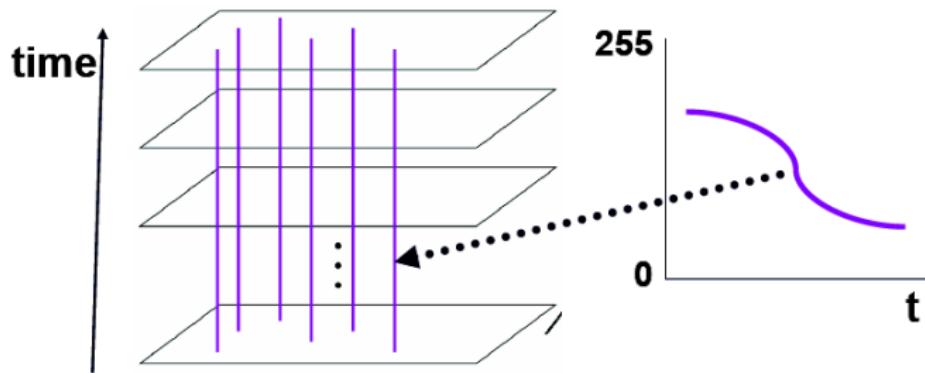
- The motion field is the projection of the 3D scene motion into the image.



- Definition: optical flow is the apparent motion of brightness patterns (or colors) in the image.
- Ideally, optical flow would be the same as the motion field.
- Have to be careful: apparent motion can be caused by lighting changes without any actual motion.
- To estimate pixel motion from image we have to solve the pixel correspondence problem.
- Given a pixel in frame t , look for nearby pixels with same characteristics (color, brightness, ...) in frame $t - 1$.

Background subtraction

- It is possible to look at video data as a spatio-temporal volume.
- If camera is stationary, each line through time corresponds to a single ray in space.



Background subtraction

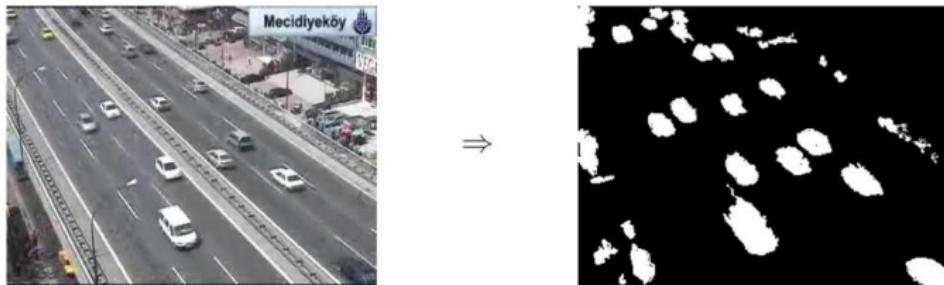
- Background subtraction is a commonly used class of techniques for segmenting out objects of interest in a scene for applications such as:
 - Surveillance
 - Robot vision
 - Object tracking
 - Traffic applications
 - Human motion capture
 - Augmented reality

Background subtraction

- It involves comparing an observed image with an estimate of the image if it contained no objects of interest.
- The areas of the image plane where there is a significant difference between the observed and estimated images indicate the location of the objects of interest.
- The name background subtraction comes from the simple technique of subtracting the observed image from the estimated image and thresholding the result to generate the objects of interest.

Important issues

- foreground detection – how the object areas are distinguished from the background;
- background maintenance – how the background is maintained over time;
- post-processing – how the segmented object areas are postprocessed to reject false positives.



Machine Learning in Robotics

Nuno Lau

Universidade de Aveiro/DETI
IEETA

Robótica Móvel e Inteligente
25/11/2022

Outline

- 1 Introduction
- 2 Supervised Learning
- 3 Unsupervised Learning
- 4 Evolutionary Learning
- 5 Reinforcement Learning

Outline

1 Introduction

2 Supervised Learning

3 Unsupervised Learning

4 Evolutionary Learning

5 Reinforcement Learning

Motivation

Programming robots is a hard work!

- No high-level programming language;
- Sensors and actuators are noisy;
- Robotics is moving towards increasingly unstructured environments.

If only robots could learn how to perform tasks by themselves...

Machine Learning

- Machine Learning is:
 - “A **computer program** is said to **learn** from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, **improves with experience E**” *Tom Mitchell. Machine Learning, 1997*
- Key concepts:
 - Experience (data);
 - Task;
 - Performance Measure (metric);
 - Improvement
- Machine Learning can be seen as a search problem of finding a policy that maps states to responses, ($\pi : S \rightarrow R$), to perform a desired task.

Machine Learning

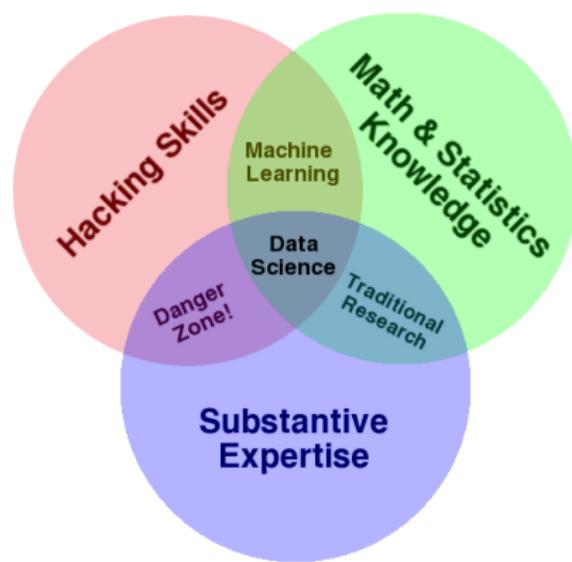


Figure: Data Science Venn Diagram, by Drew Conway

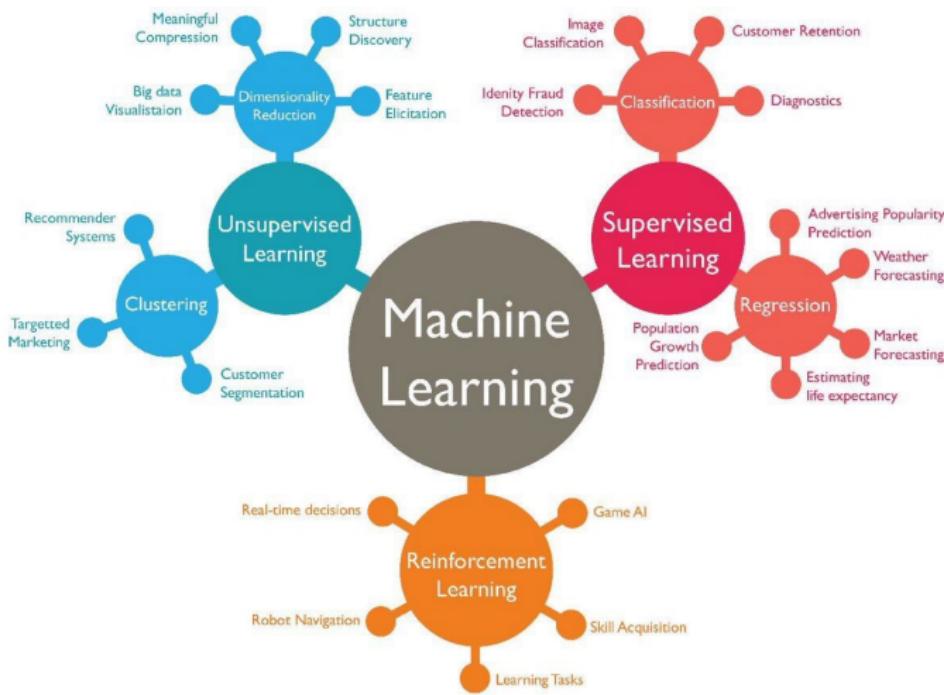
Challenges in Robot Learning

- Limited data;
- Generalization;
- Curse of Dimensionality;
- Which is the best action to take?
- Different Machine Learning paradigms
 - Supervised;
 - Unsupervised;
 - Semi-supervised Learning

Machine Learning Paradigms

- Supervised Learning
 - Learn from examples
- Unsupervised Learning
 - Discover structure in data
- Reinforcement learning
 - Learn from interaction

Machine Learning Paradigms



Outline

1 Introduction

2 Supervised Learning

3 Unsupervised Learning

4 Evolutionary Learning

5 Reinforcement Learning

Supervised Learning

Traditional Programming



Machine Learning



Supervised Learning

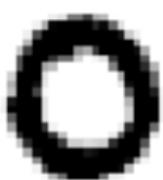
- A “teacher” **provides** training data consisting of **states** (S) and the **desired response** (R).
 - The learning process knows the desired output for several inputs.
 - The supervisor indicates what is the best action to take (sometimes).
- The robot must learn to **fit** the training data data and **generalize** a policy to the states not covered by the training data.
- Common methods:
 - k Nearest Neighbor, Logistic Regression, Neural Networks, Decision Trees, Support Vector Machines, Gaussian Processes,
...

Supervised Learning

Label: 1



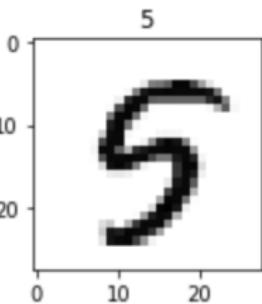
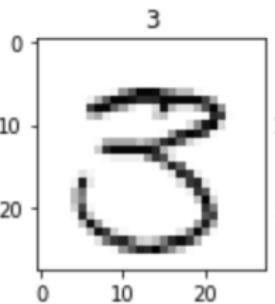
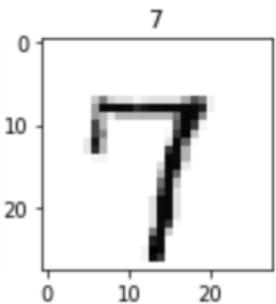
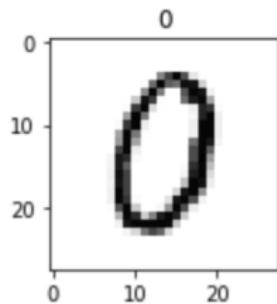
Label: 0



Label: 1



Label: 4



Supervised Learning

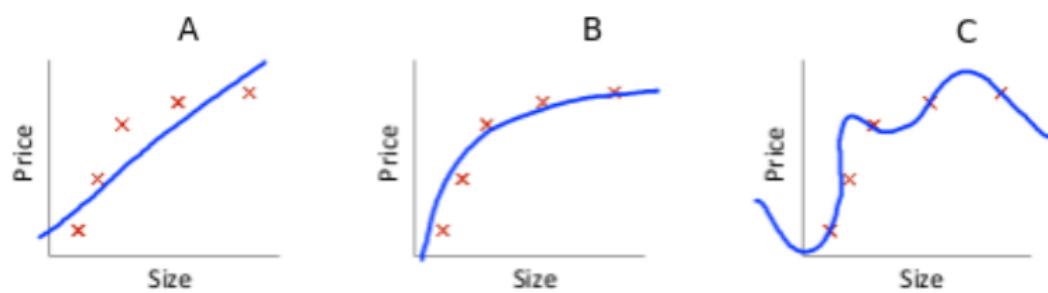


Figure: A regression problem: bias vs. variance

Supervised Learning

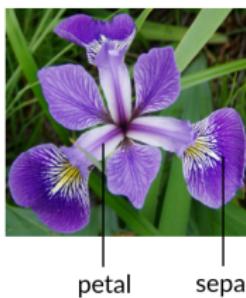
IRIS dataset

- Find flower species from:
 - Sepal width
 - Sepal length
 - Petal width
 - Petal length

iris setosa



iris versicolor



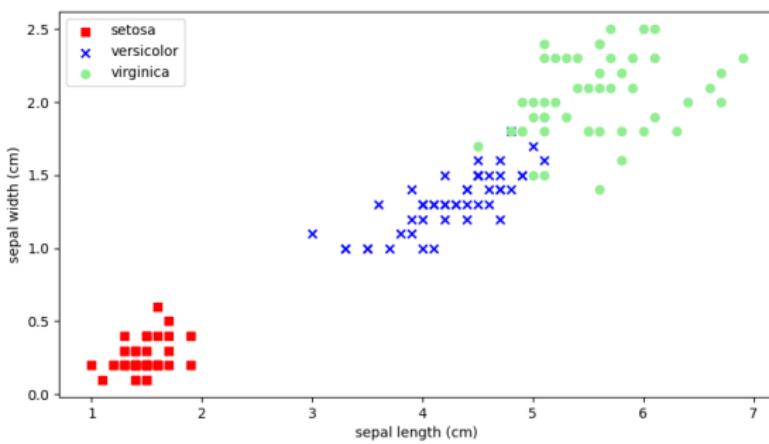
iris virginica



Supervised Learning

IRIS dataset

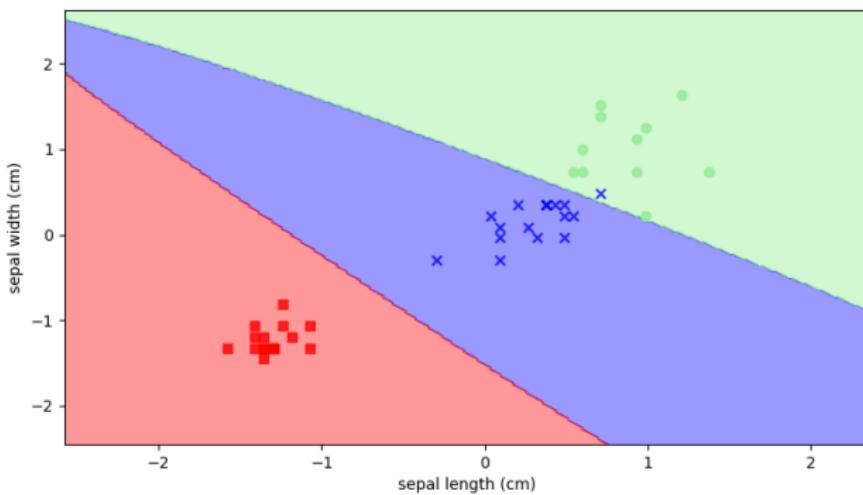
- Considering only:
 - Sepal width
 - Sepal length



Supervised Learning

IRIS dataset - SVM

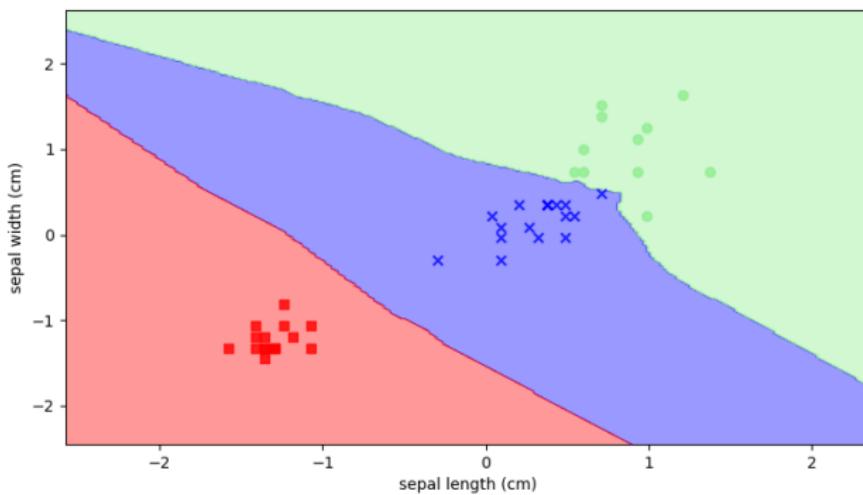
- Support Vector Machine Classification
 - Find the maximum margin hyperplane that separates categories



Supervised Learning

IRIS dataset - KNN

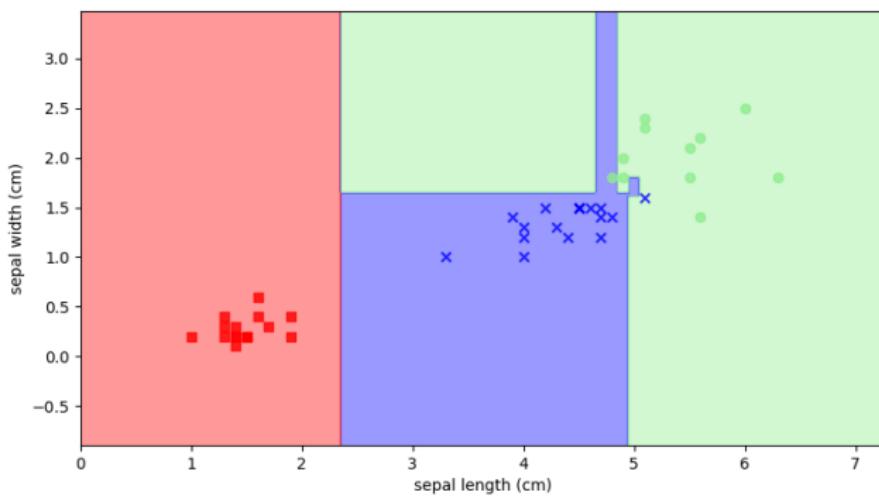
- K-Nearest Neighbor Classification
 - Find the majority among closest neighbors



Supervised Learning

IRIS dataset - Decision Tree

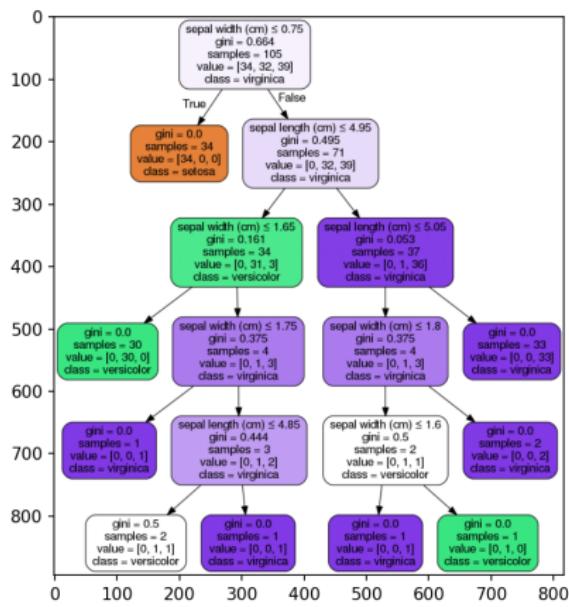
- Decision Tree Classification
 - Find best splitting rules based on feature values



Supervised Learning

IRIS dataset - Decision Tree

- Decision Tree Classification
 - Find best splitting rules based on feature values



Supervised Learning

Applications

- Very powerful when applied for Perception in Robotics!
 - Pattern Recognition for Robot Vision.
 - Probabilistic Models for Kalman and Particle Filters.
 - Fault detection.
 - Change detection.
- Learning from Demonstration for Control

Supervised Learning

Examples

- Autonomous Car Driving
 - A human drives a car and the driving behaviour is recorded.
 - [ALVINN: Autonomous Land Vehicle In a Neural Network](#) (Dean Pomerleau, 1988)
- Robotic Soccer
 - Four Legged League, Sony Aibo Platform.
 - Based on Accelerometer data, the robot was able to determine the surface it was moving on. (*Vail-2004*)
 - Also learned when it was moving freely or stuck or even entangled in other robots. (*Vail-2004*)
- Robotic Arm Control
 - Learn a probabilistic model for control over time.

Supervised Learning

Examples

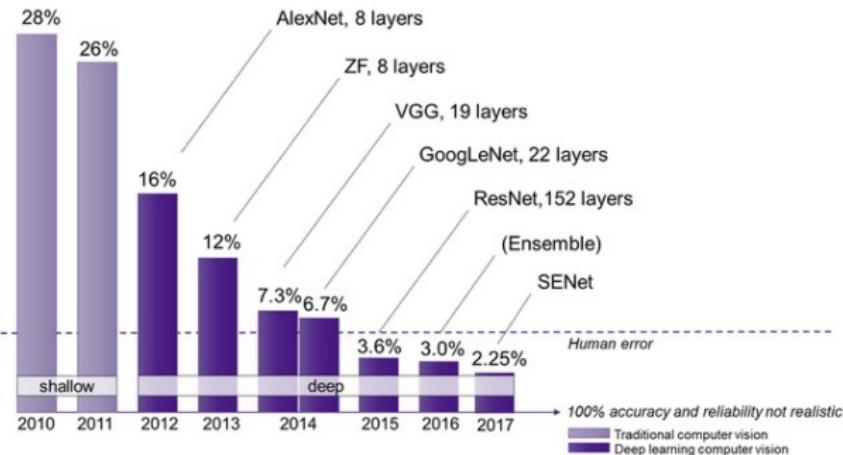


Figure: ImageNet Visual Recognition Challenge Results.

Outline

- 1 Introduction
- 2 Supervised Learning
- 3 Unsupervised Learning
- 4 Evolutionary Learning
- 5 Reinforcement Learning

Unsupervised Learning

- Learns directly from data, without any labeled dataset;
- Finds patterns/structure in data;
- Examples:
 - Clustering;
 - Anomaly detection;
 - Autoencoders, Self-organizing maps.

Clustering

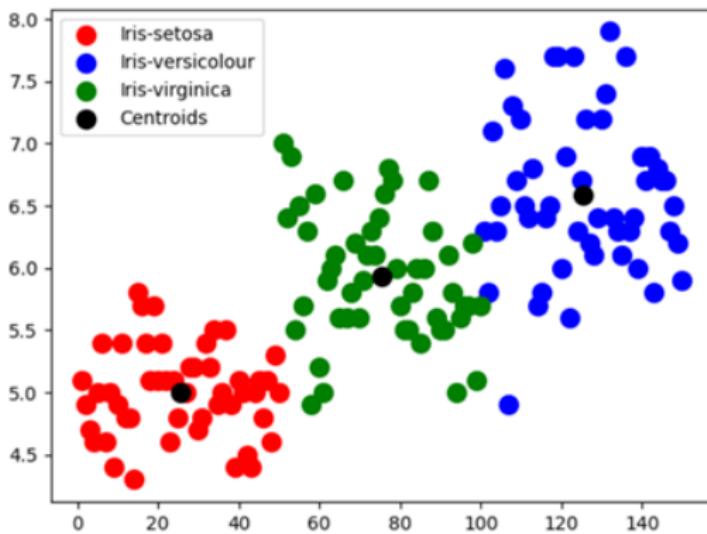


Figure: Clustering flowers.

Outline

- 1 Introduction
- 2 Supervised Learning
- 3 Unsupervised Learning
- 4 Evolutionary Learning
- 5 Reinforcement Learning

Evolutionary Learning

- Unsupervised/Semi-supervised Learning Paradigm.
- A **policy may be encoded** in **strings** (Genetic Algorithms) or in **computer programs** (Genetic Programming).
- The learning process works on a **set of policies** (generation).
- The robot is provided a **fitness function**.
- Each individual on the generation is evaluated given the fitness function.
- **Genetic operators:** selection, crossover, mutation.
- Generations are **regenerated** trying to get better fitness values

Evolutionary Learning

Examples

- Robot Motion
 - Applied in-house to learn biped walking gaits (Picado-2009).
 - Applied in-house to learn kick behavior (Abdolmaleki-2016).
 - Applied to learn the model of the robot (Lipson-2008).
- Robot Hardware
 - Applied to evolve the shape of the robot, to obtain previously unknown robot shapes (Lipson-2006).



Figure: One of Hod Lipson morphologically evolved robots.

Outline

- 1 Introduction
- 2 Supervised Learning
- 3 Unsupervised Learning
- 4 Evolutionary Learning
- 5 Reinforcement Learning

Reinforcement Learning

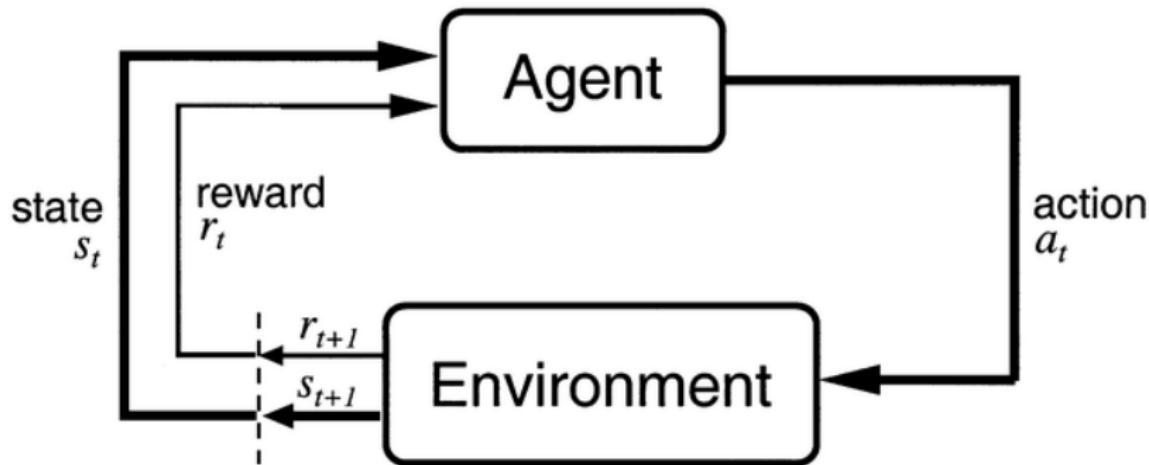


Figure: A Reinforcement Learning System.

Reinforcement Learning

- Unsupervised/Semi-supervised Learning Paradigm.
- Modeled as a Markov Decision Process:
 - ① a set of states S ;
 - ② a set of actions $A(s)$;
 - ③ a state transition model: $P(s, a, s') \rightarrow [0, 1]$
 - ④ a reward function : $R(s, a, s') \rightarrow r_t \in \mathbb{R}$
- The goal?
- To determine a policy that maximizes the return (total reward),
$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$
- How?
- By calculating a *Value Function*

Reinforcement Learning

- OpenAI gym (Markov Decision Process API)

```
import gym

# load the environment
env = gym.make('Example')

# perform N episodes
while True:

    # prepare the environment for the next episode
    obs = env.reset()

    # flag for episode completion
    done = False

    # run the episode
    while not done:

        # choose how to act based on the current state
        # usually the output of a neural network
        action = env.action_space.sample()

        # advance the simulation by one timestep
        # by interacting with the world
        obs, reward, done, info = env.step(action)

    # cleanup
    env.close()
```

Reinforcement Learning

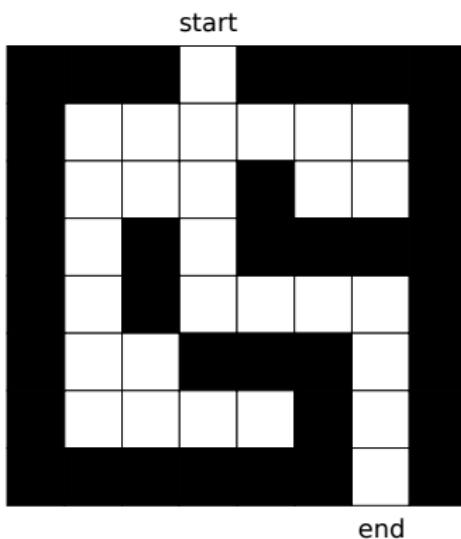


Figure: A maze environment.

Reinforcement Learning

- A Value Function estimates the expected return for *all* states
- $V^\pi(s_t) = \mathbb{E}[R_t]$

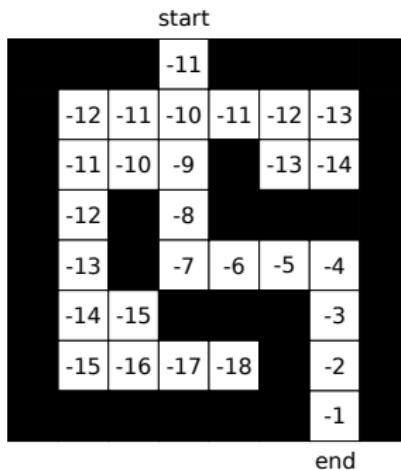


Figure: The optimal value function.

Reinforcement Learning

- A *Value Function* represents “how good” it is to be in a given state, $V^\pi(s_t) : s_t \rightarrow \mathbb{E}[R_t]$
- Bellman Equation:
$$V^\pi(s) = r(s, \pi(s), s') + \gamma V(s'), s' = f(s, \pi(s))$$
- How can we learn a Value Function?

Reinforcement Learning

- A *Value Function* represents “how good” it is to be in a given state, $V^\pi(s_t) : s_t \rightarrow \mathbb{E}[R_t]$
- Bellman Equation:
$$V(s) = \sum_{s'} P(s, \pi(s), s')[r(s, \pi(s), s') + \gamma V(s')]$$
- How can we learn a Value Function?

Reinforcement Learning

Algorithm 1 Policy Iteration (Policy Evaluation)

Require: $V(s)$ arbitrarily initialized, $\forall s \in S$

```
1: repeat
2:   repeat
3:      $\Delta \leftarrow 0$ 
4:     for all  $s \in S$  do
5:        $v \leftarrow V(s)$ 
6:        $V(s) \leftarrow \sum_{s'} P(s, \pi(s), s')[r(s, \pi(s), s') + \gamma V(s')]$ 
7:        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
8:     end for
9:   until  $\Delta \leq \theta$                                 ▷ a small positive value
10:  ...
11: until StablePolicy == True
```

Reinforcement Learning

Algorithm 2 Policy Iteration (Policy Improvement)

Require: $V(s)$ arbitrarily initialized, $\forall s \in S$

```
1: repeat
2:   ...
3:   StablePolicy  $\leftarrow$  True
4:   for all  $s \in S$  do
5:      $b \leftarrow \pi(s)$ 
6:      $\pi(s) \leftarrow \arg \max_a \sum_{s'} P(s, a, s')[r(s, a, s') + \gamma V(s')]$ 
7:     if  $b \neq \pi(s)$  then
8:       StablePolicy  $\leftarrow$  False
9:     end if
10:   end for
11: until StablePolicy == True
```

Reinforcement Learning

Algorithm 3 Value Iteration

Require: $V(s)$ arbitrarily initialized, $\forall s \in S$

```
1: repeat
2:    $\Delta \leftarrow 0$ 
3:   for all  $s \in S$  do
4:      $v \leftarrow V(s)$ 
5:      $V(s) \leftarrow \max_a \sum_{s'} P(s, a, s') [r(s, a, s') + \gamma V(s')]$ 
6:      $\Delta \leftarrow \max(\Delta, \|v - V(s)\|)$ 
7:   end for
8: until  $\Delta \leq \theta$                                 ▷ a small positive value
```

Reinforcement Learning

- That's nice, but what do we do with a *Value Function*?
- Extract a *policy*!
- A *policy* maps states to actions, $\pi : s \rightarrow a$
- $\pi(s) = \arg \max_a \sum_{s'} P(s, a, s')[r(s, a, s') + \gamma V(s')]$

Reinforcement Learning

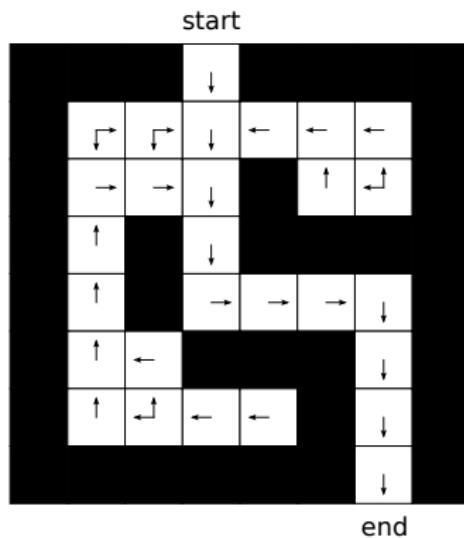


Figure: The optimal policy

Reinforcement Learning

- ➊ Finite State Space
- ➋ Finite Action Space
- ➌ Curse of Dimensionality

Reinforcement Learning

- ➊ Finite State Space
- ➋ Finite Action Space
- ➌ Curse of Dimensionality

Question?

What if we don't know the state transition model?

Example

- Where is the opponent dribbling the ball?
- To where in the goal is the opponent shooting the ball?

Reinforcement Learning

- ① Finite State Space
- ② Finite Action Space
- ③ Curse of Dimensionality

Question?

What if we don't know the state transition model?

Example

- Where is the opponent dribbling the ball?
- To where in the goal is the opponent shooting the ball?

Reinforcement Learning

- We use an *action Value Function* $Q^\pi(s_t, a) = \mathbb{E}[R_t | a_t = a]$
- We let the robot interact with the world and observe the collected rewards
- From that we build a *Value Function*

Reinforcement Learning

Algorithm 4 The Q-Learning algorithm

Require: $Q(s, a)$ initialized arbitrarily $\forall s \in S, \forall a \in A(s)$

- 1: **loop**
 - 2: Initialize $s = s_0$
 - 3: **repeat**
 - 4: $a \leftarrow \pi(s)$
 - 5: take action a ; observe reward, r , and successor state s'
 - 6: $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_b Q(s', b) - Q(s, a)]$
 - 7: $s \leftarrow s'$
 - 8: **until** s is terminal
 - 9: **end loop**
-

Reinforcement Learning

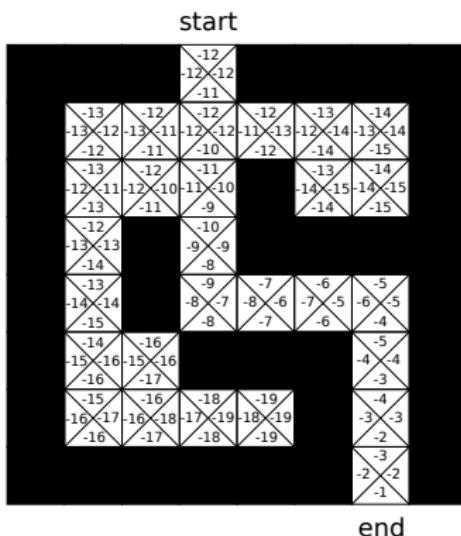


Figure: The optimal Q-function.

- $\pi(s) = \arg \max_a Q(s, a)$

Reinforcement Learning

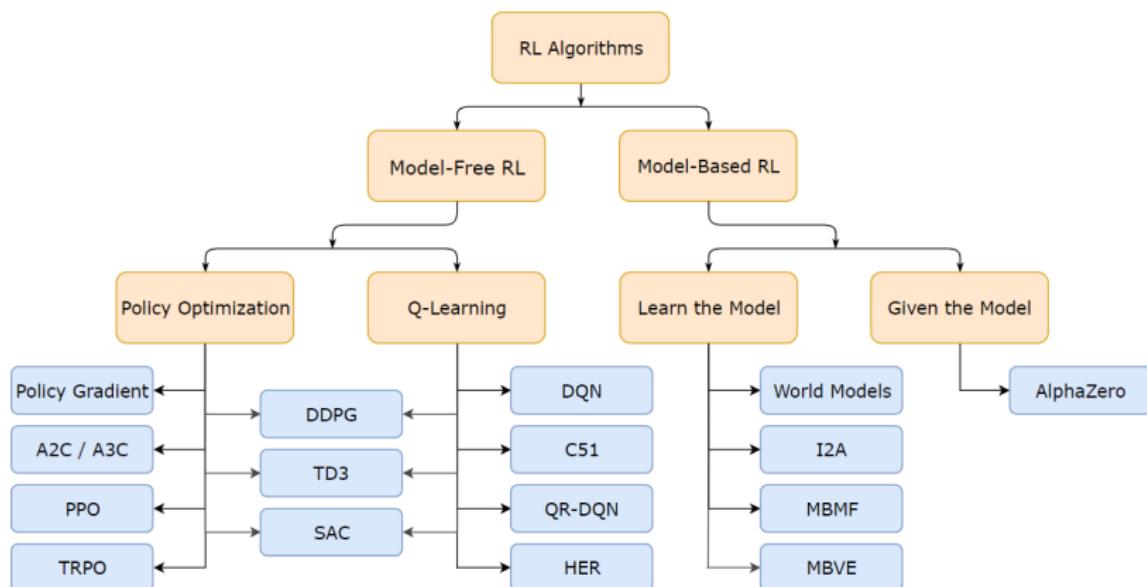


Figure: Reinforcement Learning Taxonomy.

Summary

- Machine Learning is a valid software development tool for programming robotic agents.
- Many paradigms, many challenges, many solutions, even more problems...
- The future of Robotics?

References

- Pattern Recognition and Machine Learning, Christopher Bishop, Springer, 2006
- Machine Learning, Tom Mitchell, McGraw Hill, 1997
- Reinforcement Learning: An Introduction, Richard Sutton, Andrew Barto, MIT Press, 2018
- OpenAI, scikit-learn, TensorFlow, PyTorch, stable baselines, Keras, RapidMiner, clsquare, OpenCV, Shark, ...