BASICS of MOBILE ROBOTICS 0 Introduction

Prof. Francesco Mondada
Laila El-Hamamsy

EPFL

2021-2022

Table of Contents

What is a Mobile Robot?

Some definitions:

- A machine that senses, thinks, and acts. (G.A. Bekey, 2005)
- Oxford English Dictionary : “A machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer.”

No general consensus on a clear definition for “mobile robot”

- What does “think” mean?
- What does “complex” mean?

Robots that are not “mobile” will not be tackled in this course, e.g.:

- Industrial robots
- Torsos
- Prostheses

What Kind of Mobile Robots?

Wheeled Robots

Thymio (EPFL)

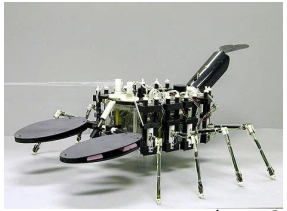
Roomba (iRobot)

Uranus (CMU)

Walking and Running Robots

BigDog (Boston Dynamics)

Asimo (Honda, Japan)

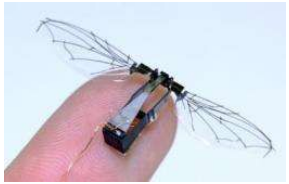


Lobster robot (U of Northeastern USA)

Flying Robots

Hummingbird (AeroVironment)

Dragon fly (WowWee HK)



Micro aerial vehicle, Harvard

Swimming and Crawling Robots

G6 Fish Robot, University of Essex

Penguin Robot (Festo, Germany)

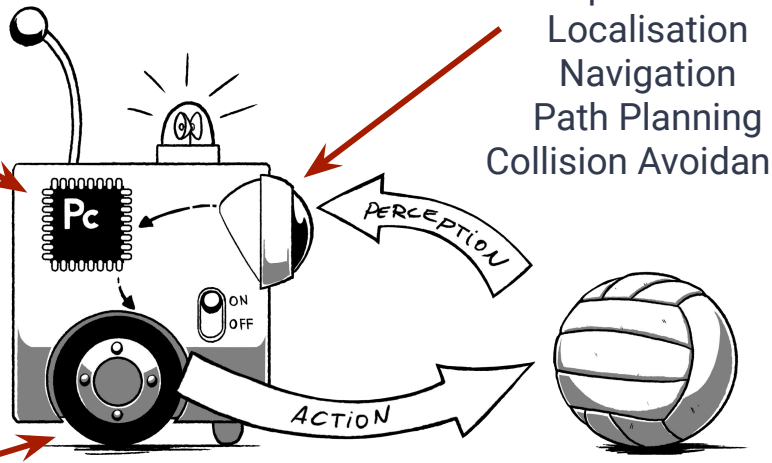
Snake Robot (CMU, USA)

Aspects of Mobile Robotics

System Integration

Processors
Embedded electronics
Architectures
Communication
Programming
Real-time Control

Sensors Integration
Computer Vision
Localisation
Navigation
Path Planning
Collision Avoidance



Mechanics
Actuators
Energy
Locomotion

Table of Contents

Course Objectives

1. Get an overview of the mainstream engineering techniques involved in mobile robot development.
2. Get a deeper understanding of a subset of techniques, presented more in detail in the course, addressed in the exercises and revised in the case studies.
3. Acquire hands-on experience in mobile robotics by means of practical laboratories on a real robot.

Course Topics (with project)

Week 1	Components of a mobile robot	**Week 8**	Uncertainties
Week 2	Vision	**Week 9**	Localisation 2 + Project week 1
Week 3	Vision & ANN & ML	**Week 10**	Project week 2
Week 4	Navigation	**Week 11**	Project week 3
Week 5	Navigation	**Week 12**	Project week 4 + Project presentations
Week 6	Localisation 1	**Week 13**	Project presentations
Week 7	Uncertainties	**Week 14**	Pr. presentations + Conclusion + Dry Exam

Weekly Course Organisation

- 15:15-16:00 **Case studies** on the topics seen and trained the week before
- 16:15-17:00 **Lecture**
  - One global topic
  - An overview on several techniques related to the topic
  - Some techniques in more detail
  - When needed, one numerical example
- 17:00-19:00 **Exercises** with the Thymio robot
  Assignments on Moodle
  https://moodle.epfl.ch/course/view.php?id=15293

The course is continously restructured and adapted to the COVID situation, so please be patient 🙂 and feedback is always welcome! (see moodle)

Course Organisation

Slides on Moodle: https://moodle.epfl.ch/course/view.php?id=15293

Enrollment: self-enrolment.

Feedback: every week on moodle, anonymous

Case studies: slides presenting the cases on moodle, interactive discussion

Exercices: every week on moodle, solution published one week later

Thymio robot: one for each student for the whole semester

Course Organisation

Thymio robot : last year we asked a 50CHF retainer and we lost money and robots. We will not do that anymore.

We apply a simple rule:

- normally the project grade is published ASAP after the project defenses.
- This year we will not publish the grades until ALL Thymio are back, but we will publish the names of students not returning Thymio.

Distribution of Thymios during the exercises of week 1 and 2.

Course Material

References :

- **Mobile Robots Course** - EPFL - J.-C. Zufferey, Felix Schill, 2013
- **Introduction to Autonomous Mobile Robots** R. Siegwart, I. Nourbakhsh, and D. Scaramuzza, MIT Press, 2011.
- **Elements of Robotics,** M. Ben-Ari, F. Mondada, Springer, 2018. (free download!)
- **Autonomous Robots: From Biological Inspiration to Implementation and Control** G.A. Bekey, MIT Press, 2005.
- **Probabilistic Robotics** S. Thrun, W. Burgard and D. Fox, MIT Press, 2005.
- **Springer Handbook of Robotics** B. Siciliano, and O. Khatib (Eds.), 2nd edition, Springer, 2016.

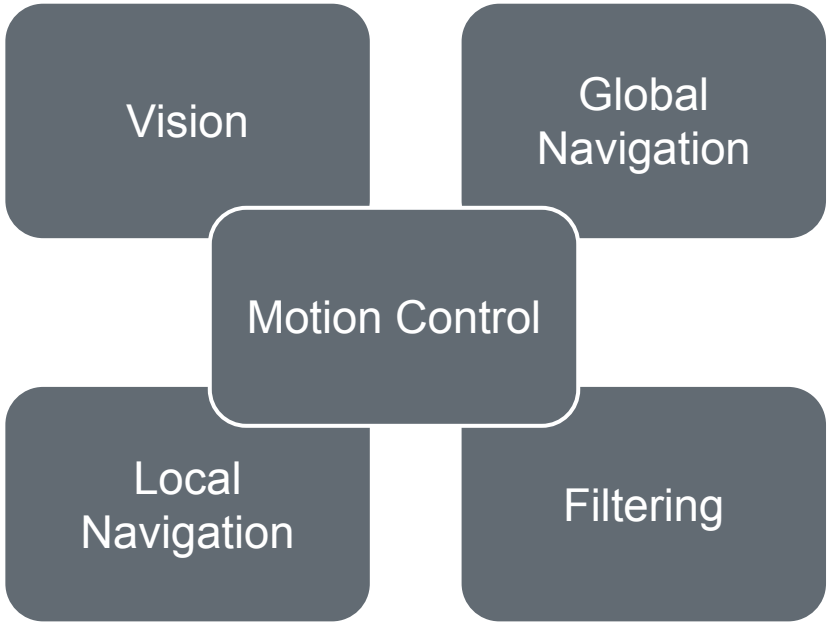
Evaluation (with project)

In 2 parts

- 60% Project on one of the topics seen during the semester + the exercise sessions
- 40% Written exam during the winter examination session
  - Related to the case studies (multiple choice answer + explanation)
  - No documentation allowed
  - No electronics allowed (no calculator, phone, smart watch, …)

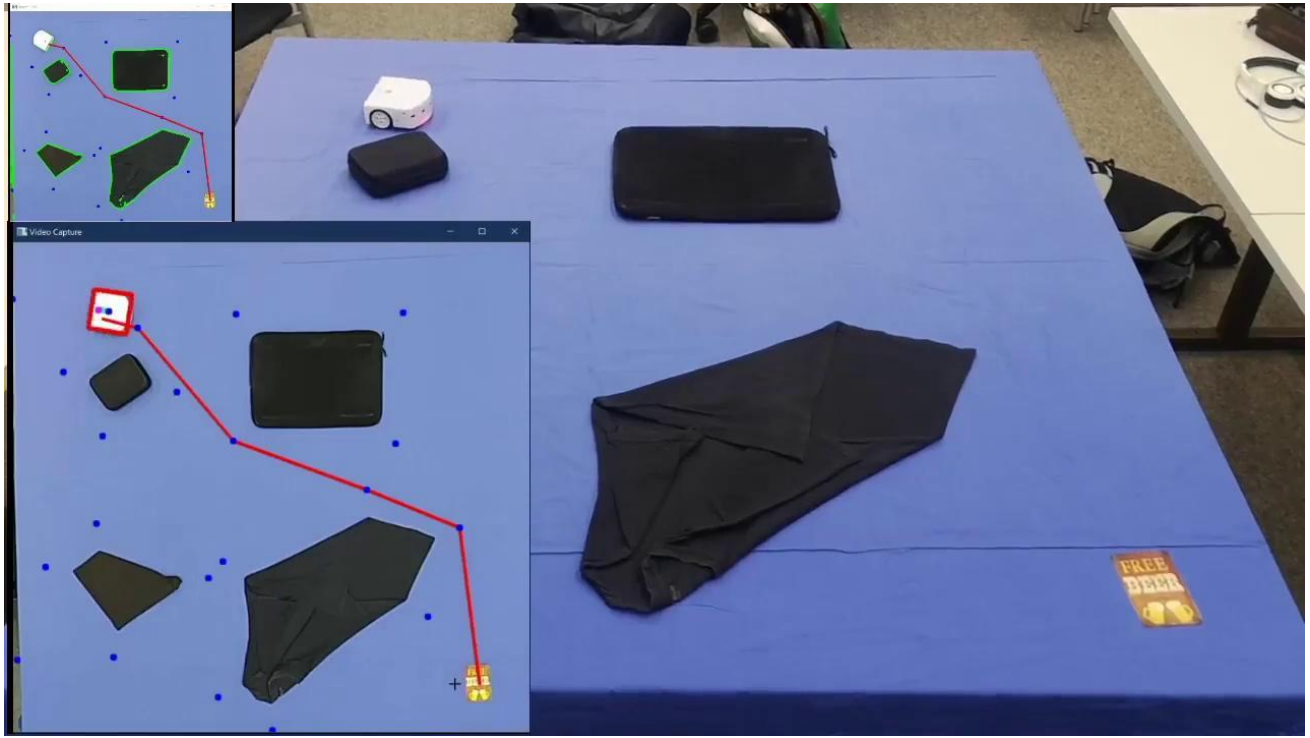
Project Information

- Groups of 4 students
- Presentations weeks 12 & 13 & 14
- 4 weeks without exercise sessions (weeks 9-12) to work on it, note that there will be a full lecture week 9 and a case study week 10.
- TAs available from 17:15 - 19:00 on Tuesdays
- Please use the forum, to allow everybody to benefit from the response.



Components that are required for the project

Examples from Autumn 2019-2020



https://www.youtube.com/watch?v=UDhiHlllJEQ

Table of Contents

Core Robotics Labs at EPFL

STI-LIS (Prof. D. Floreano): Flying robots, swarm robotics, bio-inspired A.I.

STI-LASA (Prof. A. Billard): Machine learning, imitation, humanoids

STI-BioRob (Prof. A. Ijspeert): Bio-inspired locomotion, biomedical robotics, industrial robotics

STI-Mobots (Prof. F. Mondada): Robot design, miniature mobile robots, educational robotics

STI-RRL (Prof. J. Paik): Robot design, foldable robots STI-MICROBS (Prof. S. Sakar): Microrobotics

STI-MICROBS (Prof. M. Sakar): MicroBioRobotic Systems Laboratory

STI-CREATE (Prof. Josie Hughes): fabrication and computational design tools

ENAC-DISAL (Prof. A. Martinoli): Collective systems

ENAC-VITA (Prof. A. Alahi): Visual Intelligence for transportation

BASICS of MOBILE ROBOTICS 1
Components of a Mobile Robot

Prof. Francesco Mondada
Laila El-Hamamsy

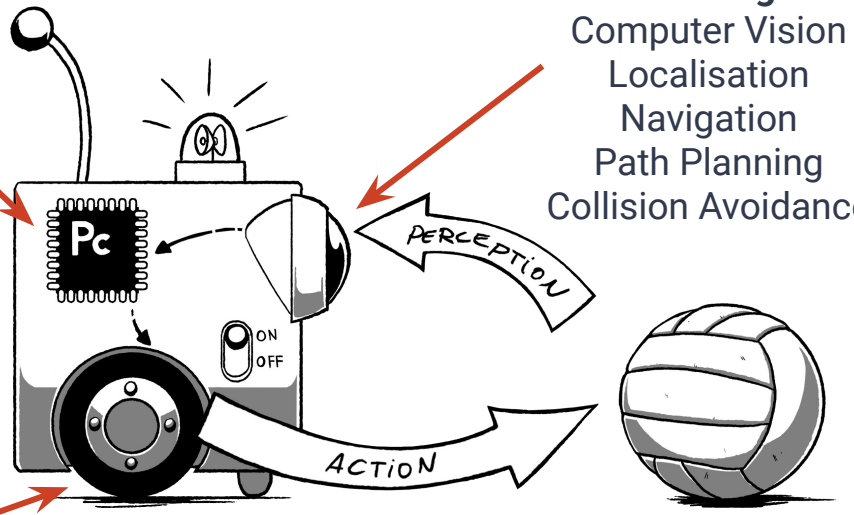
EPFL

Overview of the Components of a Mobile Robot

System Integration

Processors
Embedded electronics
Architectures
Communication
Programming
Real-time Control

Sensors Integration
Computer Vision
Localisation
Navigation
Path Planning
Collision Avoidance



Mechanics
Actuators
Energy
Locomotion

credits : Thymio MOOC

Table of Contents

Mechanics – Locomotion Control

Is a difficult and unsolved problem :

Locomotion and movement are due to **complex interactions between the controller, the body, and the environment.** Requires solving multiple complex computational challenges:

- good coordination of multiple DOFs,
- dealing with uncertainties,
- keeping balance,
- adapting to terrain/environment,
- adapting to changing body properties, …

Challenge for mechanics, actuation, energy, sensing, control, ... This is still not properly solved in robotics, and still not properly understood in animals

Mechanics – Trade Offs

	What	Characteristics	Implications
Stability	Not falling over.	Static stability does not require any motion to maintain balance, as opposed to dynamic stability.	Static stability is guaranteed with 3 wheels. More wheels increases dynamic stability, but these should be free to avoid impacting maneuverability as these systems are hyperstatic and require a flexible suspension system.
Controllability	Ease of converting motor commands into rotational and translational velocities.	Useful for accurate steering and dead reckoning (i.e. estimation of one's position based on previous positions).	Inverse correlation between controllability and maneuverability. • cars have good controllability but poor maneuverability • omnidirectional robots using Swedish wheels have good maneuverability but poor controllability (due to uncertainty in steering and speed).
Maneuverability	Ability to change direction.	Omnidirectional drive offers the highest maneuverability (ability to move at any time in any direction on the ground plane).	

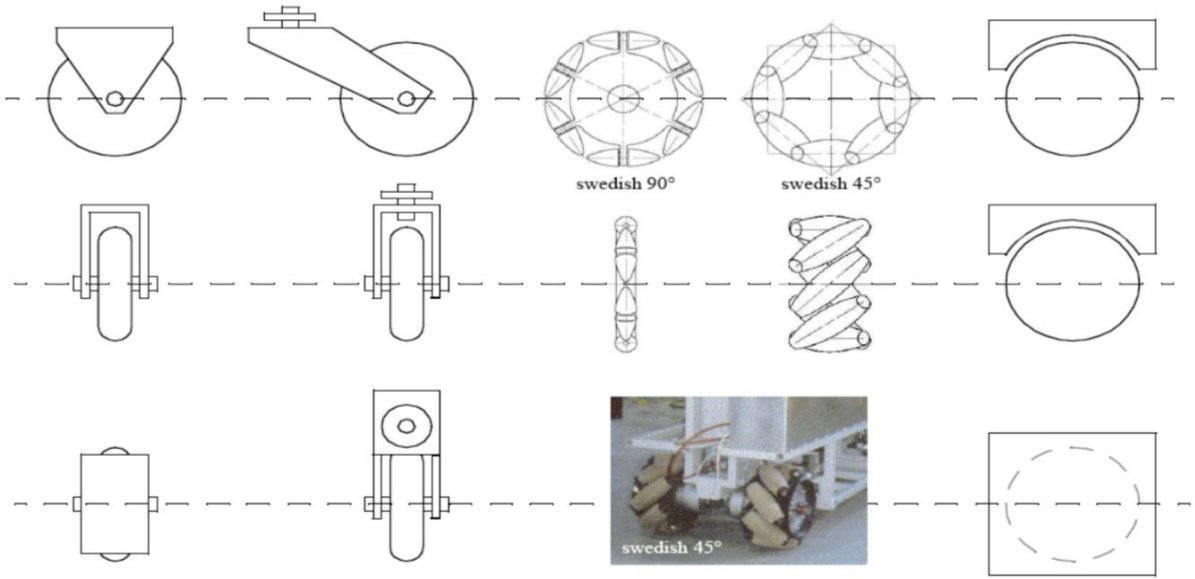
Mechanics – 4 Basic Wheel Types

a) **Standard wheel**: rotation around the (motorized) wheel axle and the contact point.

b) **Castor wheel**: rotation around the wheel axle, the contact point and the castor axle.

c) **Swedish (or mecanum) wheel**: rotation around the (motorized) wheel axle, around the rollers and around the contact point.

d) **Ball or spherical wheel.**



Different arrangements with 2, 3, 4 and 6 wheels exist

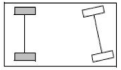
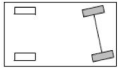
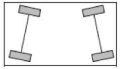
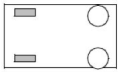
Mechanics – Arrangements of wheels

Different arrangements with 2, 3, 4 and 6 wheels exist

number of wheels	Arrangement	Description	Typical examples
2		One steering wheel in the front, one traction wheel in the rear	bicycle, motorcycle
		Two-wheel differential drive with the CG below the axle	Cye personal robot

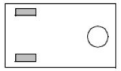
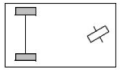
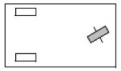
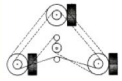
actuated wheel

free wheel

4		2 motorized wheels in the rear, 2 steered wheels in the front; Steering has to be different for the two wheels to avoid slipping/ skidding.	car with rear wheel drive
		2 motorized and steered wheels in the front, 2 free wheels in the rear; Steering has to be different for the two wheels to avoid slip-ping/skidding.	car with front wheel drive
		4 steered and motorized wheels	four wheel drive, four wheel steering
		Two traction wheels (differen-tial) in rear/front, two omnidirec-tional wheels in the front/rear	Charlie (DMT-EPFL)
		Four omnidirectional wheel	CMU Uranus
		Two wheel differential drive with two additional points of contact	EPFL Khepera, Hyperbot Chip

"Ackermann"

differential drive

3		Two-wheel centered differential drive with a third point of contact	Nomad Scout, smartRobII EPFL
		Two independently driven wheels in the rear/front, one unpowered omnidirectional wheel in the front/rear	many indoor robots, including the EPFL robots Pygmalion and Alice
		Two connected traction wheels (differential) in rear, one steered free wheel in front	Piaggio mini-trucks
		Two free wheels in rear, one steered traction wheel in front	Neptune (Carnegie-Mel-lon University
		3 motorized swedish or spheric wheels arranged in a triangle. Ommidirectional movement is possible.	Stanford-wheel Tribolo EPFL
		3 synchronously motorized and steered wheels. The orientation is not controllable.	'synchro drive' Denning MRV-2, Georgia Institute of Technology, I-Robot B24, Nomad 200

differential drive

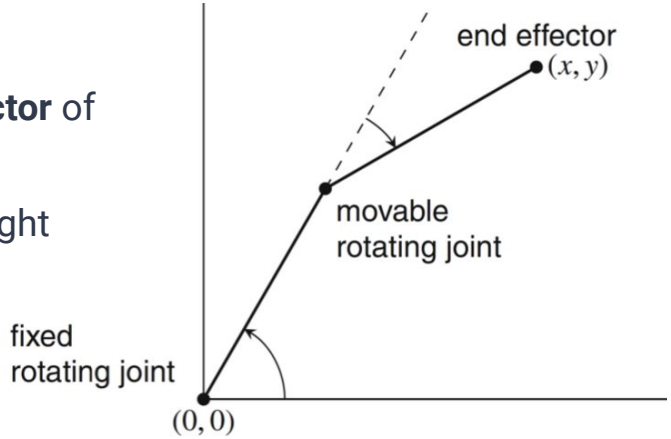
omnidirectional drive

synchro drive

Mechanics – Degrees of Freedom (DOF)

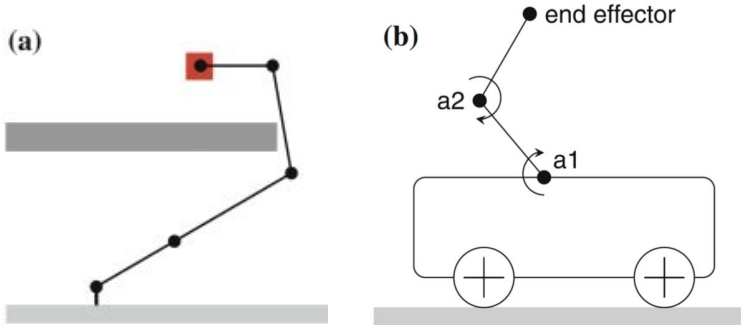
Number of DOF = dimensionality of the coordinates needed to describe the **pose** of a mobile robot or the pose of **the end effector** of a robotic manipulator

- e.g. Thymio has 3 DOF, a train has 1 DOF, the arm on the right has 2



DOF and Actuators

- **n DOF = n Actuators** : e.g. a train, the two-line robotic arm shown
- **n Actuators < n DOF** : e.g. a differential drive robot which has 2 actuators but can reach all 3 dimensional poses in the plane -> cheaper but harder to plan and control
- **n Actuators > n DOF** : redundant systems which can be useful in practice for complex movements or combining several characteristics



Mechanics – Degrees of Mobility (DOM)

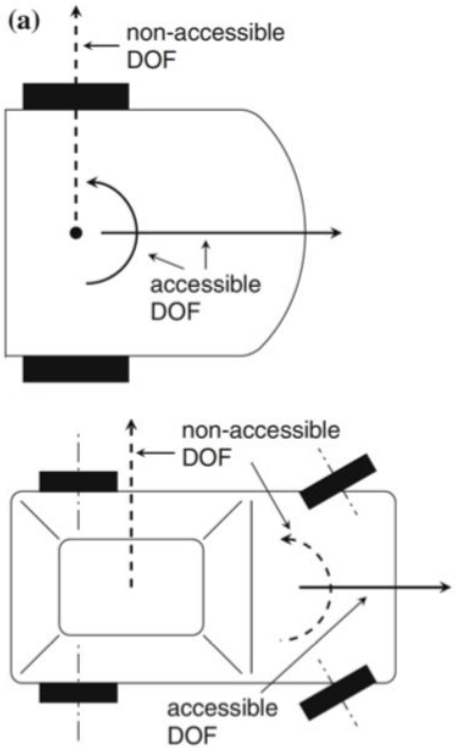
The controllable DOF or degree of mobility (DOM) δm = number of degrees of freedom that can be directly accessed by the actuators.

- A differential drive robot has 3 DOF: position (x,y) and orientation (θ). But only 2 DOF are controllable: forward speed and rotation.
- A car has also 3 DOF but only 1 DOF is directly controllable: forward speed. The other actuator, the steering wheel, does not give direct access to any additional DOF, it can only orient the first DOF.

  -> A differential mobile robot has better mobility than a car. A car cannot turn on the spot.

Holonomicity implies that the controllable DOF or DOM is equal to the total degrees of freedom (in the task space). If DOM < DOF then the robot is non holonomic (can move in some directions but not others)

Caution: (in general) omnidirectional != holonomic



Mechanics –Differential Drive Motion Control

Motion Controller to follow a trajectory or to reach a target pose. Not straightforward for non holonomic or nonlinear systems.

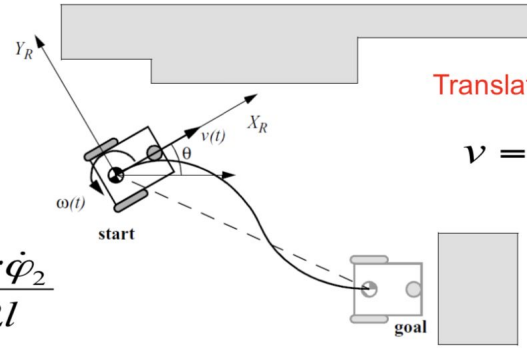
Examples of controller (not perfect, does not consider all forces and velocities in the state vector)

There are several approaches to bring the robot at desired position and orientation

- option 1 : combination of rotations on the spot + straight segments -> simple but not smooth
- option 2 (Astolfi): control law that smoothly modulates forward and rotational velocities

Mechanics –Differential Drive Motion Control

Astolfi Controller - 1. Extracting Rotational and Translational Velocity From Motor Speeds. Provided the relation between linear and angular velocities, the following can be deduced from the robot's geometry



Translational velocity

$$v = \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2}$$

Rotational velocity

$$\omega = \frac{r\dot{\varphi}_1}{2l} + \frac{-r\dot{\varphi}_2}{2l}$$

Where $\dot{\varphi}_i$ is the motor speed, r is the wheel radius, l is the axle length, v is translational velocity and ω is the rotational velocity.

Mechanics –Differential Drive Motion Control

Astolfi Controller - 2. Kinematic Model : Mapping velocities from the robot's reference to the global reference frame. Requires using the inverse rotation matrix.

Translational velocity

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = R(\theta)^{-1}\begin{bmatrix}\dot{x}_R\\ \dot{y}_R\\ \dot{\theta}_R\end{bmatrix} = R(\theta)^{-1}\begin{bmatrix}v\\ 0\\ \omega\end{bmatrix} = \begin{bmatrix}cos\theta & sin\theta & 0\\ -sin\theta & cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{\dot{\varphi}_1 r + \dot{\varphi}_2 r}{2}\\ 0\\ \frac{\dot{\varphi}_1 r - \dot{\varphi}_2 r}{2l}\end{bmatrix}$$

Global Frame

Local Robot Frame

Rotational velocity

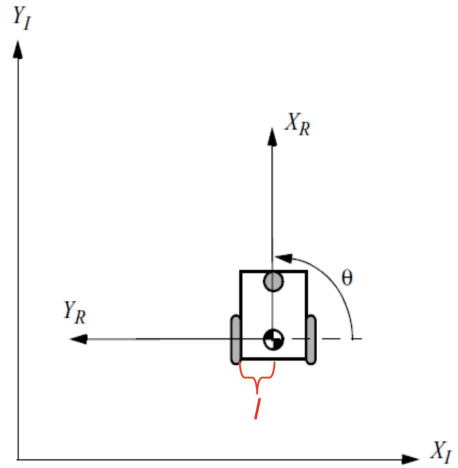


Figure 3.2
The mobile robot aligned with a global axis.

Mechanics –Differential Drive Motion Control

Astolfi Controller - 3. Transform to polar coordinates with origin at goal position to simplify the control law

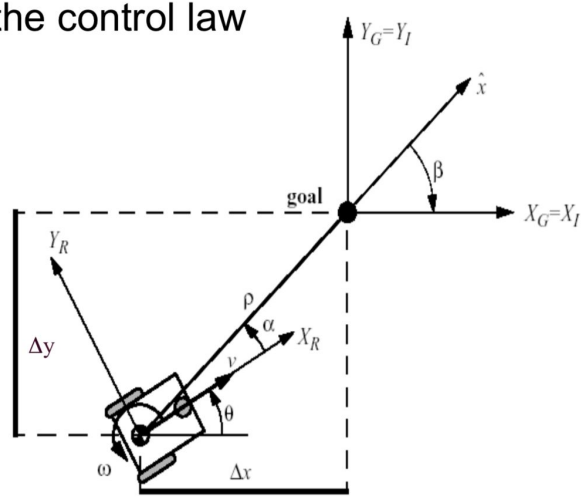
The following transformation into **polar coordinates** with origin at the goal position allows to simplify the control law

$$\begin{cases} \rho = \sqrt{\Delta x^2 + \Delta y^2} \\ \alpha = -\theta + \operatorname{atan2}(\Delta y, \Delta x) \\ \beta = -\theta - \alpha \end{cases}$$

note: $\operatorname{atan2}(a,b) = \arctan\left(a/b\right)$

where the signs of both arguments are used to determine the quadrant of the result.

Target: $(\rho,\alpha,\beta) = (0, 0, 0)$

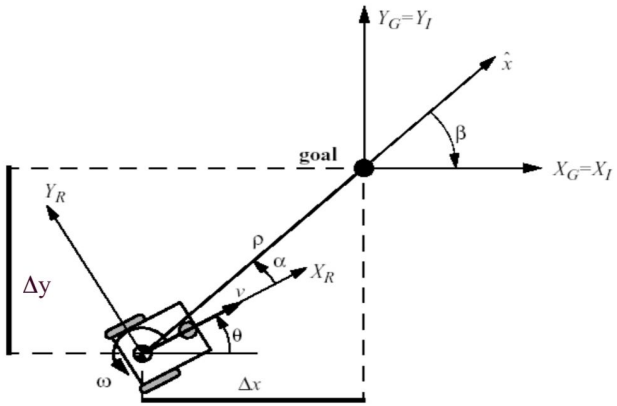


Mechanics –Differential Drive Motion Control

Astolfi Controller - 4.

The kinematic model in the new polar coordinates:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos\alpha & 0 \\ \frac{\sin\alpha}{\rho} & -1 \\ -\frac{\sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$



Mechanics –Differential Drive Motion Control

Astolfi Controller - 4.

It can be shown, that the linear control law:

$$v = k_\rho \rho \qquad \omega = k_\alpha \alpha + k_\beta \beta$$

yields the closed loop system:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho \rho \cos\alpha \\ k_\rho \sin\alpha - k_\alpha \alpha - k_\beta \beta \\ -k_\rho \sin\alpha \end{bmatrix}$$

which has a **unique equilibrium point** at $(\rho, \alpha, \beta) = (0, 0, 0)$

Using the Lyapunov theory, it can be shown (Astolfi, 1995, 1997) that the closed loop control system is **exponentially stable** if:

$$k_\rho > 0 \; ; \; k_\beta < 0 \; ; \; k_\alpha - k_\rho > 0$$

The control signal v has always constant sign:

- ⇒ the direction of movement is kept positive or negative during the entire movement.
- ⇒ parking maneuver is performed always in the most natural way and without ever inverting its motion.

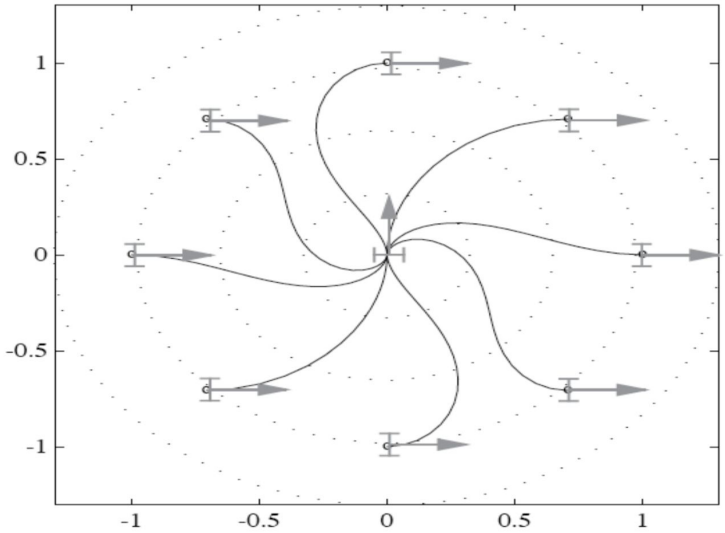
Mechanics –Differential Drive Motion Control

Astolfi Controller - 4.

RESULTING PATHS

Exponential stabilization of a wheeled mobile robot via discontinuous control

A Astolfi - Journal of dynamic systems, measurement, and control, 121 (1), pp 121-126,1999



Mechanics – Actuators

Actuators	Advantages	Inconveniences	Usage Examples
Combustion Engines	• High energy density (60x more than a Lithium-Polymer battery)	• Poor efficiency (30%) • Heavier for the same power • Pollutant emissions (not for indoors)	• robotized cars, trucks, etc. • flying robots • Sometimes used in outdoor legged robots (BigDog, Boston Dynamics)
Hydraulic Actuators -	High torque-to-weight ratio, higher than electric motors. Useful for large loads	• Maintenance problems => not often used in mobile robotics.	Exceptions: Legged robots from BostonDynamics + some humanoid robots (CB from Sarcos, Atlas from Boston Dynamics)
Pneumatic Actuators	lightweight, easy attachment, toughness, high efficiency and high power over short distances	• Limited control • Highly nonlinear • Need for large compressor	e.g. air muscles
Electric Motors	• Large power range (mW-MW) • Easy to control • Excellent efficiency (~90%) • No pollutant emissions • Can integrate gears & encoder		

Mechanics – Actuators : Electrical Motors

Actuator		Advantages	Inconveniences	Examples
DC Motors	Brushed (commutation at the level of the rotor)	• Brushed easier to control (mechanical commutation)	• Friction -> power losses, wear down • Speed & torque trade-off	
	Brushless (commutation at the level of the stator)	• Higher torque (w.r.t brushed) • No friction -> longer life	• Trade-off between speed and torque • Complex electronics for the controls	
Servomotors (integrated position control, generally DC brushed motor)		• Integrated regulation	• Limited range • Generally limited torque	
Stepper Motors (converts electrical pulses into discrete mechanical movements)		• Open-loop drive by counting steps, accuracy of ±1 step. • High holding torque when the rotor is stationary.	• Cannot tell when steps have not been counted and cannot be recovered • Complex electronic control	e-puck

Mechanics – Actuators : Electrical Motors

Actuator		Advantages	Inconveniences	Examples
AC Motors	Synchronous (rotor speed = stator magnetic field speed)	• Precise speed and position control • Speed independent of the load	• Complex start-up	
	Asynchronous (small difference between rotor speed and stator magnetic field speed), also called induction motor	• Robust • Easy start-up • No wear	• Slipping • More complex electronics compared to a synchronous motor	

Mechanics – Energy Sources

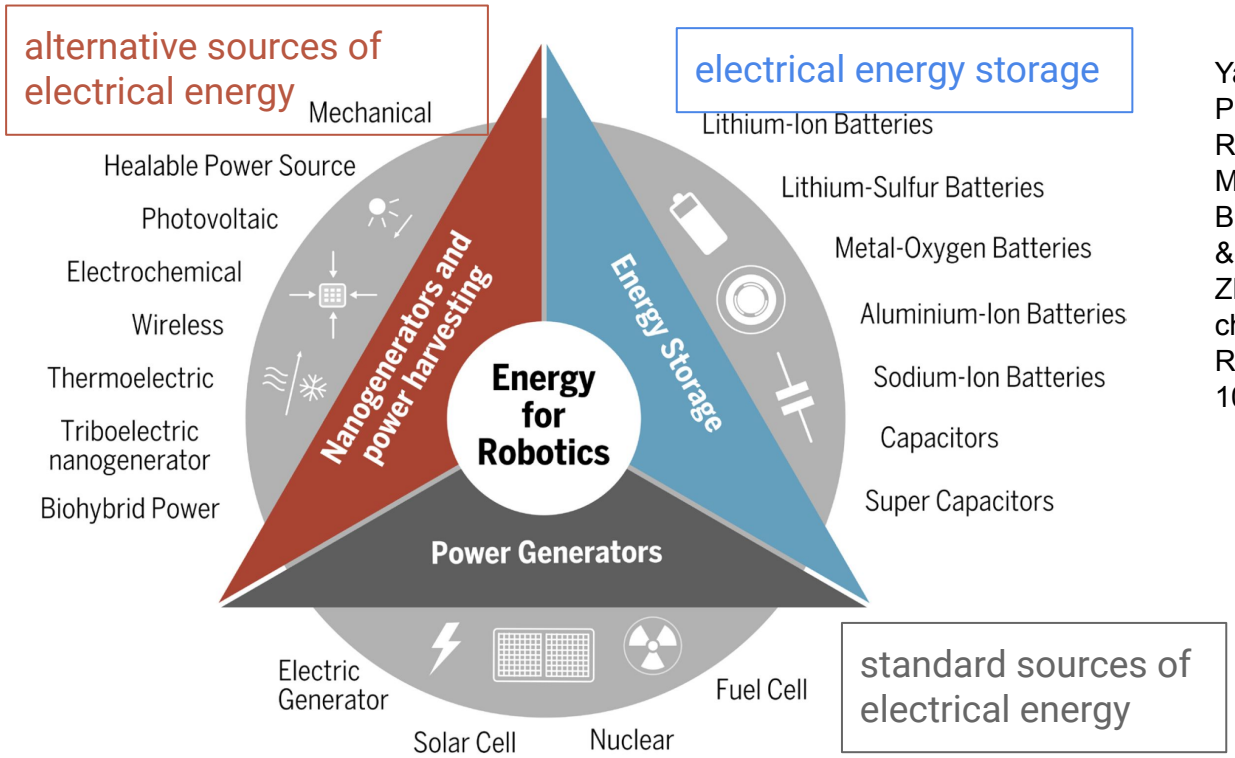


Fig. 4. A summary of different energy sources for robotics. Power generators, which include fuel cells, classical electromagnetic generators, and solar cells. Energy storage, including batteries and capacitors/supercapacitors. Power harvesting and newly developed nanogenerators, as micro-/nano-energy sources, self-powered sensors, and flexible transducers.

Yang, Guang-Zhong & Bellingham, Jim & Dupont, Pierre & Fischer, Peer & Floridi, Luciano & Full, Robert & Jacobstein, Neil & Kumar, Vijay & McNutt, Marcia & Merrifield, Robert & Nelson, Brad & Scassellati, Brian & Taddeo, Mariarosaria & Taylor, Russell & Veloso, Manuela & Wang, Zhong & Wood, Robert. (2018). The grand challenges of Science Robotics. Science Robotics. 3. eaar7650. 10.1126/scirobotics.aar7650.

Mechanics – Electric Energy Storage

What to consider :

- **Charge / discharge time [s]**: give the autonomy of the robot, ideally high discharge over charge ratio
- **Operating temperatures [C]**: must be considered for robots operating in special conditions
- **Operating voltages [V]**: determines which actuators and electronics can be used
- **Life [cycles]:** consider number of cycles needed for the application, progressive decrease in performance
- **Power Density [W/g]:** determines how much energy can be delivered at a given point in time, w.r.t the weight. Must be higher than the maximal consumption of all components (actuators, electronics, etc…).
- **Energy Density [Wh/g]:** determines the overall energy that can be delivered over one charge, w.r.t the weight. Should be high to guarantee best autonomy
- **Pulse load [A]:** determines the maximal current that can be delivered and must be sufficient for all components

Types :

- **Capacitors :** very fast charge and discharge, but low capacity
- **Super Capacitors :** when high output power density is needed and quick recharges are possible
- **Fuel Cells :** fuel in (hydrogen) and electricity out, 40-60% energy efficient
- **Batteries** : long missions, long down time, power density lower than capacitors

Mechanics – Electric Energy Storage

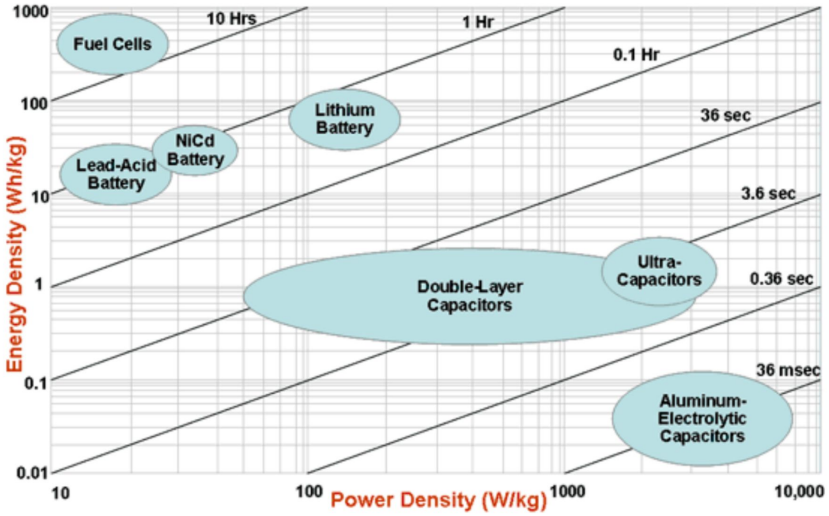


Figure 6. *A Ragone chart storage device energy density versus power density on a log-log coordinate system, with discharge times represented as diagonals.*

Artal, J. S., Dominguez, J. A., & Caraballo, J. (2012, March). Autonomous Mobile Robot with Hybrid PEM Fuel-Cell and Ultracapacitors Energy System, Dedalo 2.0. In *International Conference on Renewable Energies and Power Quality, Santiago de Compostela, Spain* (pp. 1-6).

Characteristics / Parameters	High Power Stationary Storage Technologies			
	Lithium-ion	*Flywheel*	*Super-capacitor*	*SMES*
Energy Density (Wh/kg)	70 - 200	10 - 50	0.5 - 5.0	1 - 10
Energy Density (kWh/m^3)	200 - 600	20 - 100	4.0 - 10.0	0.2 - 2.5
Power Density (W/kg)	150 - 500	500 - 4000	1000 - 10000	500 - 2000
Power Density (MW/m^3)	0.4 - 2.0	1.0 - 2.5	0.4 - 10.0	1.0 - 4.0
Efficiency (%)	90 to 97	90 to 95	90 to 97	95 to 98
Discharge time	Approx. 1 to 8 hr.	8 sec. to 15 min.	milliseconds to 1 hr.	Up to 30 min.
Operating Temp.(° C)	- 20 to 60	- 40 to 40	- 40 to 65	- 50 to 60
Self discharge rate (%/day)	0.1 to 0.3	20 to 100	20 to 40	10 to 15
Lifespan (years)	5 to 15	15 to 20	10 to 30	20 to 30
Lifecycle (cycles)	1000 to 10000	20000 to 21000+	10000 to 50000+	100000+

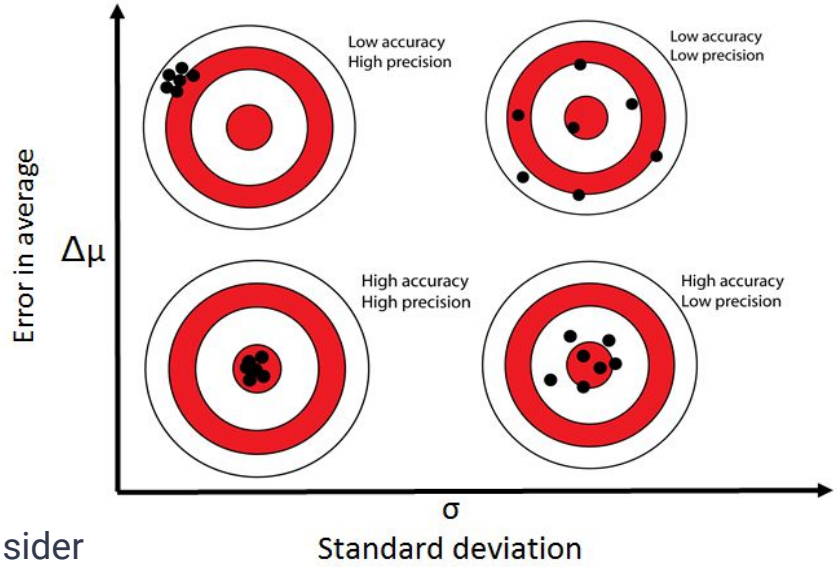
Lachuriya, A., & Kulkarni, R. (2017). Stationary electrical energy storage technology for global energy sustainability: A review. *2017 International Conference on Nascent Technologies in Engineering (ICNTE)*, 1-6.

Table of Contents

Sensors – What are they?

A sensor measures a physical quantity by converting it into a signal which can be read (processed) by an observer (an electronic system).

An **ideal sensor** is
- Only sensitive to the measured property
- Not sensitive to other properties it is exposed to
- Does not influence the measured property
- Has a clearly defined relationship between the measured property and its output signal (e.g. proportional)

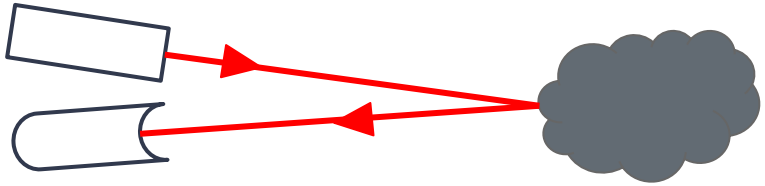


Unfortunately sensors **are not ideal**, that is why we must consider
- **Accuracy** : conformity between the measurement and true value
- **Precision** : random spread of measured values around the average measured values
- **Resolution** : smallest input change that can be detected (due to noise, physical limitations or determined by the A/D conversion)

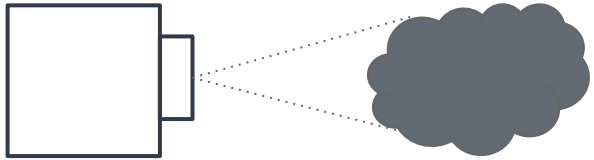
Sensors – Main Characteristics

Active vs. Passive

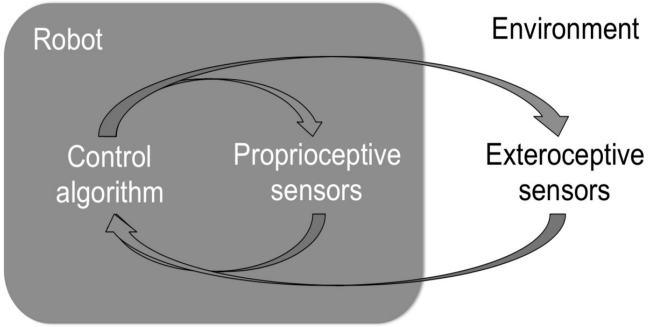
Active : emit their own energy & measure the reaction ; rely less on the environment, but has some influence on the environment and generally consumes more energy



Passive : energy from the environment



Proprioceptive vs. Exteroceptive



Proprioceptive : measure the internal state of the system (motor speed, joint position, battery status...)

Exteroceptive : information about the environment (distances, light intensity...)

Sensors – Other Properties

Range (or full scale)

Lower and upper limits of what may be measured.

Bandwidth or Frequency

- The **speed** at which a sensor can provide a stream of readings
- Usually there is an **upper limit** depending on the sensor and the sampling rate
- One has also to consider **delay** (phase) of the signal

Dynamic Range

- Ratio between the full scale and the smallest reasonable value (noise floor)
- often specified in decibels (ratio between powers)

$$D = 20 \log_{10}((x_{max} - x_{min})/x_{min})$$

Sensitivity

- **Ratio** of output change to input change

Linearity

- Form of the variation of output signal with respect to the input signal
- Sometimes expressed as % of the deviation from a linear behavior from the full scale

Cross-sensitivity (and cross-talk)

- sensitivity to other **environmental** parameters
- influence of other active **sensors**

Sensors – Errors

Systematic errors: deterministic

• caused by factors that can (in theory) be modeled and predicted
• e.g. distortion caused by the optics of a camera

Random errors: non-deterministic

• no prediction possible
• however, they may be described probabilistically
• e.g. hue instability of camera, black level noise of photoreceptors, non-returning echoes in ultrasonic sensors, etc.

Others

• cross-sensitivity (or cross-talk) of sensors (sensitivity to other environmental parameters or influence of other active sensors), motion blur
• rarely possible to model -> appear as random errors but are neither systematic nor random
• systematic errors and random errors might be well defined in controlled environment. This is often not the case for mobile robots!

Sensors – Use Cases

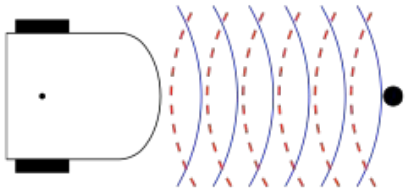
Motion (relative localisation)

- Accelerometer (acceleration and inclination, noisy)
- Gyroscope (changes in orientation, drifts)
- IMU (accelerometer, gyroscope, magnetometer)
- Wheel / Motor incremental encoders (relative, high resolution, two signals are necessary for direction detection, homing is necessary for absolute positioning)
- Wheel / Motor absolute encoders (limited resolution)

Object Detection (obstacle avoidance)

TOF Sensors

- IR Proximity sensor (directional and precise)
- Ultrasonic sensors (conical beams)
- Laser scanner (high field of view and range, precise), e.g. LIDAR - expensive
- Radars - similar to LIDAR but with radio waves, expensive



Tactile : pressure sensor, bumpers...

Sensors – Use Cases

Absolute Localisation

- GPS (outdoor, limited resolution)
- Laser scanner (uses feature matching in controlled environments)
- Beacon based positioning (triangulation, indoor)
- Magnetometer (compass -> absolute orientation)

Vision (object detection, feature extraction, feature based localisation)

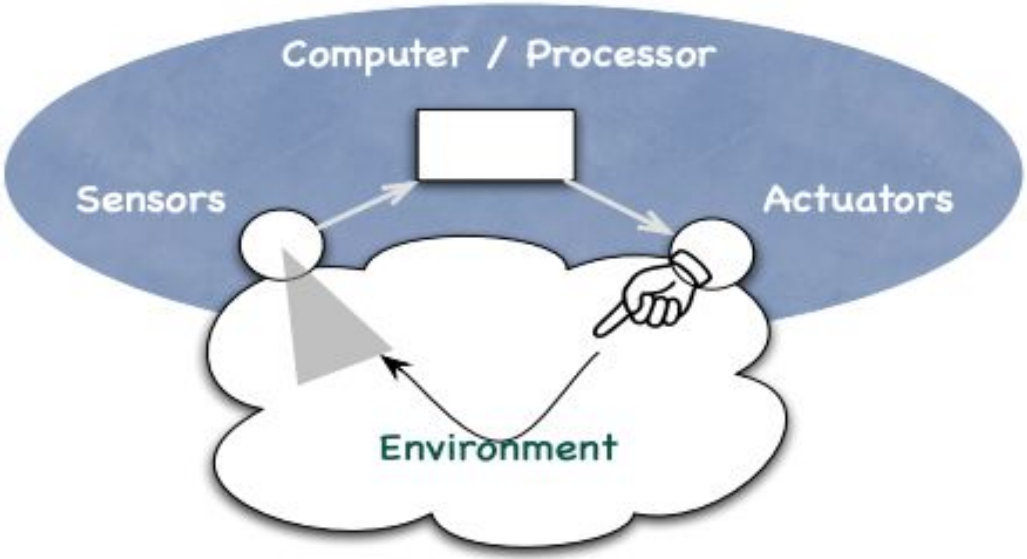
- 2D Cameras
- Depth Cameras
- Structured Light

Architectures – Sensor Integration In a Robotic Syst.

The term robot architecture is often used to refer to two related, but distinct, concepts. Architectural structure refers to how a system is divided into subsystems and how those subsystems interact. The structure of a robotic system is often represented informally using traditional boxes and arrows diagrams ... In contrast, architectural style refers to the computational concepts that underlie a given system. (Handbook of Robotics, 2016)

- A well designed **architecture needs to be defined at the very early stage of system development!**

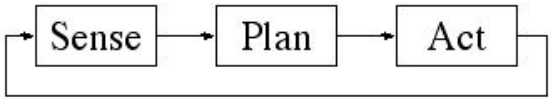
- All robotic systems use their own architectural structure and style.
- A single robotic system will often use several styles together
- Well-conceived, clean architecture can have significant advantages in the specification, execution, and validation of robotic systems



Architectures – Robotic Control Styles

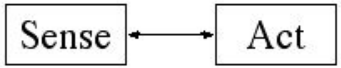
Deliberative control
"think hard, then act"

The robot takes all available **sensory information and all knowledge it has to create a plan of action** by search through potentially all possible plans. This can take a long time but allows the robot to act strategically.



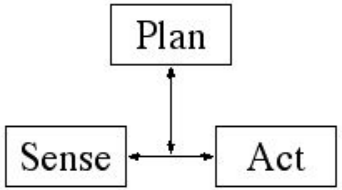
Reactive control
"don't think, (re)act"

Uses a **tight coupling between sensory inputs and effector outputs** to respond quickly to changing and unstructured environments. This type of control does not take advantage of any knowledge about the environment, have any memory or ability to learn over time.



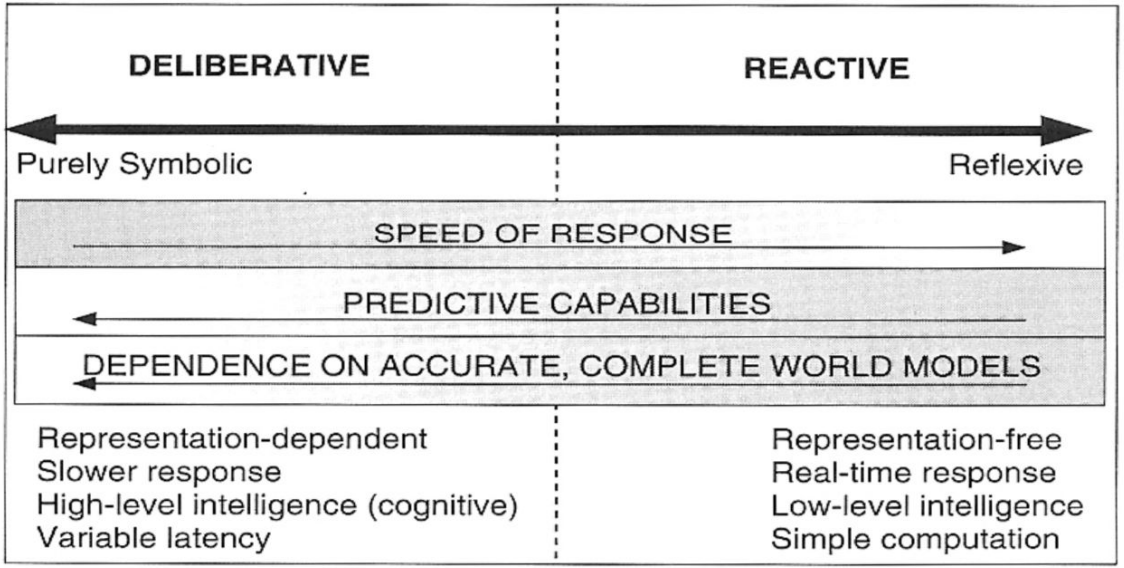
Hybrid control
"think and act in parallel"

Combines the **best of both reactive and deliberative control architectures**. One part of the "brain" plans while the other handles immediate reactions in order to avoid obstacles for example. The coordination between the two parts of the "brain" must handled by a third. That is why these systems are often callec



Architectures – Robotic Control Styles

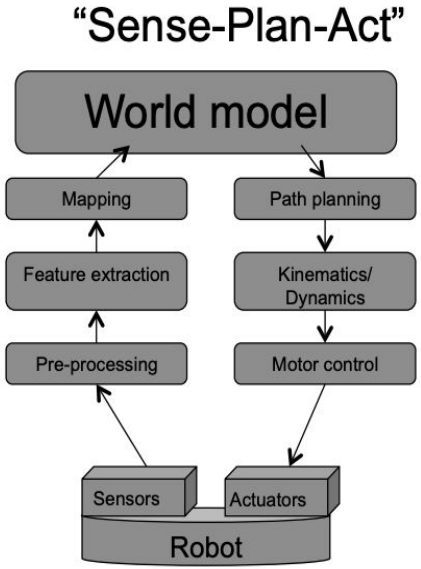
Model-based

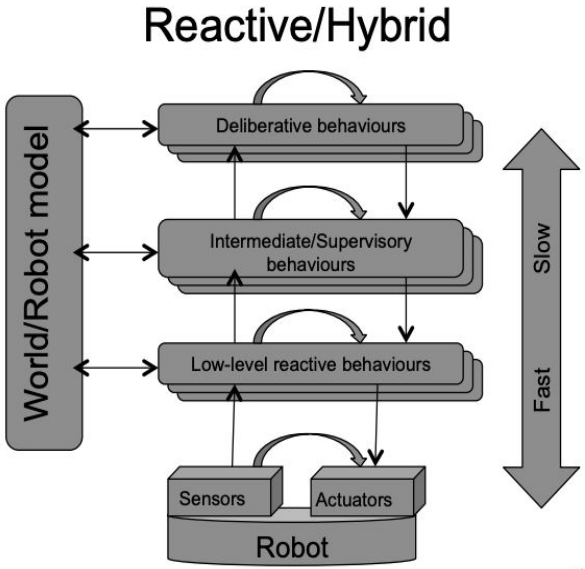


Behavior-based

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

Architectures – Robotic Architecture Styles



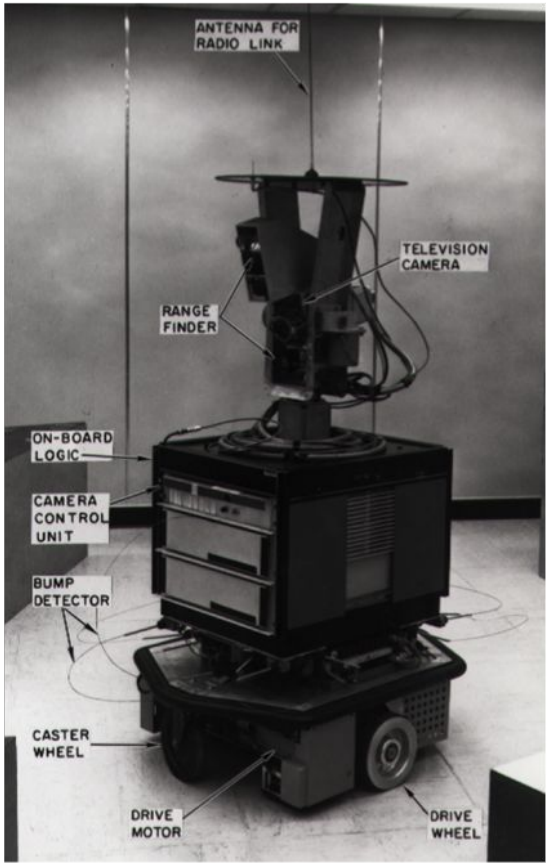


- Top down fashion, heavy on planning
- Sense the world, plan the next action, act at each step
- All sensing data gathered into one global world model

1. Deliberative planning to decompose the task into sub tasks
2. execute behaviours reactively, which is to say, in a succession of sense act couplings

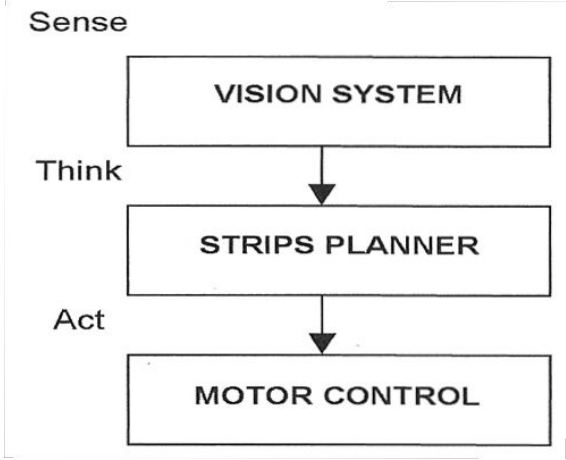
Sensor data gets routed to each behaviour that needs that sensor, but is also available to the planner for construction of a task-oriented global world model.

Architectures – Deliberative Architecture



"Shakey", one of the first mobile robot with the ability to perceive and reason about it surroundings (1972) *Wikipedia*

At the time of Shakey the 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): **sensing, planning and executing the plan.**



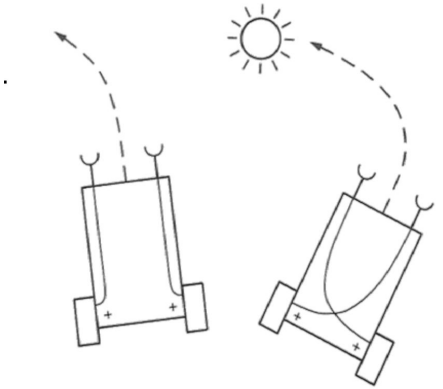
Architectures – Reactive Architecture

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.

Examples : Braitenberg, feedforward neural networks, behaviour based architectures

Here : Braitenberg applied to the Thymio robot to mimic neural networks for obstacle avoidance using the horizontal proximity sensor (Ben-Ari, Mondada, 2018)

Behaviour Based Architectures are a type of reactive architecture and are used when the real world cannot be accurately characterized or modeled.There is a big emphasis on on the importance of coupling sensing and action tightly. This is done by decomposing behaviours or situation-action pairs into contextually meaningful units, the granularity of which may vary. This can be represented by a Finite State Machine that is defined by a set of states, transition conditions and actions that take place when changing state



Braitenberg (1984) Vehicles:

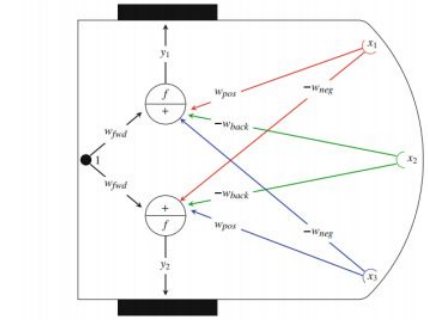


Fig. 13.4 Neural network for obstacle avoidance

Architectures – Reactive Architecture Example

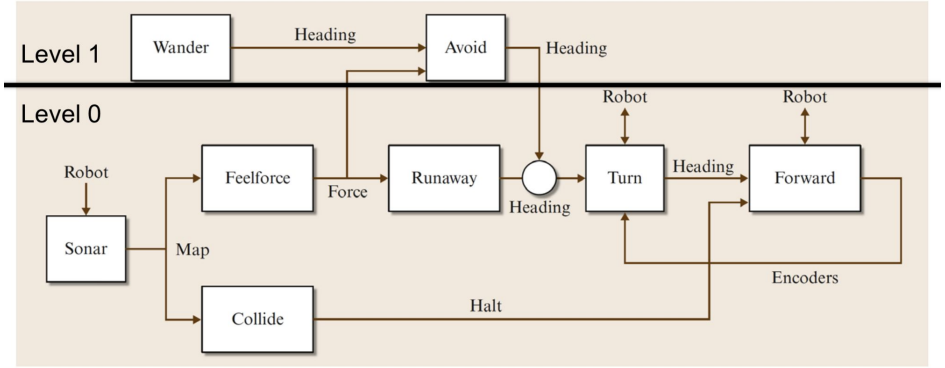
The **Subsumption Architecture** (Brooks, 1986) is a reactive architecture heavily associated with behaviour based robotics.

Brooks 1986, when designing an autonomous robot for offices did

1. The lowest level (layer 0) of control so that the robot did not come in contact with other objects
2. The second level (layer 1) of control, when combined with the lowest, imbues the robot with the ability to wander around aimlessly without hitting obstacles.
3. The third layer (level 2) is meant to add an exploratory mode of behavior to the robot, using visual observations to select interesting places

Principles

- The architecture is built incrementally
- Start by building in lowest level of competence
- Validate on robot, debug, adjust, validate, adjust, …
- Robot is immediately operational



Architectures – Hybrid Reactive Architecture

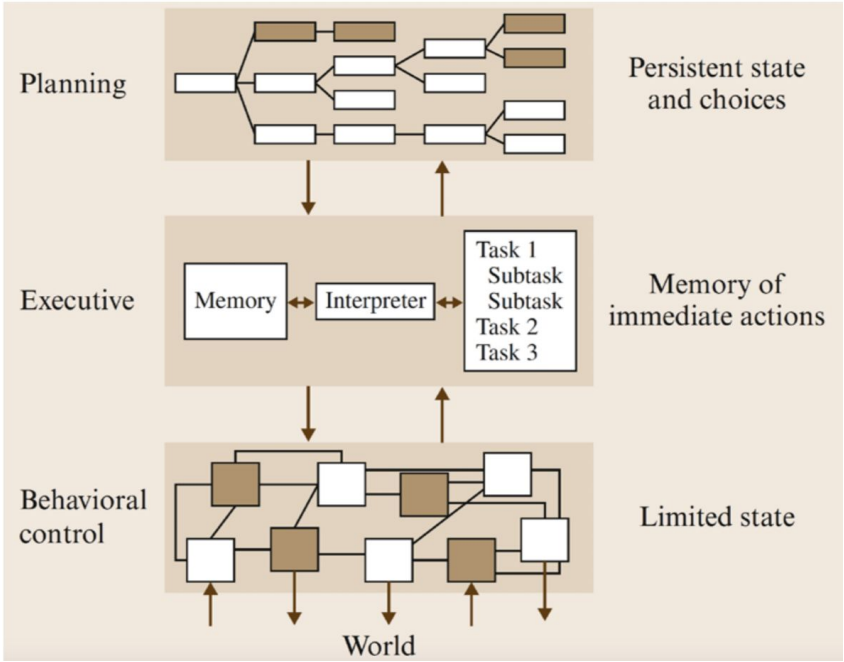
Combines the responsiveness, robustness, and flexibility of purely reactive systems with more traditional symbolic/deliberative methods.

Reasons:
• 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).
• Purely symbolic/deliberative system lack reactivity.

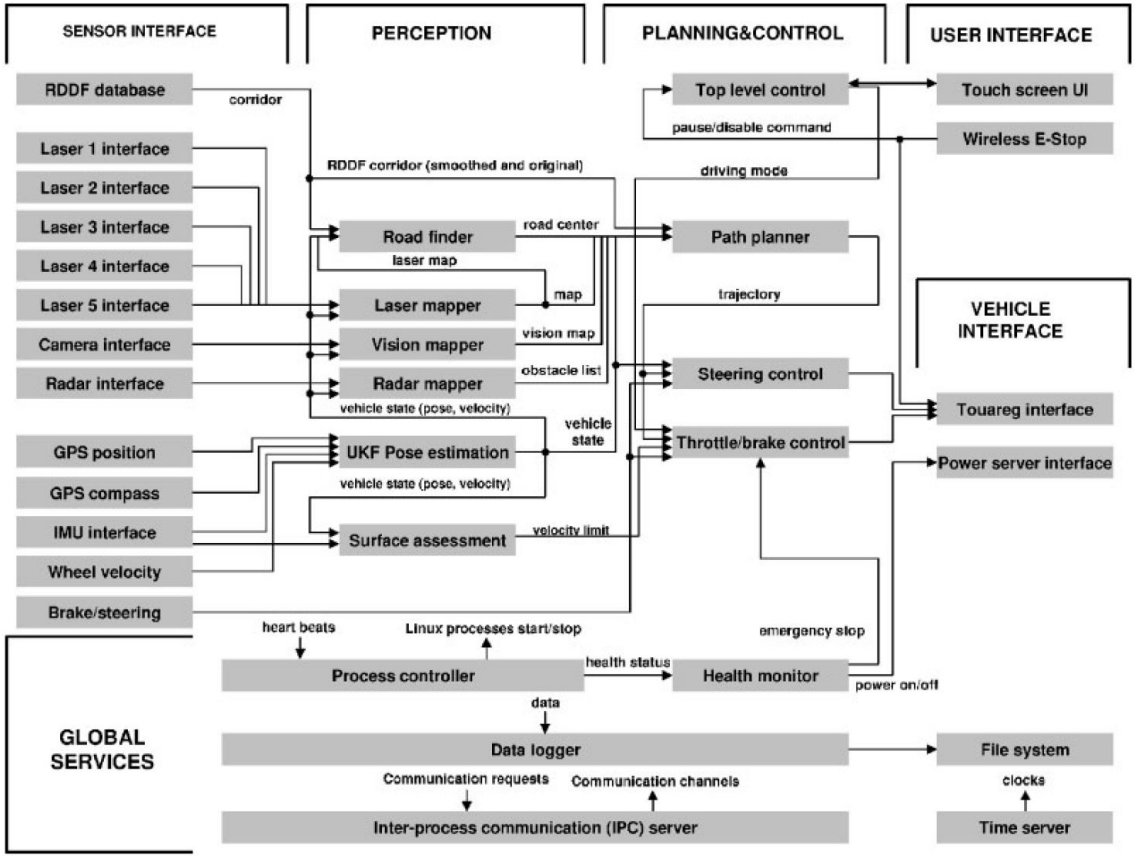
Planning Layer : handles long range activities, high level goals. Plans algorithms and does replanning on demand, based on the monitoring of task execution.

Executive Layer : at the interface between the deliberative and reactive layers. Translates high-level plans in sequences of behaviors, monitors executions, handles exceptions.

Behaviour-Based Layer



Architectures – Structure



Thrun, Sebastian, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong et al. "Stanley: The robot that won the DARPA Grand Challenge." *Journal of field Robotics* 23, no. 9 (2006): 661-692.

This week’s exercises

Part A : mechanics

1. Degrees of freedom and of mobility
2. Redundant systems

Part B : sensors

1. Sensors overview
2. Accelerometers

MOBILE ROBOTS 2+3 - Perception, Vision and Machine Learning

Prof. Francesco Mondada

EPFL

BIBLIOGRAPHY AND SOURCES

Introduction to Autonomous Mobile Robots (ch. 4) R. Siegwart, and I. Nourbakhsh, MIT Press, 2011

Elements of Robotics (ch.12), M. Ben-Ari, F. Mondada, Springer, 2017.

Mobile Robots - EPFL - J.-C. Zufferey, Felix Schill, 2013

Springer Handbook of Robotics (ch. 32-34), B. Siciliano, and O. Khatib (Eds.), 2nd edition, Springer, 2016.

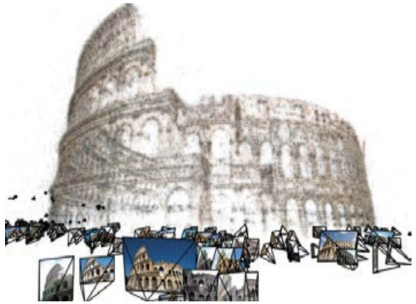
Elements of Robotics (ch.13-14), M. Ben-Ari, F. Mondada, Springer, 2017.

Reinforcement learning, Sutton & Barto, MIT Press,1998.

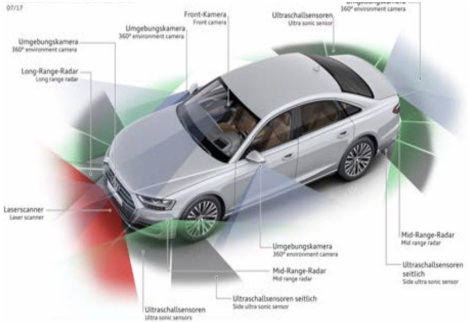
Table of Contents

Computer Vision

« Computer vision is the **science and technology of machines that see**, where see in this case means that the machine is able to extract information from an image that is necessary to solve some task. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. » Wikipedia



3D Reconstruction

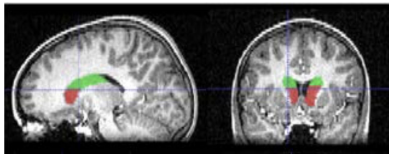


Autonomous Navigation

Security Applications

Performance Analysis

Quality Control



Biomedical imaging for organ segment.

And for Mobile Robotics?

Camera calibration

Estimates the parameters of the lens to correct for distortions, used to measure the size or displacement of objects in world from the camera scene

Camera localisation (or extrinsic calibration)

Estimate the pose of the camera in the real world

Object pose estimation and tracking

Estimate and track the position of known or moving objects in the image

3D reconstruction (e.g. stereo vision)

From successive images taken from different points of view that are known, estimate the distance to the different objects in the scene

SLAM (simultaneous localisation and mapping)

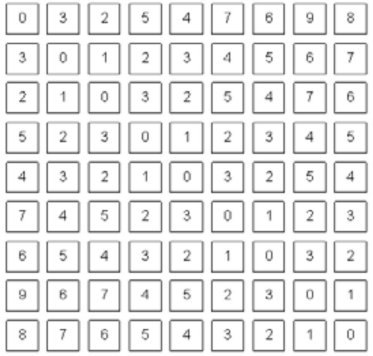
From unknown map and unknown camera poses, progressively construct the map of the environment while navigating through it

Cameras

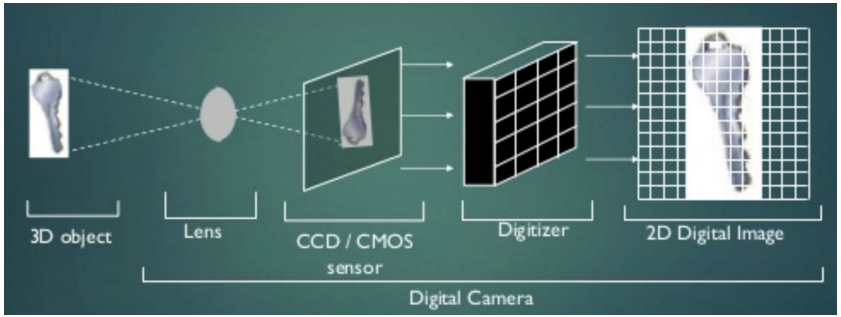
Cameras are everywhere. Why?

- Cheap in fabrication cost
- Cheap in energy consumption
- Versatile (multiple modalities, projections)
- Provide rich information on the environment
- Passive sensor
- Fast (some well over 250 fps)

What we see



What the computer sees



From an Image to Real World Information

To understand the scene, computer vision can rely on

2D aspects: known 2D patterns to understand the image, all the while considering that a good feature needs to be easy to extract and easy to match.

- Low-level features (geometric primitives) like points, lines, circles
- Higher-level features (objects recognition) like objects or places

3D aspects: use our knowledge about the image formation (optics, scene…) to come back to 3D information

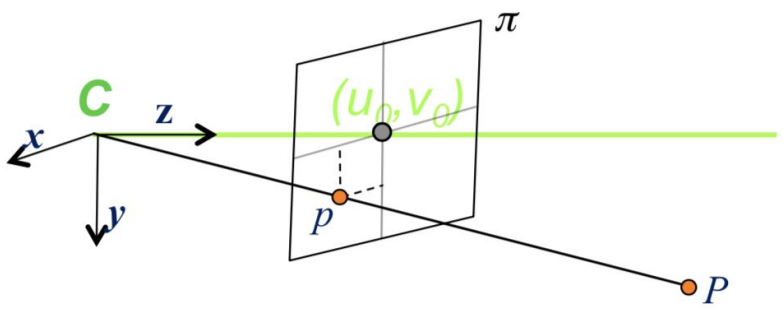
- Camera calibration
- Camera localization
- Object pose estimation/tracking
- 3D reconstruction
- SLAM (simultaneous localization and mapping)

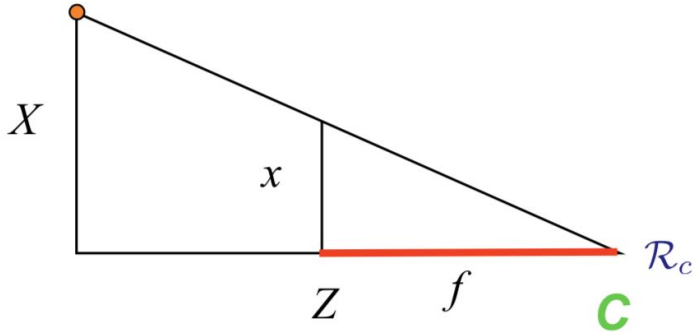
Table of Contents

From 3D to 2D - Perspective Projection

Definitions

- C is the center of projection
- **x** axes is parallel to image lines and **y** axes is parallel to image columns
- Intersection of **z** axes with image plane π is the principal point u_0, v_0
- focal distance $f = d(C,\pi)$





Credit: Eric Marchand INRIA Rennes

From 3D to 2D - Perspective Projection

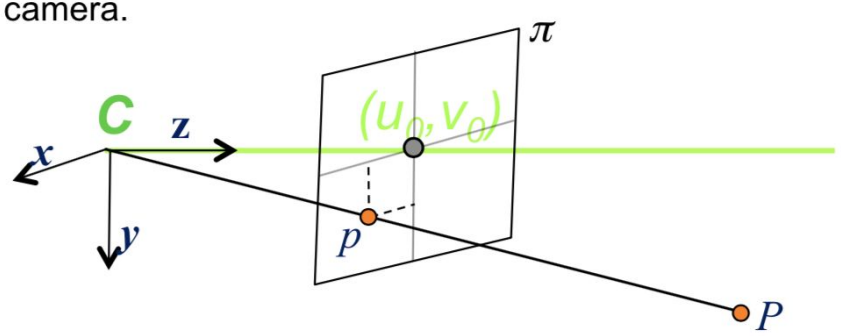
The perspective projection of a point X = *(X,Y,Z)* is the image point x = *(x,y)* with:

$$x = f\frac{X}{Z}, \qquad y = f\frac{Y}{Z}$$

All the points on the CP straight line given by:

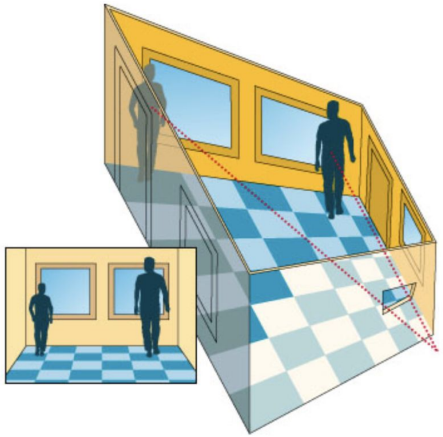
$$\begin{cases} fX - Zx = 0 \\ fY - Zy = 0 \end{cases}$$

are projected on the same point x. It is impossible to determine the position of P from p without *a priori* knowledge with one camera.



Credit: Eric Marchand INRIA Rennes

From 3D to 2D - Perspective Projection



2D Feature Processing

1. **Filtering to reduce noise in the image.**
2. **Preprocessing / Feature extraction**
    a. Edge detection
3. **Feature Detection**
    a. Hough Transform
    b. Pattern Matching
    c. Interest Point Detectors

or

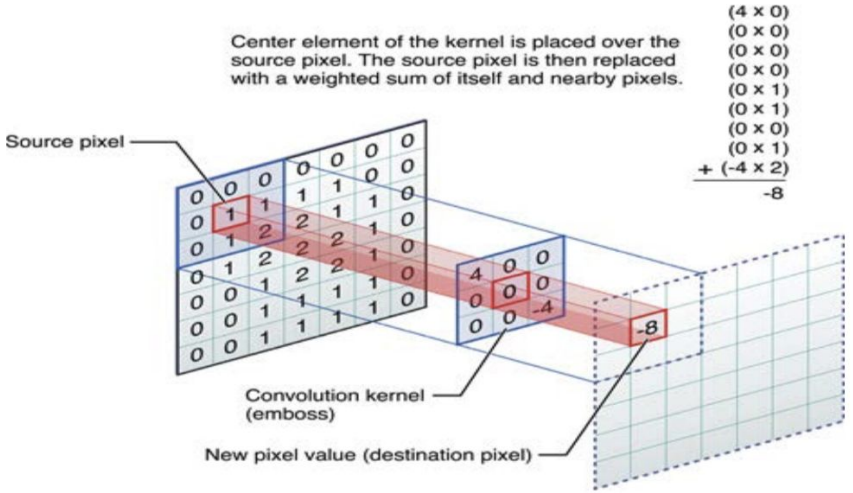
1. **Machine Learning** to train an algorithm to identify a particular feature

Filtering and Smoothing to get a good image

Gaussian smoothing

- Removes high-frequency noise
- Convolution* of intensity image I with a Gaussian kernel G:

$$\hat{I} = G \otimes I \qquad G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

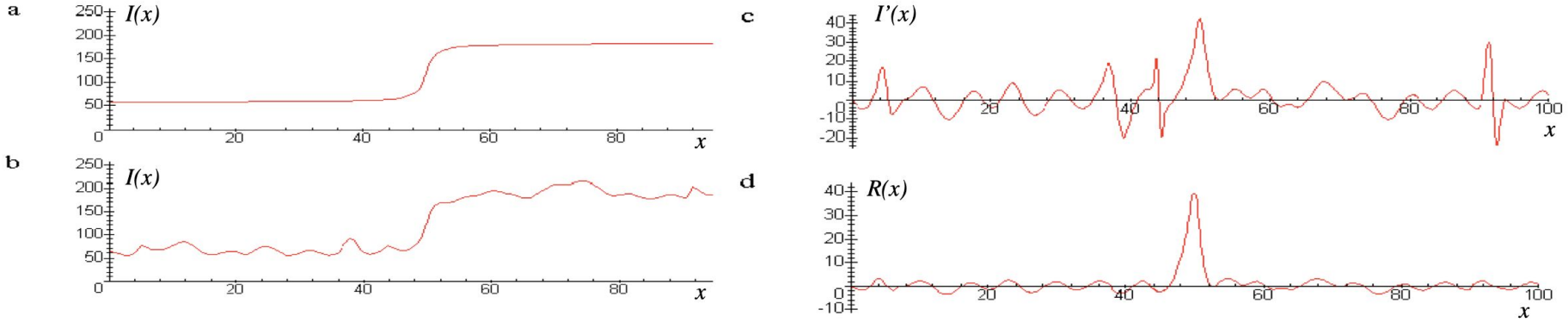


A convolution is an integral that expresses the amount of overlap of one function g as it is shifted over another function f:

$$f \otimes g \equiv \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} g(\tau)\, f(t-\tau)\, d\tau$$

22

Filtering and Smoothing to get a good image



a. Intensity 1D profile of an ideal step edge.

b. Intensity profile $I(x)$ of a real edge (with noise).

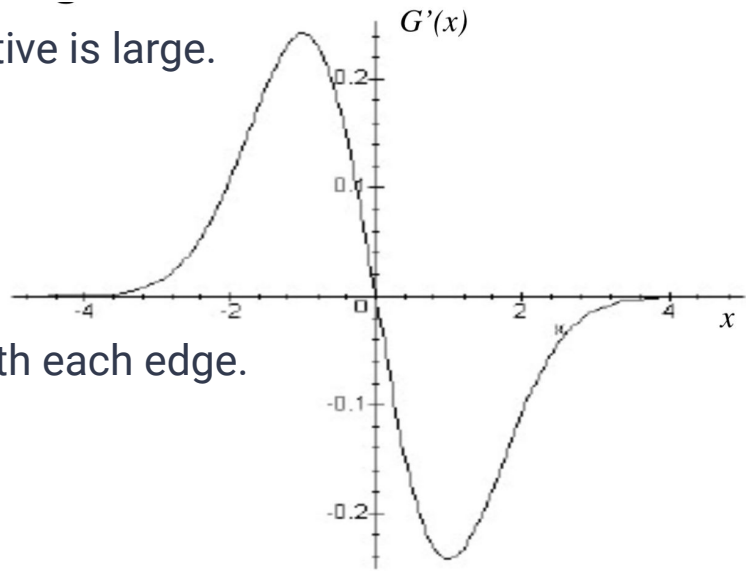
c. Its derivative $I'(x)$.

d. The result of the convolution $R(x) = G' \otimes I$, where G' is the first derivative of a Gaussian function.

Edge Detection (or Enhancement)

Goal: spot locations where the brightness undergoes sharp changes.
After filtering the image :

1. Differentiate one or two times the image.
2. Look for places where the magnitude of the derivative is large.



Example : the Canny Edge Filter an optimal edge detector with respect to the following criteria

- Maximizing signal to noise ratio.
- Achieving highest precision on the edge location.
- Minimizing the number of edge responses associated with each edge.

The processing steps :

- Convolution of the image with a Gaussian kernel G.
- Compute the derivative to find high gradient zones.
- Find maxima in the direction normal to the edge (non-maxima suppression).

Gaussian filtering and derivation can be combined into one operation: $(G \otimes I)' = G' \otimes I$

Edge Detection – Discrete Gradient Operators

Goal: numerically approximate the behavior of the Canny edge detector while reducing the computational cost and, in some cases, computing the edge orientation.

Note: G is the gradient magnitude and "Theta" the gradient direction

- **Roberts**

$$|G| \cong \sqrt{r_1^2 + r_2^2} \; ; \quad r_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \; ; \quad r_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

- **Prewitt**

$$|G| \cong \sqrt{p_1^2 + p_2^2} \; ; \quad \theta \cong \operatorname{atan}\left(\frac{p_1}{p_2}\right) ; \quad p_1 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \; ; \quad p_2 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

- **Sobel**

$$|G| \cong \sqrt{s_1^2 + s_2^2} \; ; \quad \theta \cong \operatorname{atan}\left(\frac{s_1}{s_2}\right) ; \quad s_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \; ; \quad s_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Edge Detection – Discrete Gradient Operators

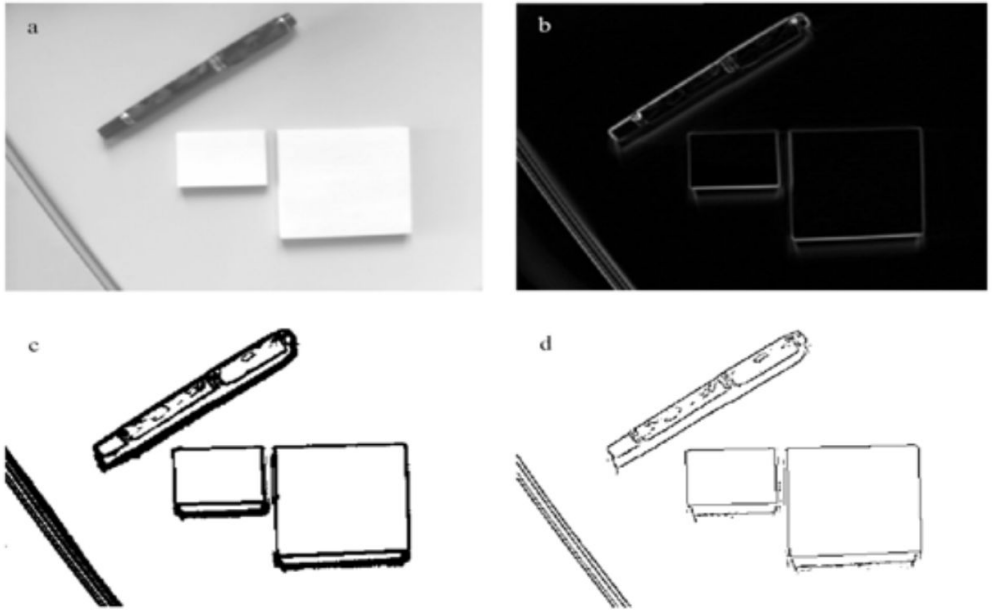
Example of the Sobel Filter

a. Raw image

b. Filtered (Gaussian or Sobel)

c. Thresholding

d. Nonmaxima suppression



Feature Detection – Hough Transform

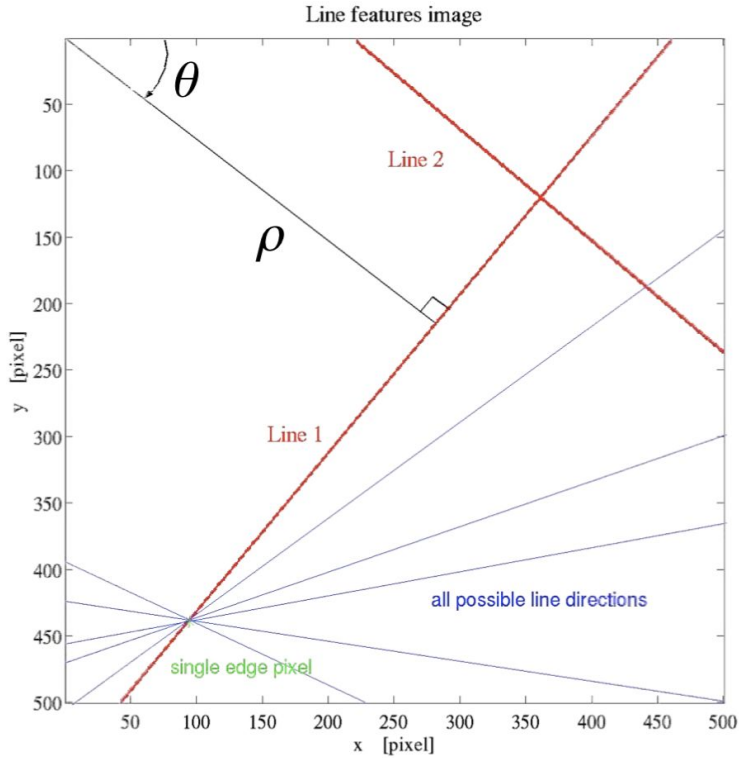
Goal: extract straight edges, which are often used in mobile robotics (e.g. doors, lane).

Each line in the image can be parameterized as:

$$\rho(\theta) = x\cos\theta + y\sin\theta$$

The Hough transform is a 2D histogram in the parameter space (ρ,θ) whose cells count the number of *edge pixels* that a particular line is voting for.

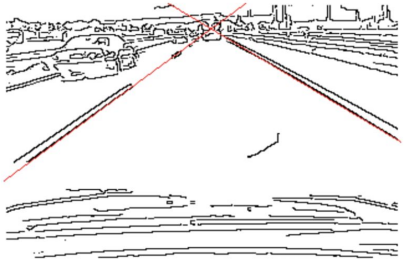
Note: the Hough transformation is assuming a preprocessing step for edge detection.

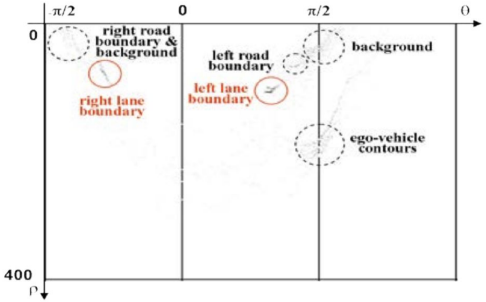


Note: the Hough transform can also be used to detect other shapes than lines depending on the used parameterization.

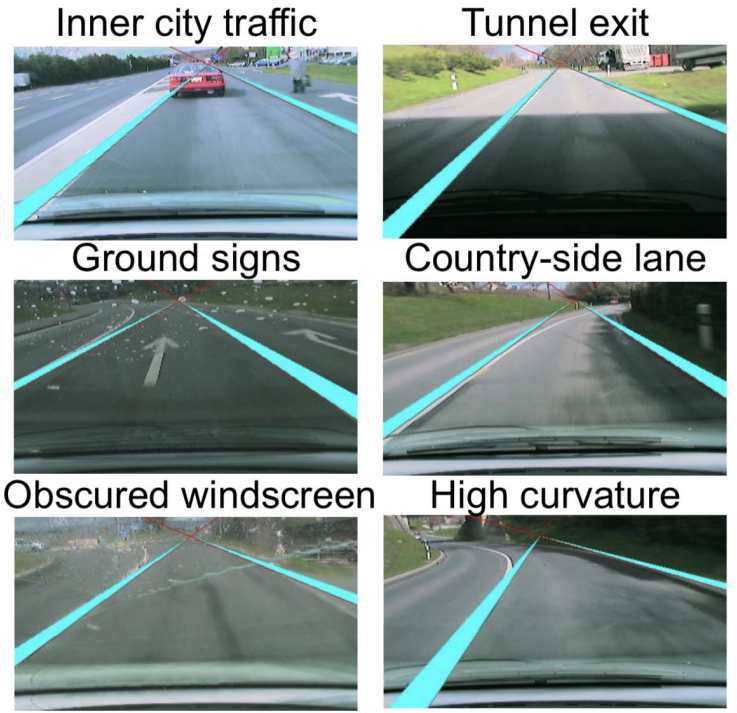
Feature Detection – Hough Transform

A Canny edge pixel map and the resulting Hough transform accumulator array:





Lane boundaries: $(\theta_l, \rho_l) = \{59°, 105\}$, $(\theta_r, \rho_r) = \{-51°, 78\}$



Feature Detection – Pattern Matching

Finding a known pattern in the input, e.g. an image template *I1* in a (larger) image *I2*

Can be computed by either the :

- sum of squared differences which should be minimised
- correlation which should be maximised

Applications :

- **Object recognition & tracking** : searching for the stored patterns in the inputs. Peaks indicate the object position in the image
- **Optic flow :** Match patches of previous image with new image. Normalising the peak by the time difference gives the optic flow
- **Stereo vision :** Match patches of the left image with the right image. Peak corresponds to disparity in the position of the patch

Feature Detection – Pattern Matching

- sum of squared differences:

$$SSD(D_x, D_y) = \sum_{(x,y)} (I_1(x,y) - I_2(x + D_x, y + D_y))^2$$

- correlation:

$$Correlation(D_x, D_y) = \sum_{(x,y)} I_1(x,y) \cdot I_2(x + D_x, y + D_y)$$

where $I(x,y)$ is the irradiance of the image point (x,y), and D_x, D_y are the translation components of the template I_1 across the image I_2

Correlation can be computed faster via Fast Fourier transform:

$$I_1 * I_2 = FFT^{-1}(FFT(I_1) \cdot \overline{FFT(I_2)})$$

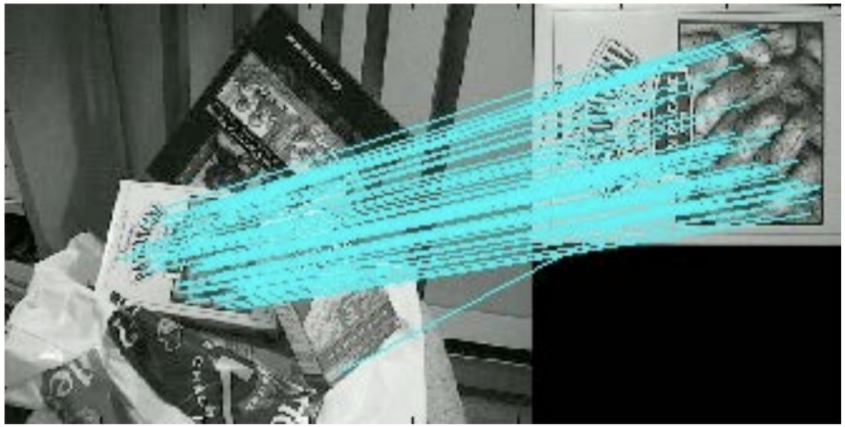
Feature Detection – Interest Point Detectors

Interest point detectors are fundamental for many problems in computer vision:

- object recognition from various perspectives
- 3D structure from multiple images (stereo / optic flow)
- motion tracking

Ideal features (interest points) should be invariant to

- image scaling and rotation
- change in illumination
- camera viewpoint
- highly distinctive (to enable correct match with high probability against a large database of features)



http://people.cs.ubc.ca/~lowe/keypoints/

Feature Detection – Interest Point Detectors

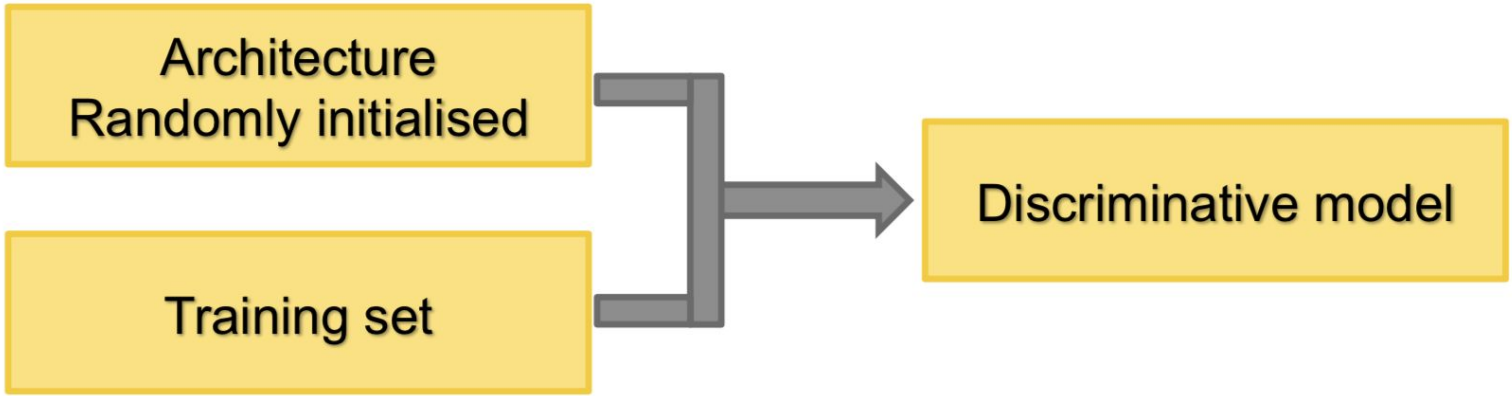
	Corner detector	"Blob" detector	Rotation invariant	Scale invariant	Affine invariant	"Repeatability"	Localization accuracy	Robustness	Computational efficiency
Harris (Harris et al., 1988)	x		x			+++	+++	++	++
Harris-Laplacian (Mikolajczyk et al., 2004)	x	x	x	x		+++	+++	++	+
Harris-Affine (Mikolajczyk et al., 2002)	x	x	x	x	x	+++	+++	++	++
SUSAN (Smith et al., 1997)	x		x			++	++	++	+++
FAST (Rosten et al., 2005)	x		x			++	++	++	++++
SIFT (Lowe et al., 1999)		x	x	x	x	+++	++	+++	+
MSER (Matas et al., 2002)		x	x	x	x	+++	+	+++	+++
SURF (Bay et al., 2008)		x	x	x	x	++	++	++	++

Table of Contents

Machine Learning

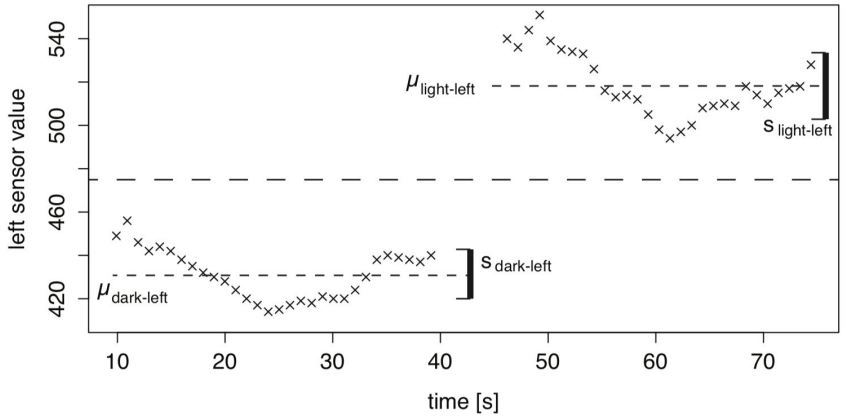
Before: All of the models were hand-crafted: researchers defined approaches to discriminate one pattern from another. Works well for hand-crafted scenes, but in the wild they hardly work as appearances can have many variations.

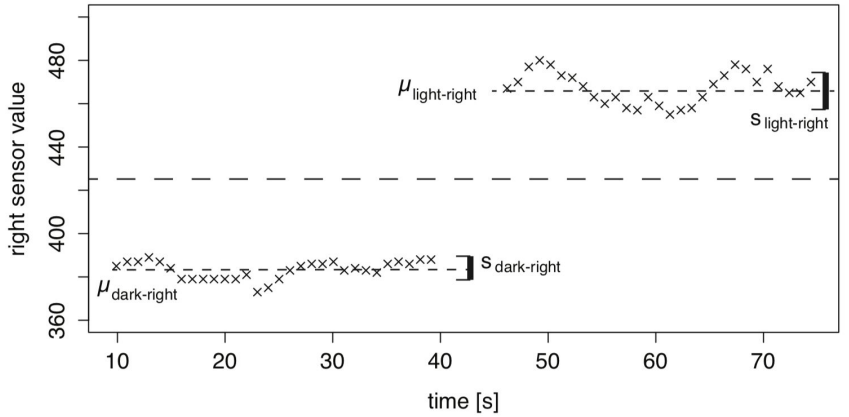
Now: many labeled images are available, it is possible to learn those variations and their discriminative models, which corresponds to finding the optimal parameters for one parametric model (architecture).



Machine Learning : Linear Discriminant

How would we proceed to distinguish between two colors? Try to find a discriminative rule between the sensor values of both colors. This criteria ideally separates well between the elements of different classes (the two colors) minimizes the spread within a given class.





Machine Learning : Linear Discriminant

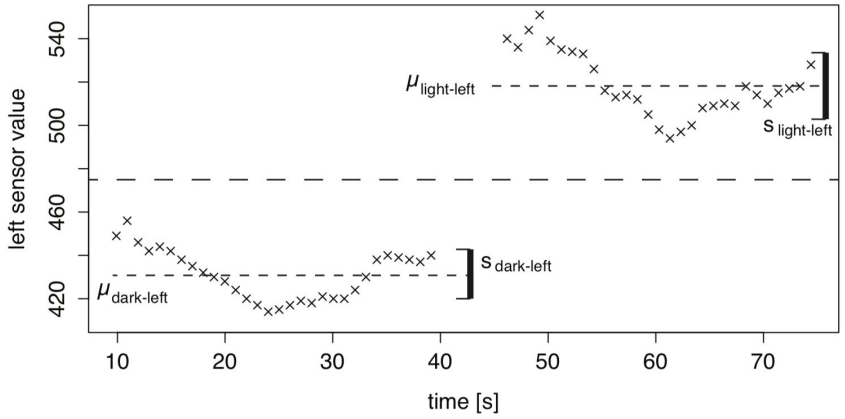
How would we proceed to distinguish between two colors?
Try to find a discriminative rule between the sensor values of both colors. This criteria ideally separates well between the elements of different classes (the two colors) minimizes the spread within a given class.

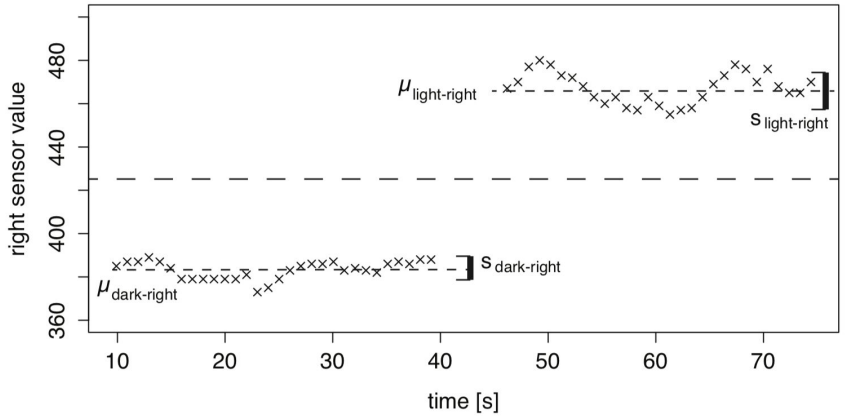
Discriminator:

$$\Delta = \frac{\mu_{dark} + \mu_{light}}{2}$$

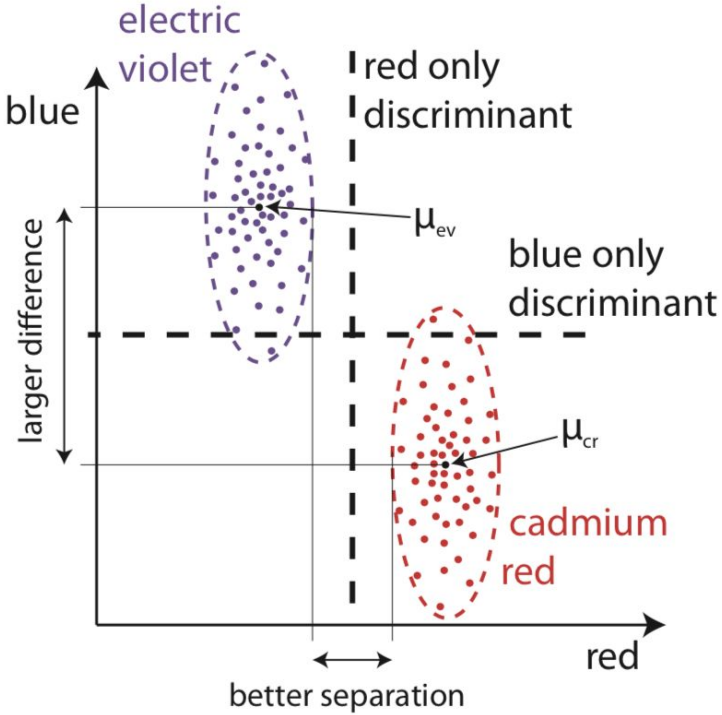
Quality criteria:

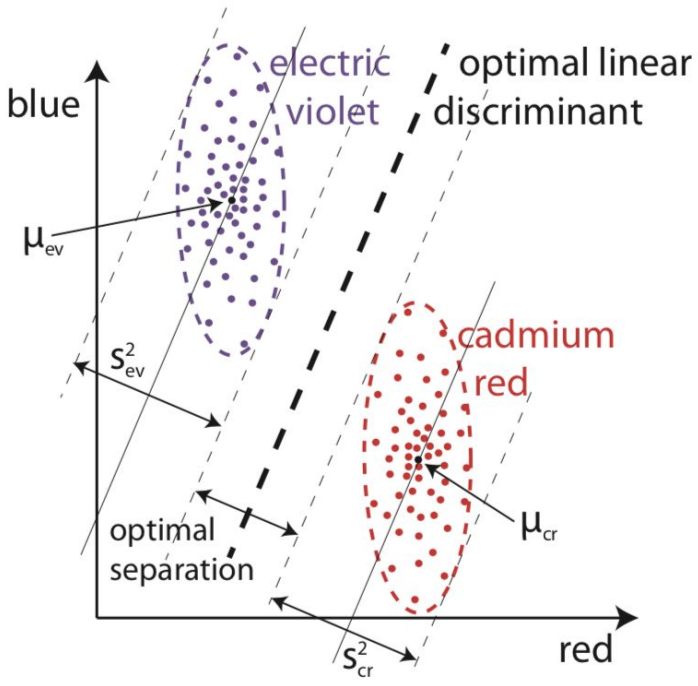
$$J_k = \frac{\left(\mu^k_{dark} - \mu^k_{light}\right)^2}{\left(s^k_{dark}\right)^2 + \left(s^k_{light}\right)^2}$$



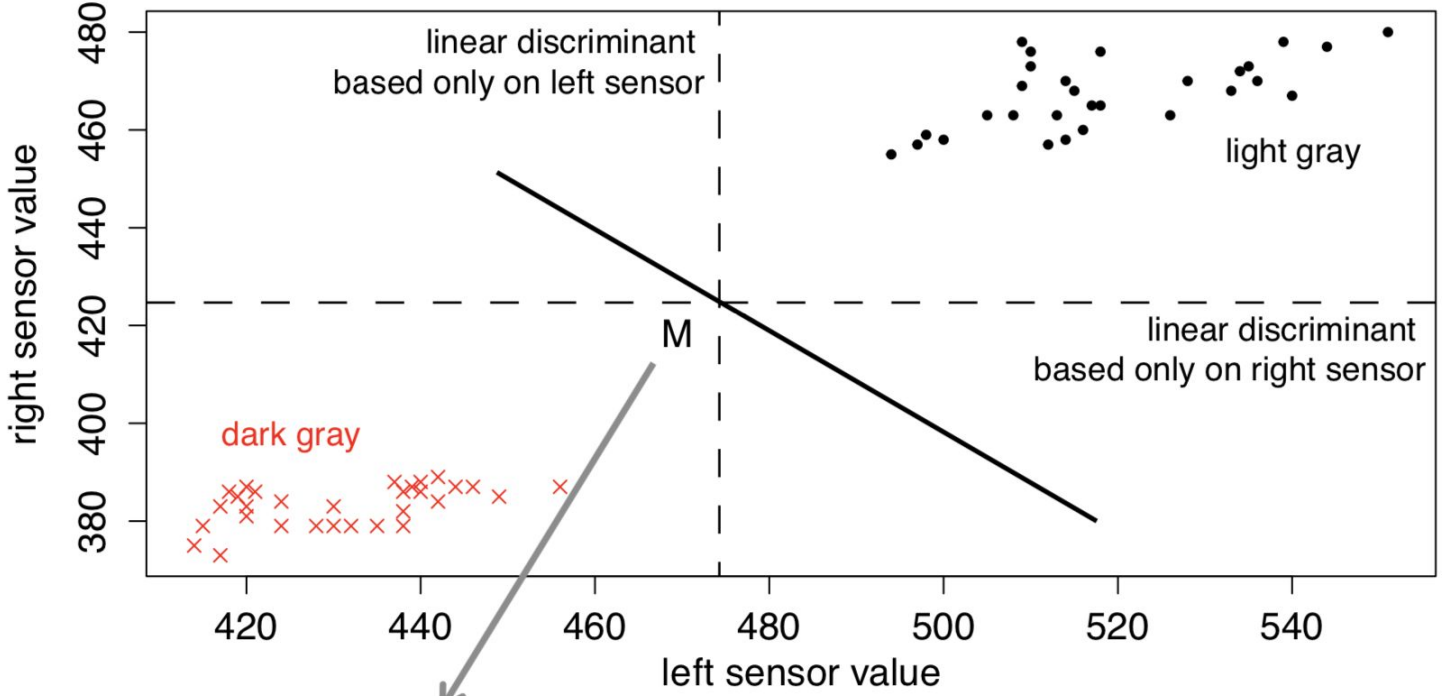


Machine Learning : Linear Discriminant





Machine Learning : Linear Discriminant



$$\left(\frac{\mu_{light}^{left} + \mu_{dark}^{left}}{2}, \frac{\mu_{light}^{right} + \mu_{dark}^{right}}{2} \right)$$

Machine Learning : Linear Discriminant Slope

How to choose the slope? The slope of the discriminant line should maximise the quality criteria J.

The discriminant line will have the form:

$$w_1 x_1 + w_2 x_2 = c$$

where x_1 and x_2 are the two inputs, w_1 and w_2 define the slope and c is a constant ensuring that M is on the discriminant li

By derivation and finding the max (= 0) we get:

$$\mathbf{w} = \mathbf{S}^{-1} (\boldsymbol{\mu}_{light} - \boldsymbol{\mu}_{dark})$$

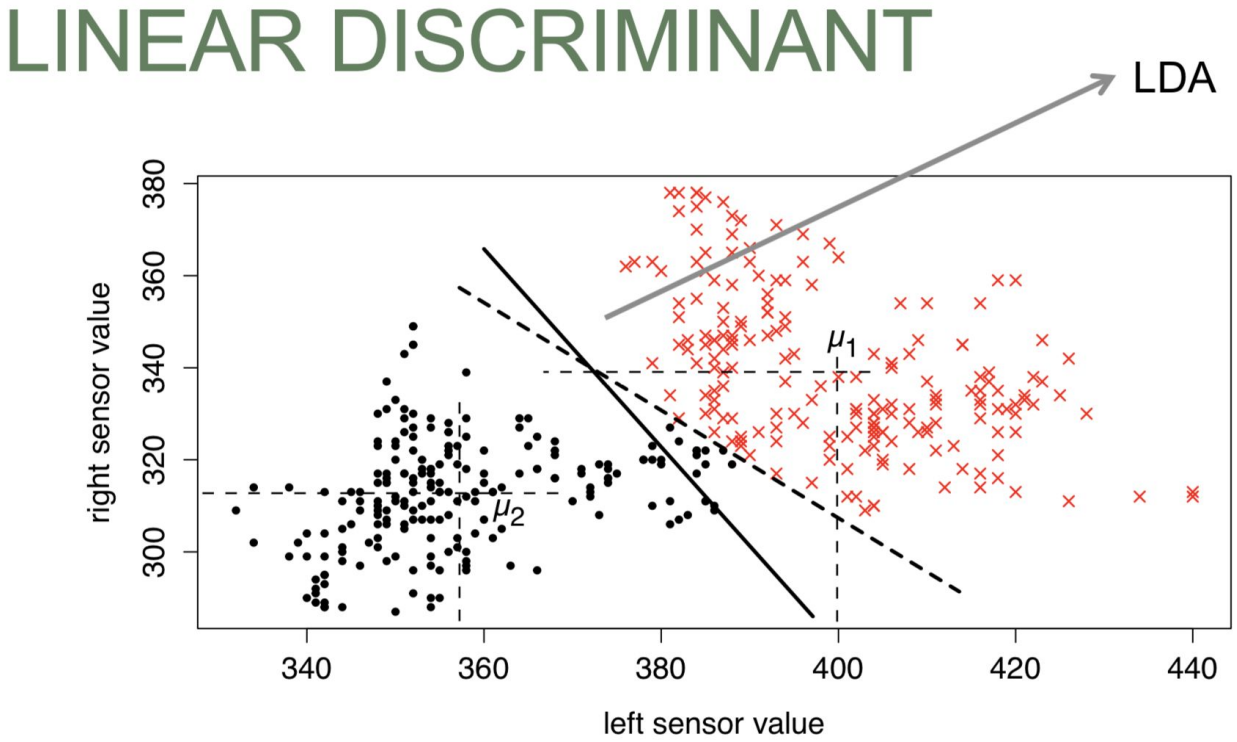
where:

$$\boldsymbol{\mu}_{light} = \begin{bmatrix} \mu_{light}^{left} \\ \mu_{light}^{right} \end{bmatrix}, \quad \boldsymbol{\mu}_{dark} = \begin{bmatrix} \mu_{dark}^{left} \\ \mu_{dark}^{right} \end{bmatrix}$$

are the mean vectors of the two classes and $\mathbf{S}^{-1}$ is the inverse of the average of the covariance matrices of the two classes[6]:

$$\mathbf{S} = \frac{1}{2} \left(\begin{bmatrix} s^2\left(\mathbf{x}_{light}^{left}\right) & cov\left(\mathbf{x}_{light}^{left}, \mathbf{x}_{light}^{right}\right) \\ cov\left(\mathbf{x}_{light}^{right}, \mathbf{x}_{light}^{left}\right) & s^2\left(\mathbf{x}_{light}^{right}\right) \end{bmatrix} + \begin{bmatrix} s^2\left(\mathbf{x}_{dark}^{left}\right) & cov\left(\mathbf{x}_{dark}^{left}, \mathbf{x}_{dark}^{right}\right) \\ cov\left(\mathbf{x}_{dark}^{right}, \mathbf{x}_{dark}^{left}\right) & s^2\left(\mathbf{x}_{dark}^{right}\right) \end{bmatrix} \right).$$

Machine Learning : Opt. Linear Discriminant



Beyond ML : Towards Artificial Neural Nets (ANN)

What if the robot could learn the same way a brain does? 10^{11} neurons..

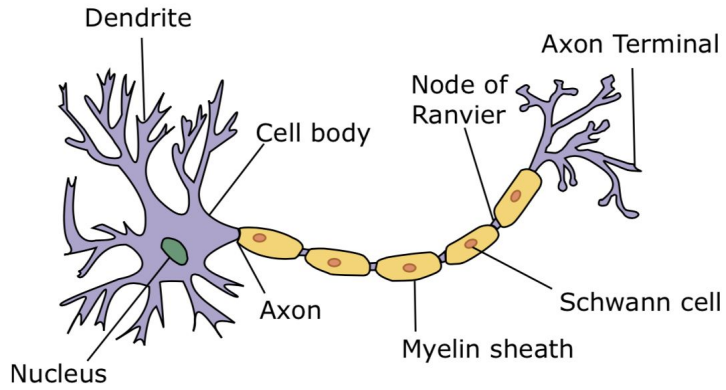


Fig. 13.1 Structure of a neuron

Real Neuron

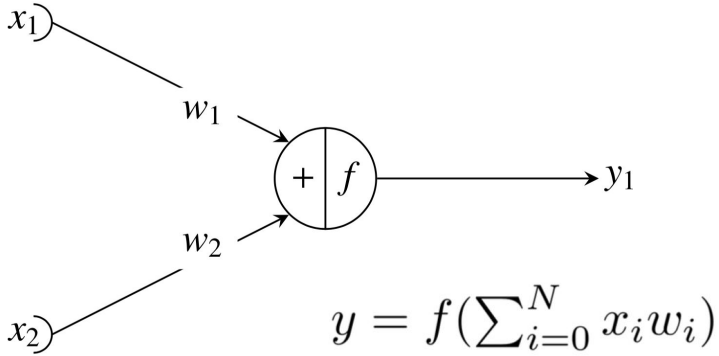


Fig. 13.2b ANN: one neuron with two inputs

Artificial Neuron

Beyond ML : ANN Output Function f

What type of output functions are typically used for ANNs?

$$y = f(\sum_{i=0}^{N} x_i w_i)$$

- Step (binary):

  y = 0 if sum <= 0, y = 1 if sum > 0

- Saturation:

  y = 0 if sum <= 0, y = 1 if sum >= 1, y = sum otherwise

- Sigmoid function:

  $y = 1 / (1 + e^{-sum})$

Beyond ML : ANN Structure

To implement non linear functions, create a structure with multiple layers.

In such a structure, a single neuron with a step function output can be seen as a classifier of the inputs into two classes (outputs 0 or 1). In the case of two inputs, the separation between the two classes is given by the following function:

$$x_1 w_1 + x_2 w_2 = 0$$

This is a linear separator with parameters w_1 and w_2. Often a third fixed input is added:

$$x_1 w_1 + x_2 w_2 + w_3 = 0$$

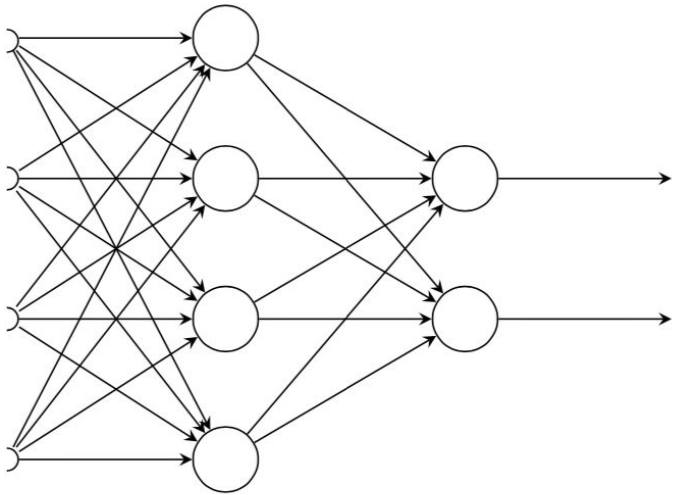


Fig. 13.6a Multilayer ANN

Beyond ML : ANN Structure

To add memory we use recurrent connections:

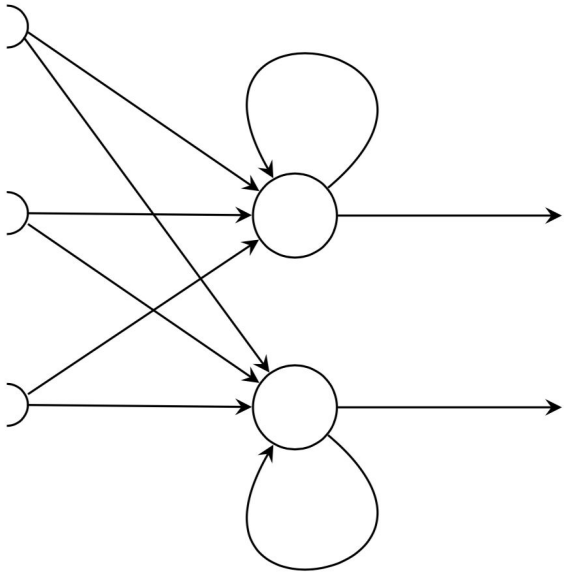


Fig. 13.6b ANN with memory

Beyond ML : ANN Learning

In ML, three types of learning algorithms exist :

• **Supervised learning :** when we know what output is expected for a set of inputs. Requires having a labelled dataset.

• **Unsupervised learning :** when the goal is to learn the underlying structure of the data without relying on the labels. In such cases, the network maps a large number of inputs

• **Reinforcement learning** : we simply tell the network if the output it computes is good or not. This is for instance done by the Hebb rule which positively reinforces behaviours which lead to a good output and “punishes” behaviours which lead to negative outputs.

When there is a good action:

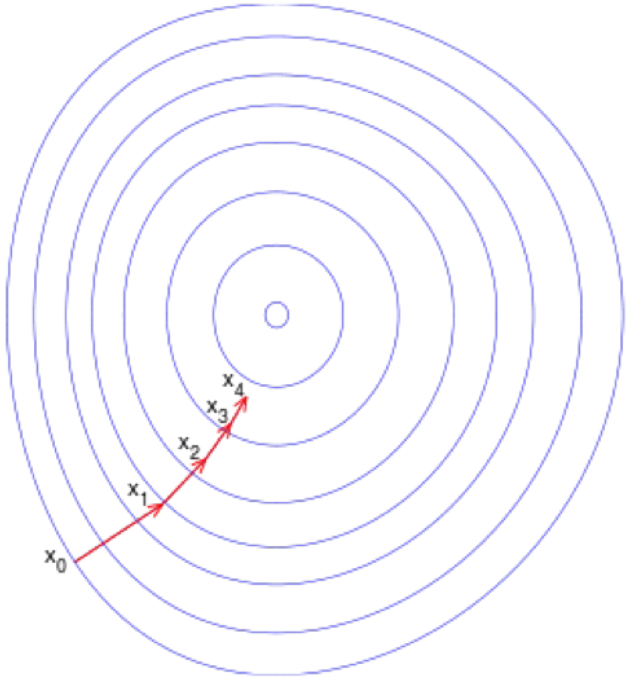
$$\Delta w_{kj} = \alpha\, y_k\, x_j$$

From ANNs to Deep Neural Nets : Supervised L.

Goal: find the set of parameters which optimize a cost function. In image labeling for instance, we want for each image as input, its label as output. The cost function is how far is our output from the desired label.

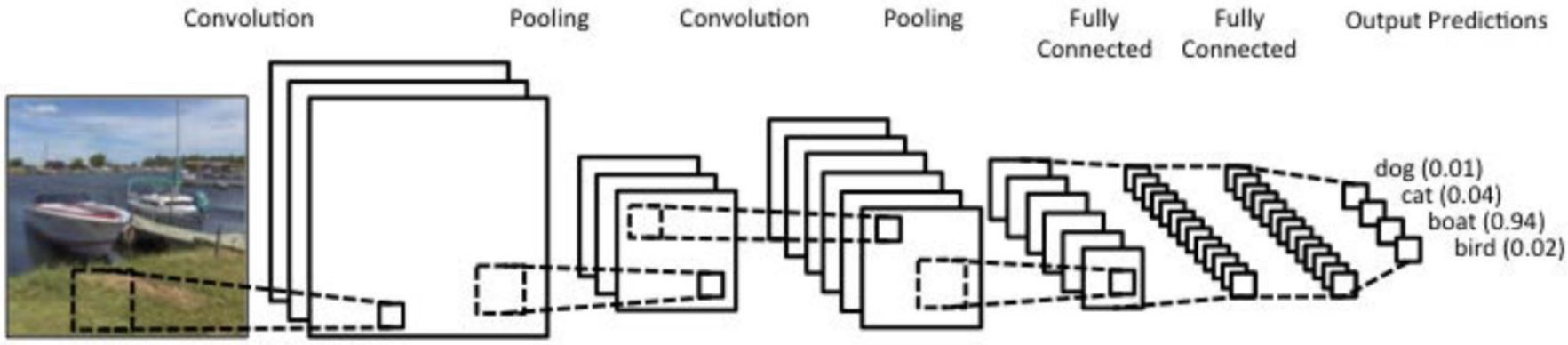
Common approach: the cost function derivatives (or slope) can be estimated for each set of parameters. The gradients can be followed to find the optimums.

Analogy: you are blind folded on a hill and you want to find the top. You can still feel the slope of the hill with your feet, how do you do ?



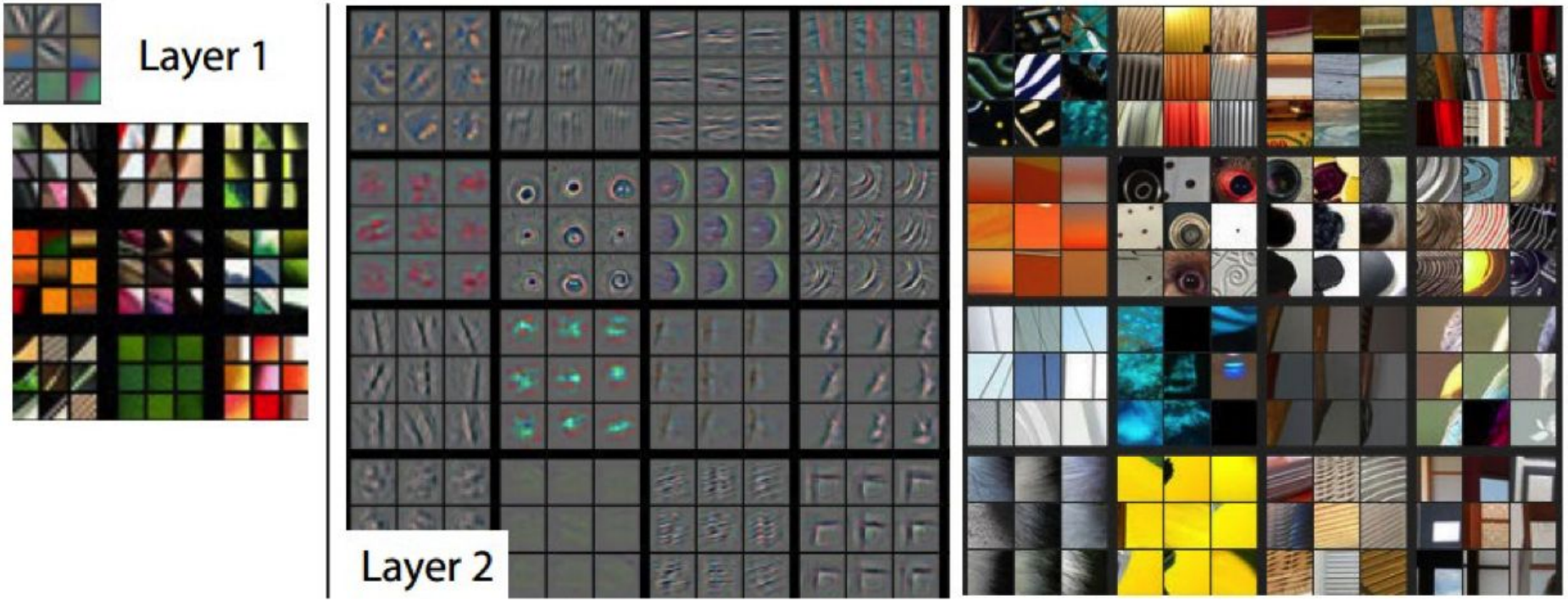
Beyond ML : From ANNs to Deep Neural Nets

Combining machine learning and deep learning with huge datasets through convolutional neural networks (CNNs) to achieve powerful results.

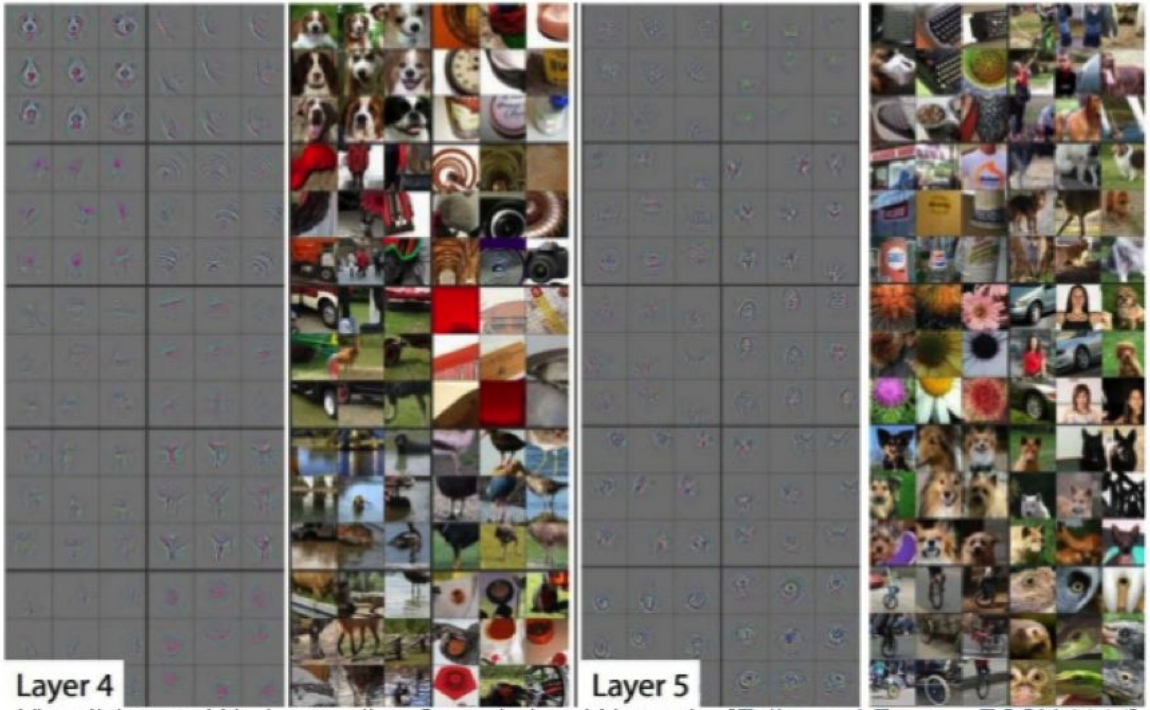


CNNs are stacked 2D convolution filters.

Beyond ML : From ANNs to Deep Neural Nets



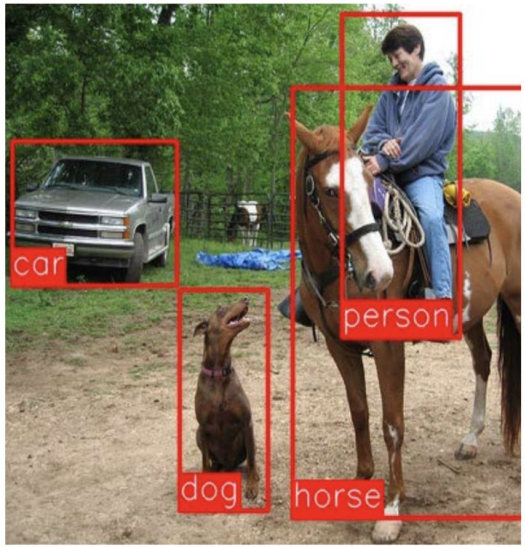
Beyond ML : From ANNs to Deep Neural Nets



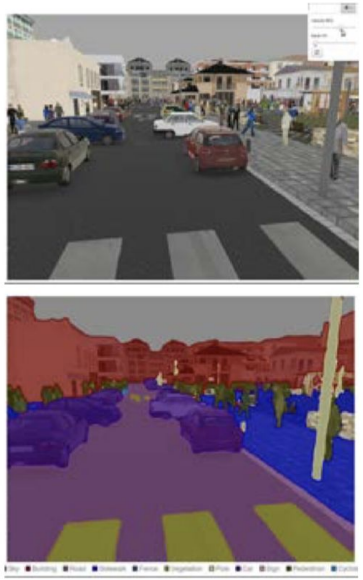
Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Beyond Machine Learning : Deep Learning

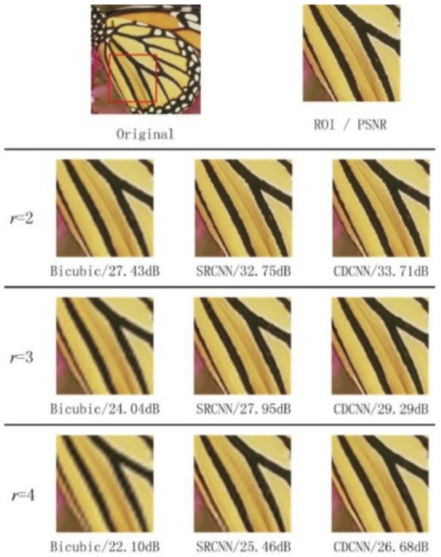
Combining machine learning and deep learning with huge datasets through convolutional neural networks (CNNs) to achieve powerful results.



Detection

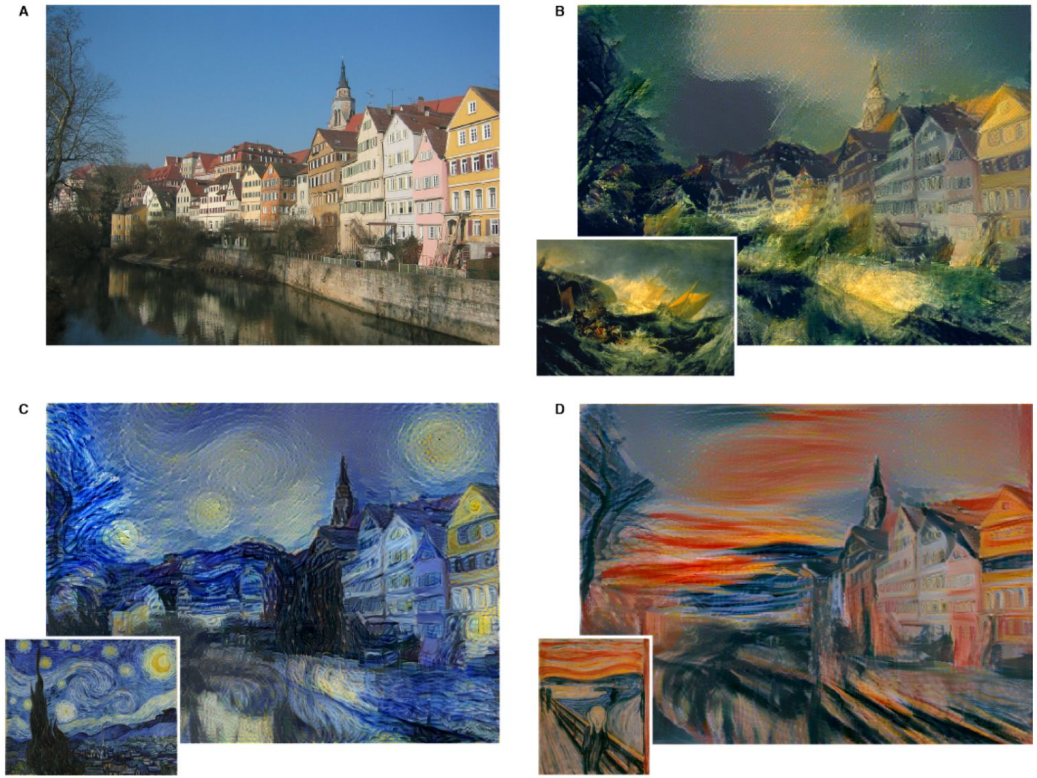


Segmentation



Denoising

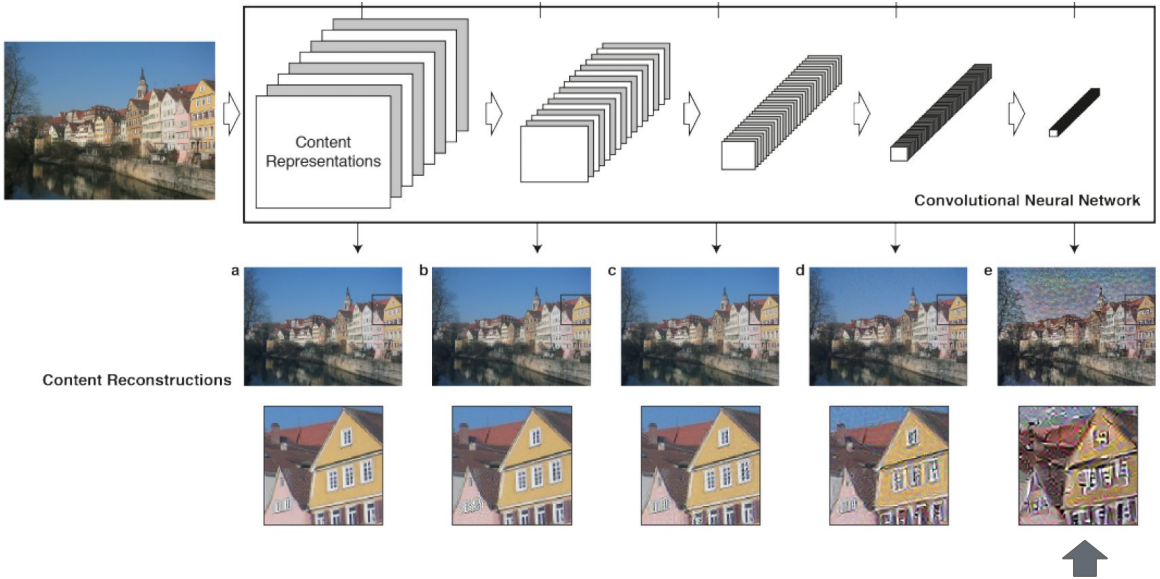
Beyond Machine Learning : Deep Learning



Understanding the structure of the neural networks one can distinguish textures from the global semantic of the image, and apply styles to images

Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

Beyond Machine Learning : Deep Learning



The convolutional NN tends to abstract more and more. Image reconstruction along the layers tend to disregard more and more the small details, keeping the general images.

Mixing this with lower layers that give the style of another image, can allow transferring styles.

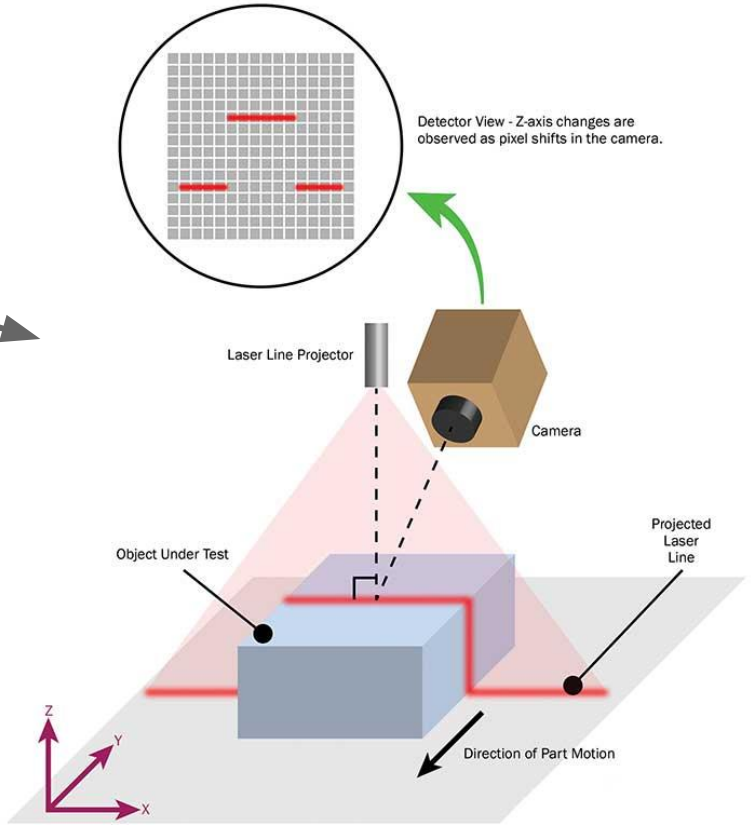
Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

Table of Contents

Depth Cameras

Addition of 3D information can be achieved :

- by triangulation
- by stereovision
- by time of flight

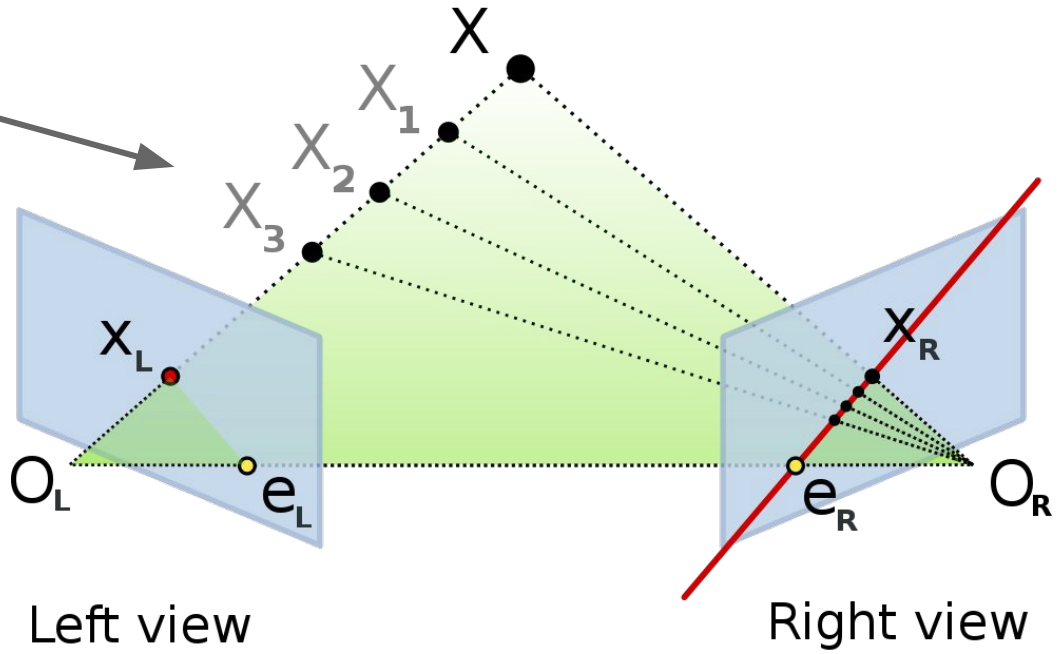


Courtesy of Coherent Inc.

Depth Cameras

Addition of 3D information can be achieved :

- by triangulation
- by stereovision
- by time of flight



Wikimedia, by Arne Nordmann

3D Reconstruction : Stereo Vision

Idealized camera geometry for stereo vision

- Disparity between two images -> computing of depth

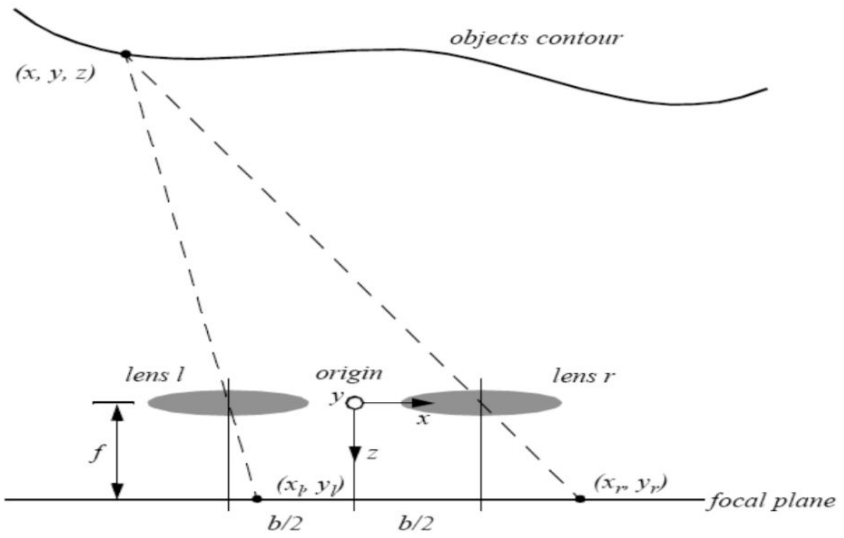
$$\frac{x_l}{f} = \frac{x + b/2}{z} \text{ and } \frac{x_r}{f} = \frac{x - b/2}{z}$$

disparity

$$\frac{x_l - x_r}{f} = \frac{b}{z}$$

$$z = b\frac{f}{x_l - x_r}$$

$$x = b\frac{(x_l + x_r)/2}{x_l - x_r} \; ; \; y = b\frac{(y_l + y_r)/2}{x_l - x_r}$$



3D Reconstruction : Stereo Vision

Distance is inversely proportional to disparity

- Closer objects can be measured more accurately (nonlinear resolution).

Disparity is proportional to the baseline b

- For a given disparity error, the accuracy of the depth estimate increases with increasing baseline b.
- However, as b increases, some objects may appear in one camera, but not in the other (occlusion).

A point visible from both cameras is a *conjugate pair*

- In the standard arrangement of cameras, conjugate pairs lie on *epipolar lines* (parallel to the x-axis for the arrangement in the figure of the previous slide).

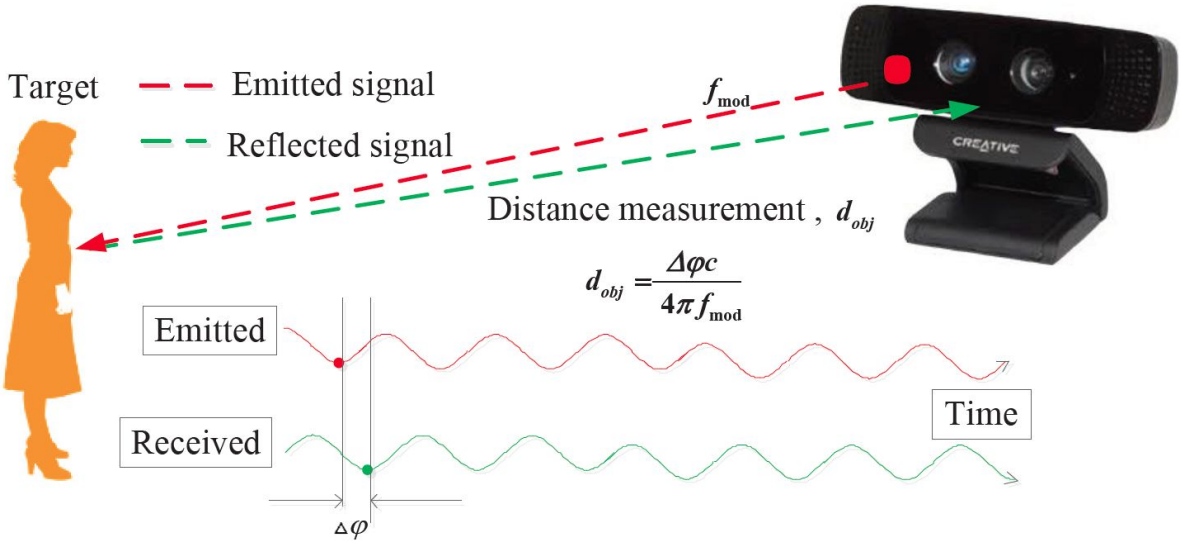
OpenCV 3.0 Depth map from Stereo

This can be extended to multiple cameras (multiple view vision). Similarly, camera poses are known. This still needs simple triangulation, with the only difference that the equations are a bit less pretty.

Depth Cameras

Addition of 3D information can be achieved :

- by triangulation
- by stereovision
- by time of flight



Islam, A., Hossain, M. A., & Jang, Y. M. (2016, July). Interference mitigation technique for time-of-flight (ToF) camera. In *2016 eighth international conference on ubiquitous and future networks (ICUFN)* (pp. 134-139). IEEE.

3D Reconstruction : SLAM

Simultaneous localization and mapping:

• Before either had : (known map and unknown camera poses) or (known camera poses and unknown map)

• Now : unknown map and unknown camera poses.

• Solution requires non-linear optimization.

MOBILE ROBOTS 4/5 – Navigation

Prof. Francesco Mondada

EPFL

2021-2022

Navigation

What is navigation? Finding a collision-free path from one pose to another

Algorithm properties:
• **Optimality:** does the planner find trajectories that are optimal in some sense (length, execution time, energy consumption)?
• **Completeness:** does the planner always find a solution when one exists?
• **Offline / online** (sensor-based: interleaving sensing, computation and action).

Questions: Do we have a model of the environment or only the direction to the goal? If a map exists, are all obstacles included? Do we need to take into account geometry, kinematics constraints, and/or the dynamics of the robot?

Note that most existing techniques make the **assumption of a mass-less, holonomic, point-like robot** => may require low-level motion control and a priori expansion of obstacles to be implemented robustly

Navigation : Local versus Global

Local : Obstacle Avoidance

• Tactical: modulating the trajectory to avoid unforeseen, local obstacles
• No map or only local and sensor-based
• Rather reactive: no complex sensor processing => fast

Necessary in mobile robotics because environments are not fully predictable (uncertainty in sensors and maps, as well as presence of dynamic obstacles)

Global :Path Planning (or motion planning)

• Strategic: planning the global trajectory
• Given a map (metric, grid-based or topological) and a goal location
• Rather cognitive: planning of a series of actions => time consuming

Compared to industrial robotics :

- less complicated (less DOF) but more frequent (discrepancies map vs. real environment)

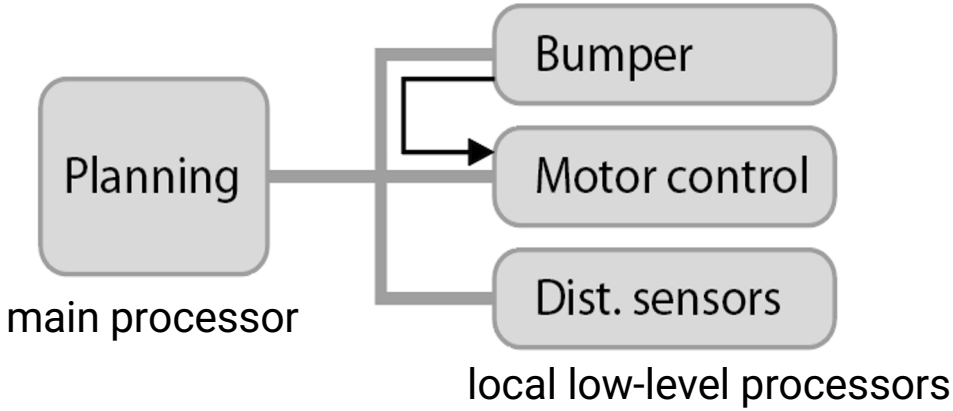
Example ASEBA architecture

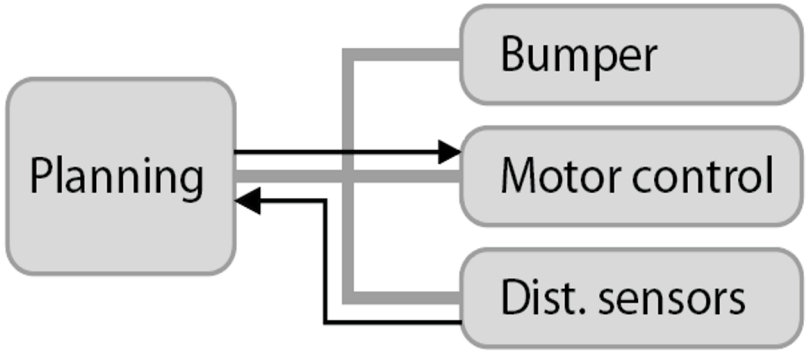
Obstacle Avoidance

- Tactical: modulating the trajectory to avoid unforeseen, local obstacles, or simply stop
- Low level

Path Planning (or motion planning)

- Strategic: planning the global trajectory
- High level processing





Obstacle Avoidance

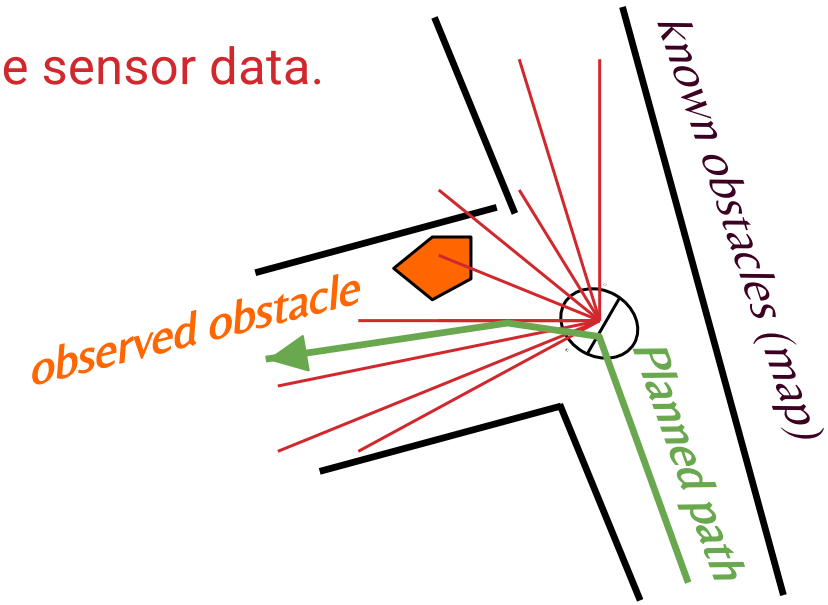
The goal of obstacle avoidance algorithms (as opposed to path planning) is to avoid collisions with unpredictable objects or due to uncertainty in the localization process or the map.

It is either based on a *local map* or directly uses the sensor data.

It is often implemented as an independent task running at high frequency.

Efficient obstacle avoidance methods should take into account:

- the sensor readings
- the kinematics / dynamics of the robot
- the overall goal direction



Obstacle Avoidance – Tactile Sensors

Mainly used for two purposes, based on observation of nature:

- **Manipulation**

  measurement of point and force of contact with objects that are manipulated, definition of stable positions or detection of dynamic situations,

- **Exploration**

  detection of close objects, as well as some physical properties like surface texture, hardness, temperature, friction coefficients, etc.

Tactile Sensors Based on Normal Pressure:

- Contact closure
  - Only binary
  - Used for security purposes
- Piezoresistive
  - Measures forces
  - Signal drifts and has hysteresis
- Piezoelectric
  - Measures forces
  - High bandwidth
  - Critical electrical junctions
- Capacitive
  - Measures forces
  - Complex circuitry

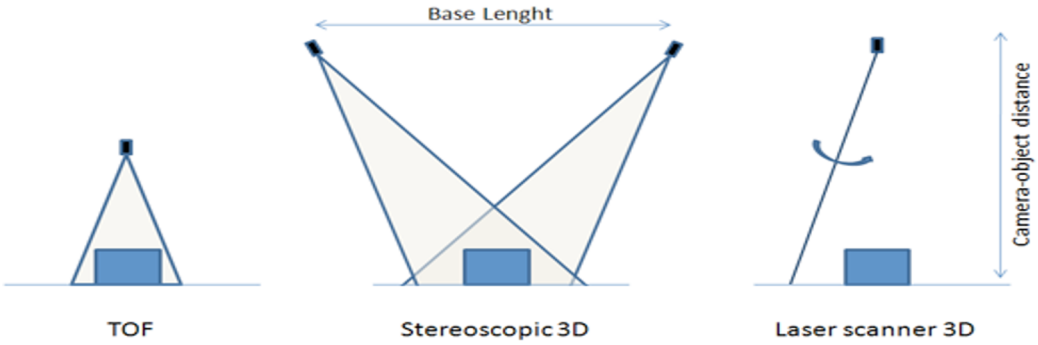
Obstacle Avoidance – Time Of Flight Sensors

- Range information:
  - key element for obstacle avoidance, localization and environment modeling
- Ultrasonic sensors and laser range finders make use of propagation speed of sound or electromagnetic waves respectively. The traveled distance of a sound or electromagnetic wave is given by

$$l = c \cdot t$$

Where

- l = distance traveled (usually round-trip)
- c = speed of wave propagation
- t = time of flight



Obstacle Avoidance – Time Of Flight Sensors

Important to keep in mind:

- Propagation speed of sound: 0.3m/**ms**
- Propagation speed of of electromagnetic signals: 0.3m/**ns**
- 3 meters correspond to 10ms for an ultrasonic system versus only 10ns for a laser range sensor

  => time of flight with electromagnetic signals involves very fast electronics

  => laser range sensors are more expensive and delicate to design

The quality of TOF range sensors mainly depends on:

- Uncertainties about the exact time of arrival of the reflected signal
- Inaccuracies in the time of fight measure (laser range sensors)
- Opening angle of transmitted beam (especially ultrasonic range sensors)
- Interaction with the target (surface, diffuse/specular reflections)
- Variation of propagation speed (sound)
- Speed of mobile robot and target (if not at stand still)

TOF Sensors – Ultrasonic Sensors

Transmit a packet of (ultrasonic) pressure waves

Distance *d* of the echoing object can be calculated based on the propagation speed of sound *c* and the time of flight *t*.

$$d = \frac{c \cdot t}{2}$$

The speed of sound *c* in air is given by

$$c = \sqrt{\gamma \cdot R \cdot T}$$

where

γ : ratio of specific heats or adiabatic index (1.402 for air)

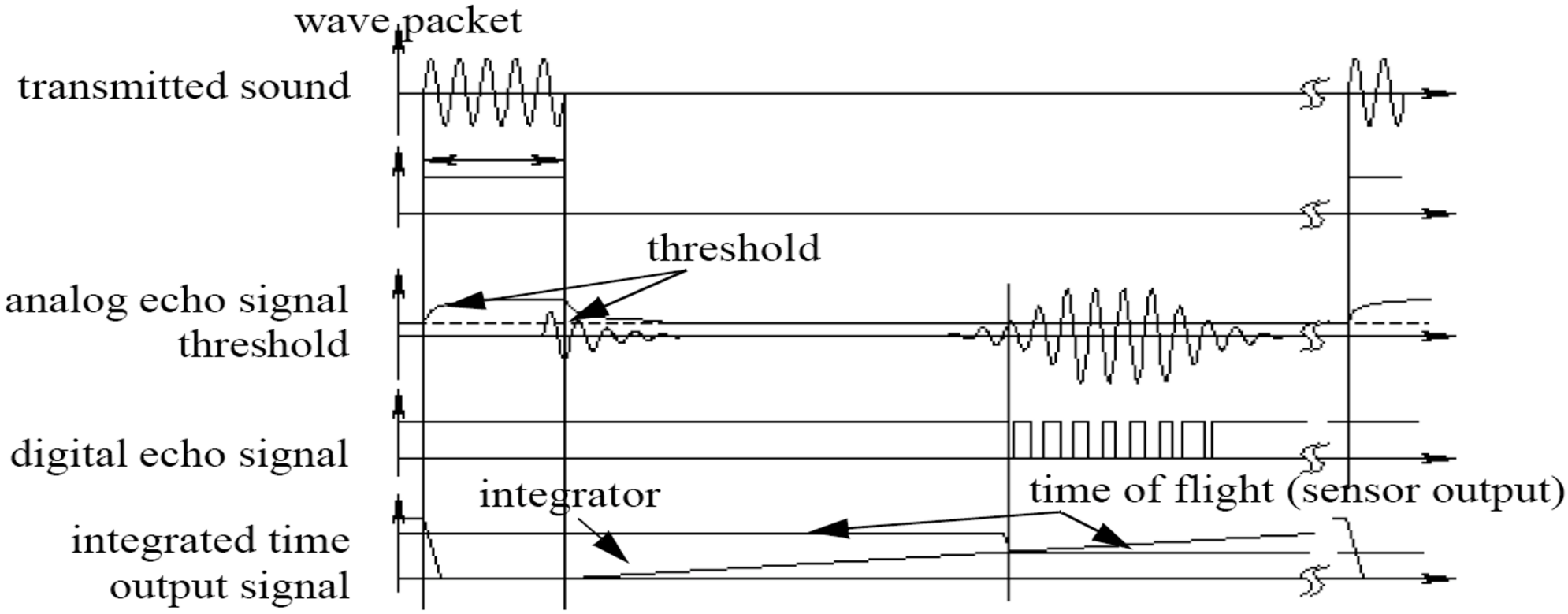
R: gas constant (287.05 $J \cdot kg^{-1} \cdot K^{-1}$ for air)

T: temperature in Kelvin

In air at standard pressure and 20° Celsius the speed of sound is around c = 343 m/s.

TOF Sensors – Ultrasonic Sensors

Possible implementation (very basic scheme)



TOF Sensors – Ultrasonic Sensors

Typical frequency: 40 - 180kHz

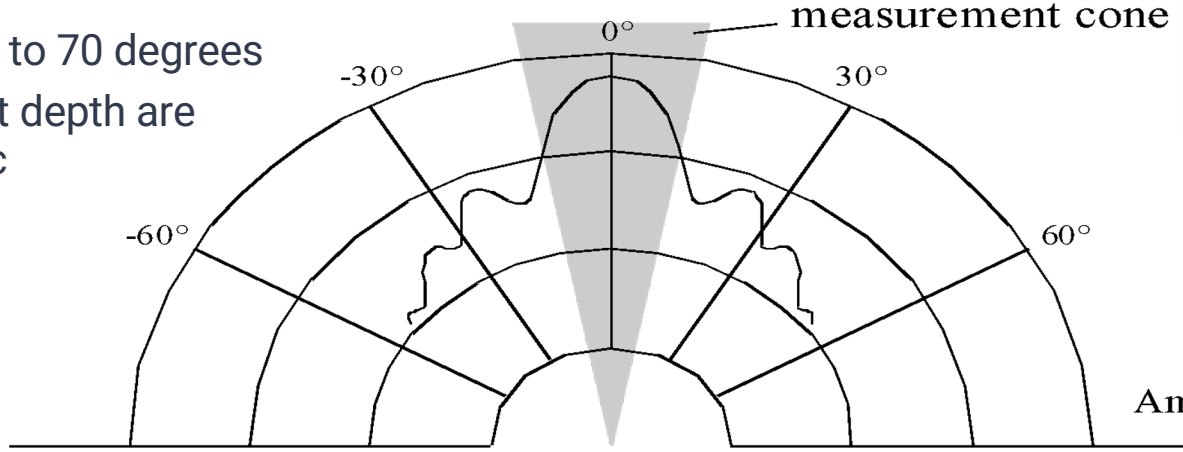
- typical range: 12 cm to 5 m; resolution: ~2 cm

Generation of sound wave: piezo transducer

- transmitter and receiver can be separated or not

Sound beam propagates in a cone (*approx.*)

- opening angles: 20 to 70 degrees
- regions of constant depth are segments of an arc (sphere for 3D)

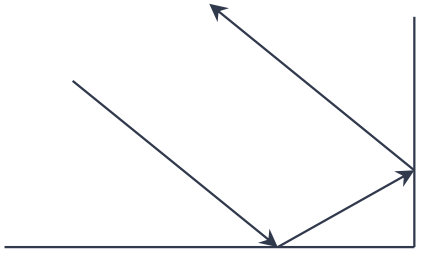


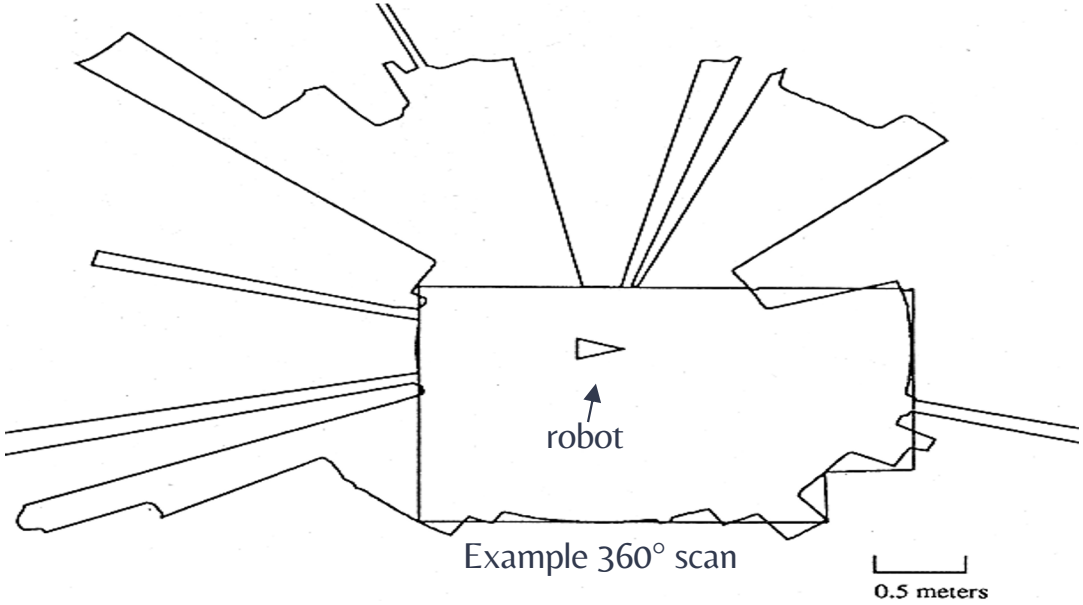
Typical intensity distribution of a ultrasonic sensor

TOF Sensors – Ultrasonic Sensors

Other limitations of ultrasonic sensors

- soft surfaces that absorb most of the sound energy
- surfaces that are far from being perpendicular to the direction of the sound
  => specular reflection
- blanking time errors
- Multipath errors
  - E.g. in corners



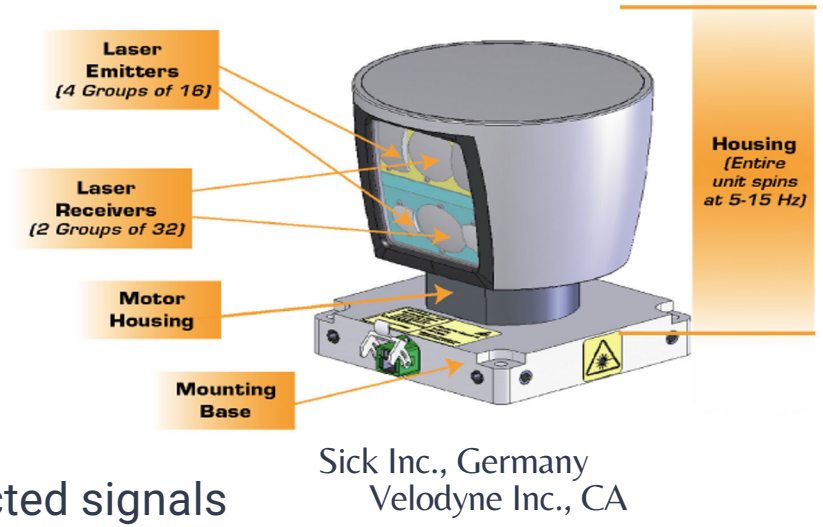


TOF Sensors – Laser RangeFinders

Time of flight measurement is generally achieved using one of the following methods:

- Pulsed laser (e.g. Sick Laser rangefinder)
  - direct measurement of time of flight
  - requires resolving picoseconds (3m = 10ns)

- Phase shift measurement (e.g. Hokuyo laser rangefinder)
  - sensor transmits 100% amplitude modulated light at a known frequency and measures the phase shift between the transmitted and reflected signals
  - technically easier than the above method because only a light intensity needs to be measured



Sick Inc., Germany
Velodyne Inc., CA

Because laser measure a point, laser are used on scanners

TOF Sensors – Laser RangeFinders : Phase Shift

Also known as *laser radar* or *LIDAR (LIght Detection And Ranging)*

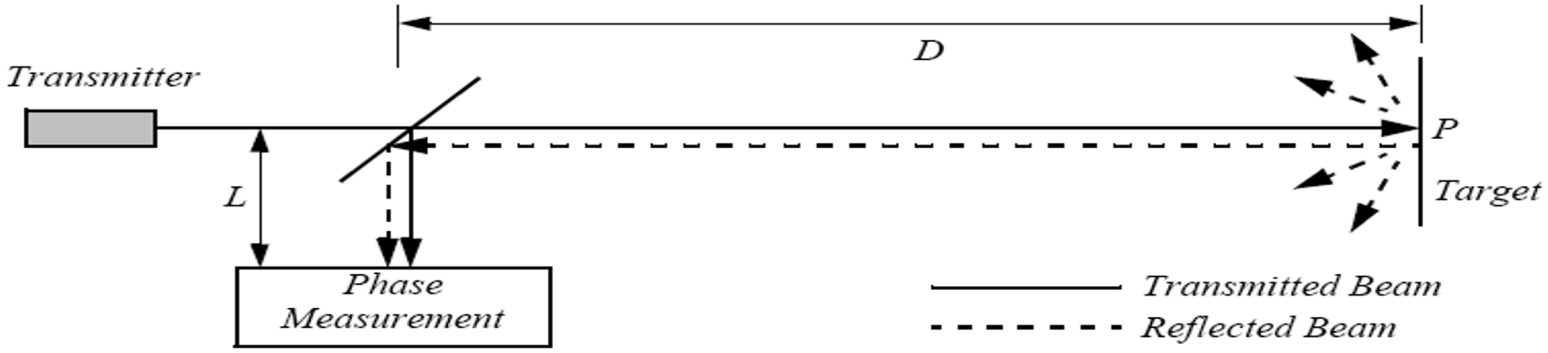
Transmitted and received beams are coaxial

Transmitter illuminates a target with a collimated beam (laser)

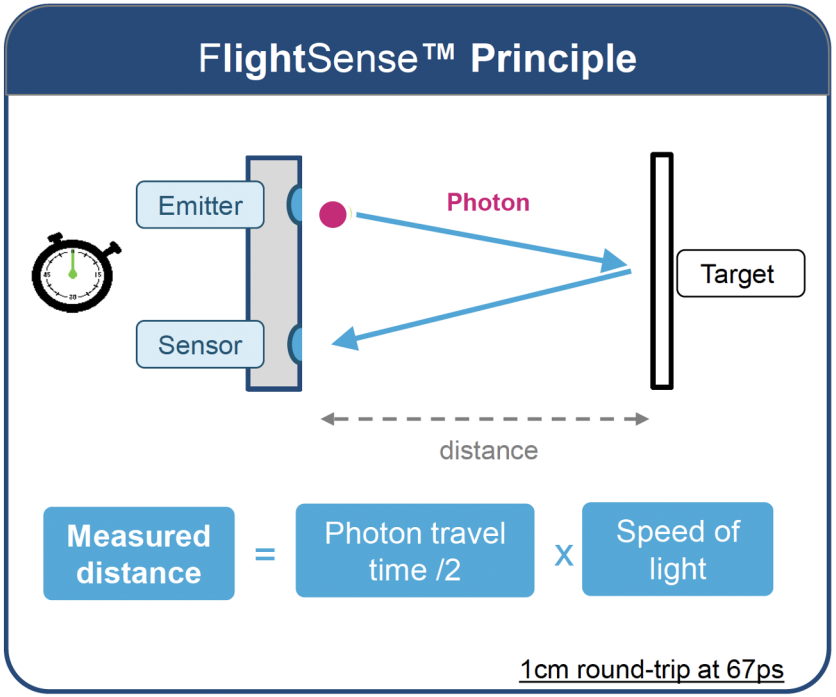
Diffuse reflection with most surfaces

Receiver detects the time needed for round-trip

An optional mechanism sweeps the light beam to cover the required scene (in 2D or 3D).

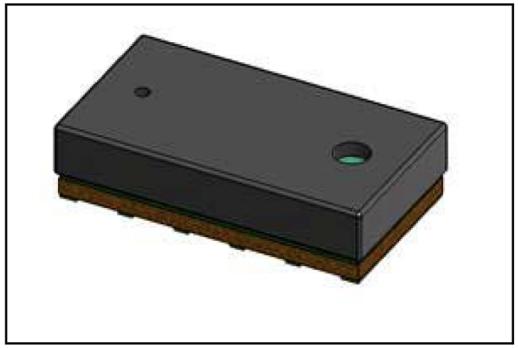


TOF Sensors – Laser RangeFinders : TOF



life.augmented

VL53L0X

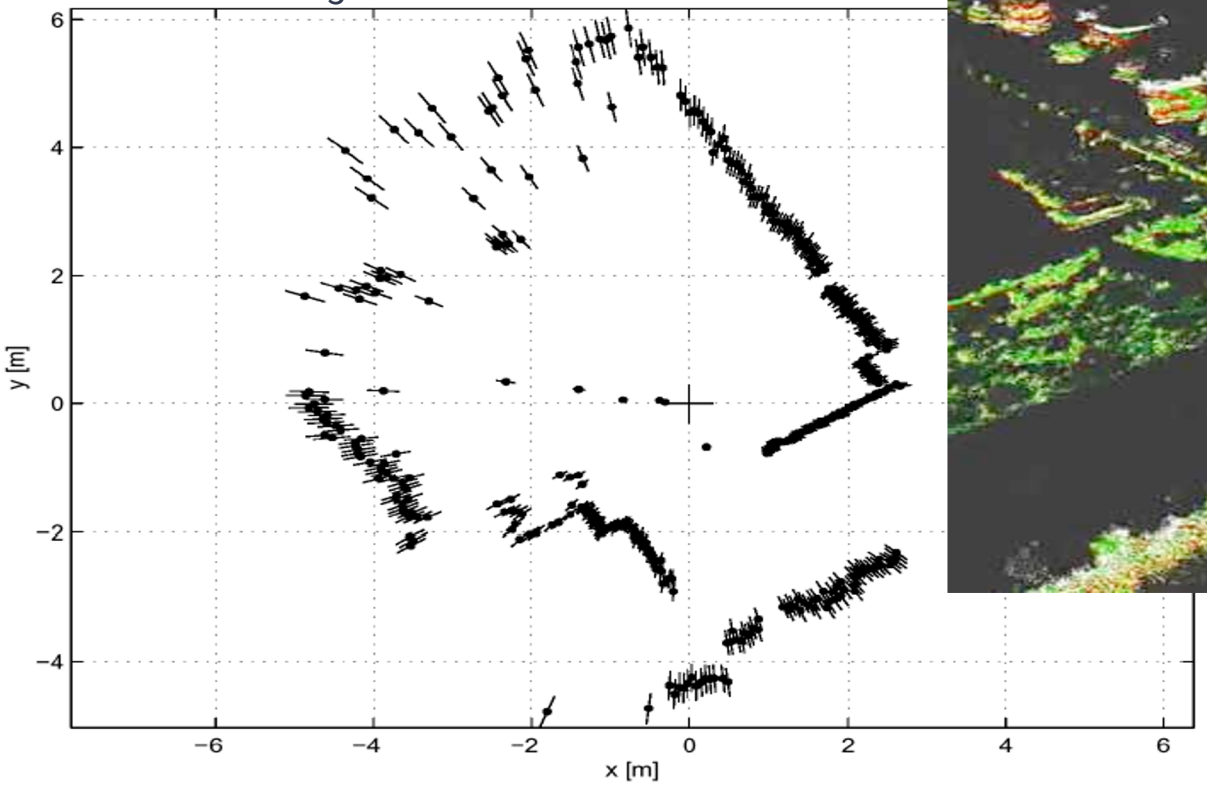


Features

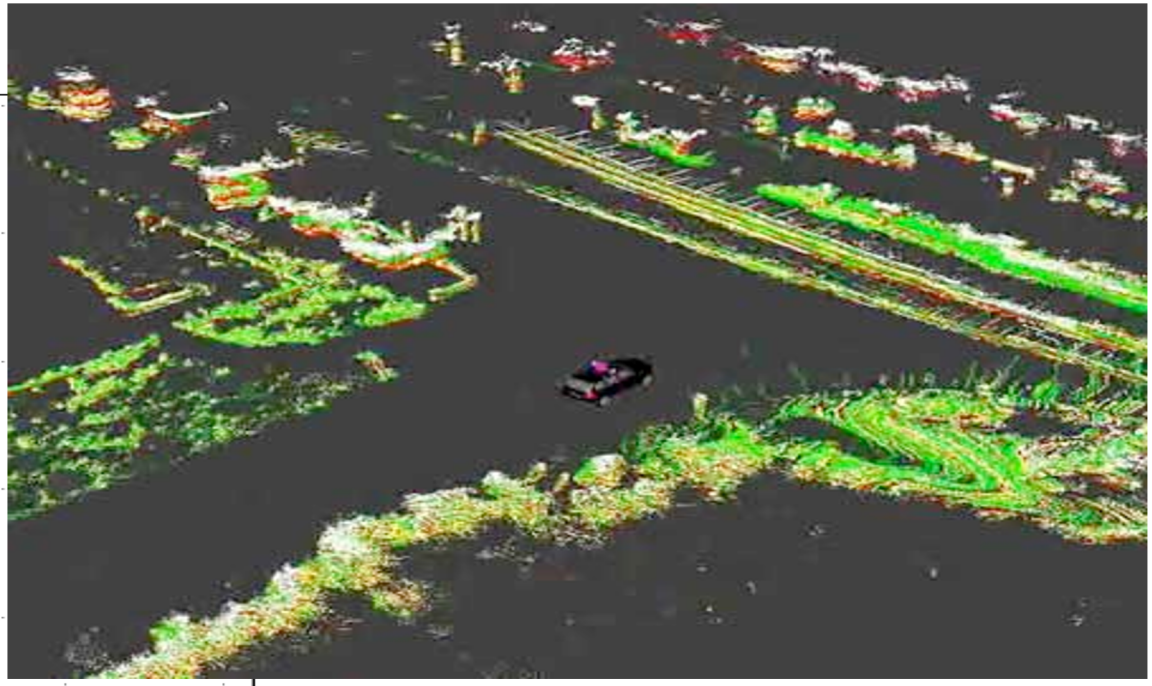
- Fully integrated miniature module
  - 940nm Laser VCSEL
  - VCSEL driver
  - Ranging sensor with advanced embedded micro controller
  - 4.4 x 2.4 x 1.0mm

TOF Sensors – Laser RangeFinders Outputs

Sick laser range finder



Velodyne LIDAR

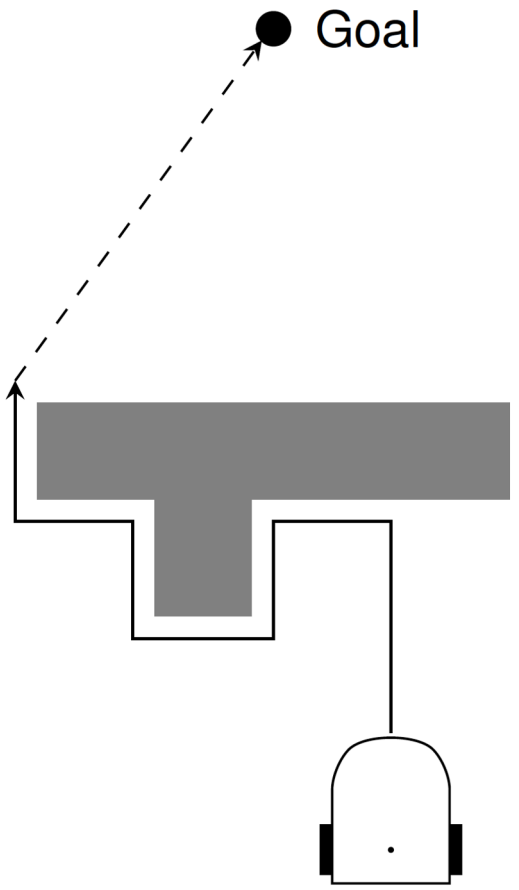


Obstacle Avoidance Strategies – Example 1

Assuming imprecise localization (distance and bearing to **visible** goal) and only proximity sensing.

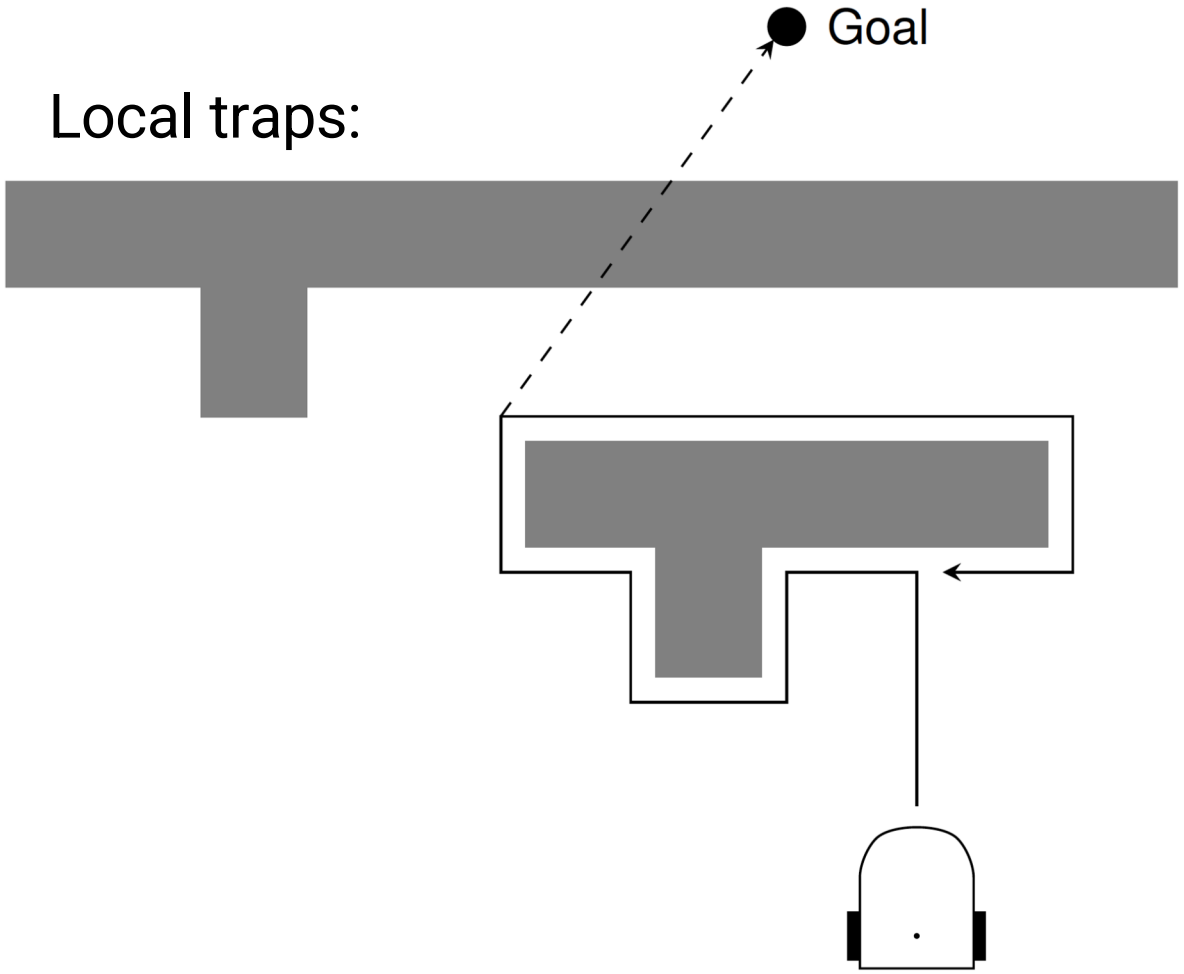
Strategy:

- Following the obstacle to avoid collision with it.
- Direction to the goal as soon as is visible again.



Obstacle Avoidance – Example 1 Problems

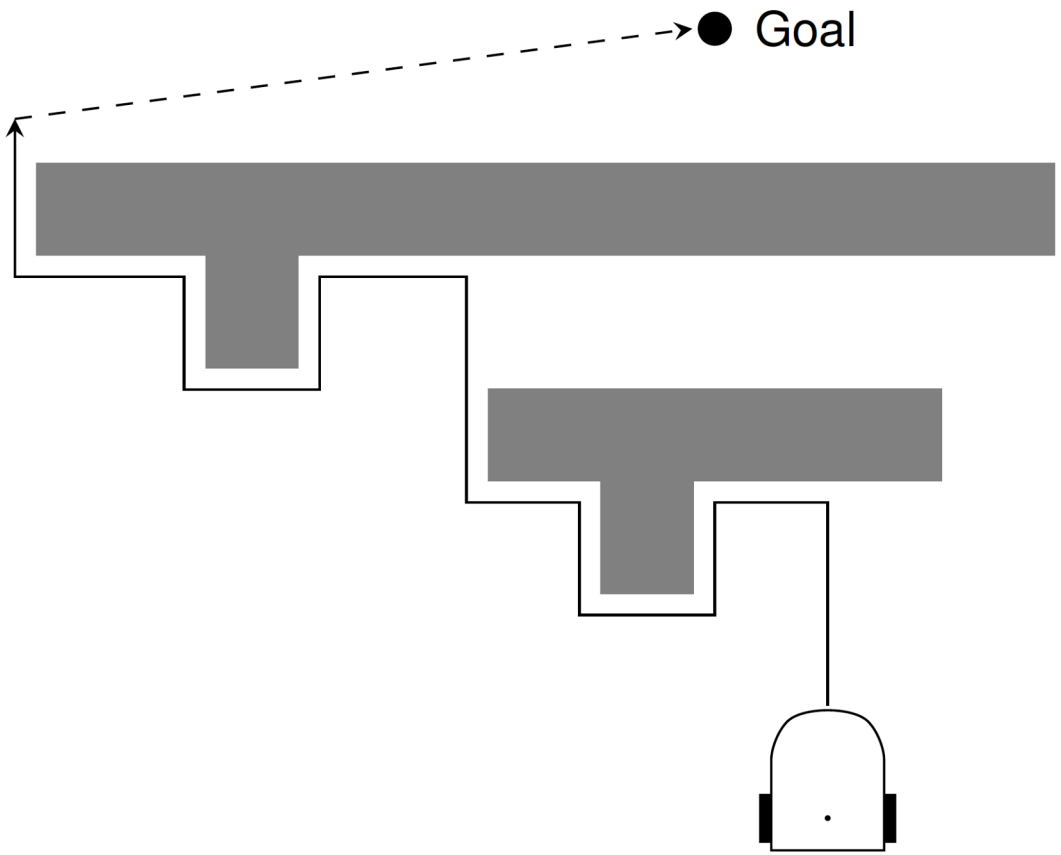
Local traps:



Obstacle Avoidance Strategies - Example 2

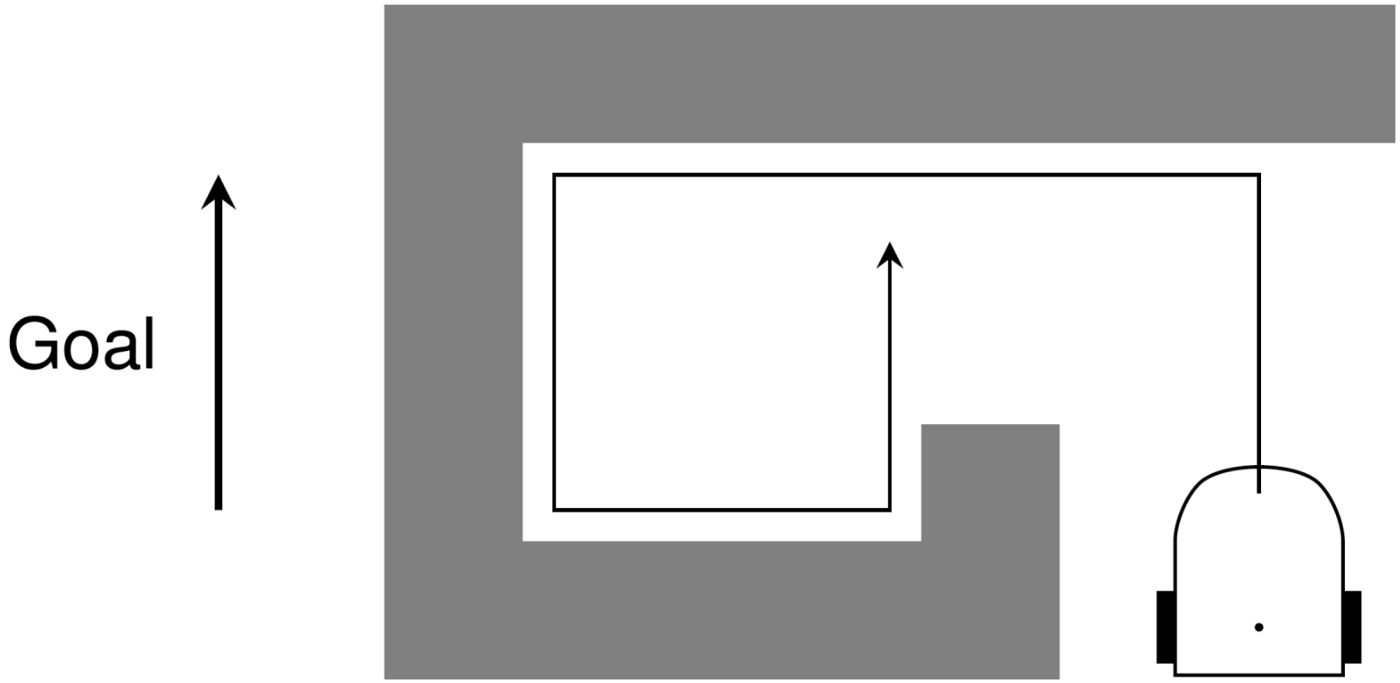
Assuming imprecise localization (distance and **continuous bearing** to goal) and only proximity sensing.

- Following the obstacle to avoid collision with it.
- Direction to the goal as soon as is **free** again. Direction can be updated by odometry. Direction is good when is a multiple of 360 degrees



Obstacle Avoidance - Example 2 Problems

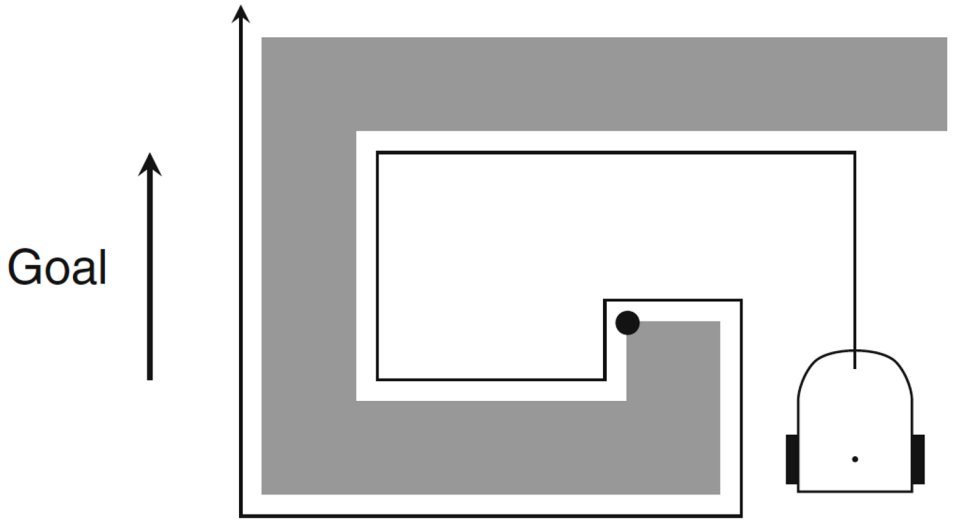
Local trap:



Obstacle Avoidance – The pledge algorithm

Assuming imprecise localization (distance and **continuous bearing** to goal) and only proximity sensing.

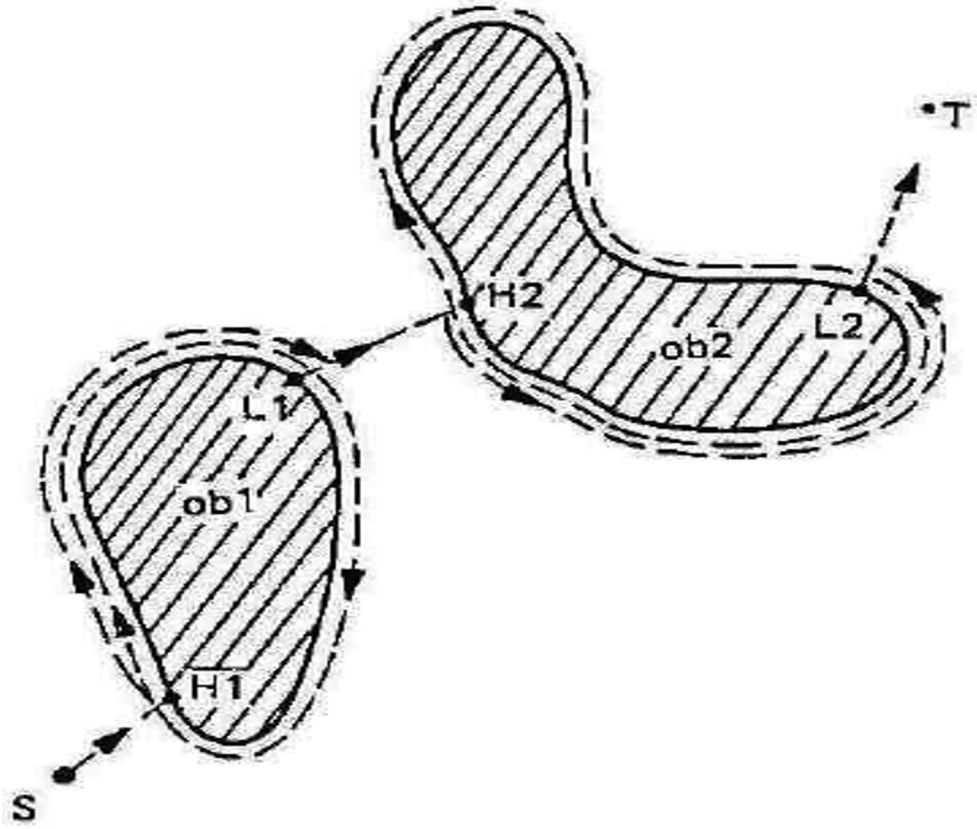
- Following the obstacle to avoid collision with it.
- Direction to the goal as soon as is **free** again. Direction can be updated by odometry. Direction is good **when is again = 0**



Obstacle Avoidance Strategies - Example 3

Assuming imprecise localization (distance and bearing to goal) and only proximity sensing.

- Following the obstacle to avoid collision with it.
- Each encountered obstacle is first fully circled before it is left at the point closest to the goal.



Obstacle Avoidance Strategies – Example 4

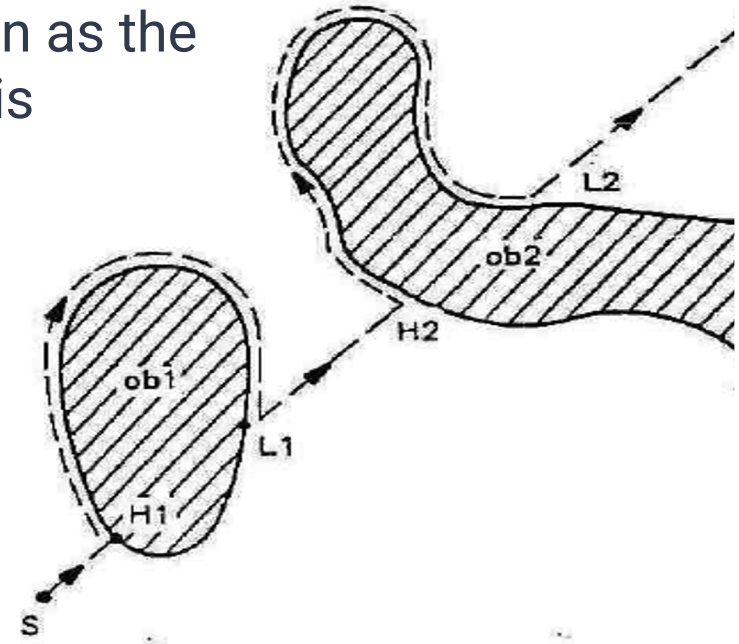
Following the obstacle always on the left (or right) side.

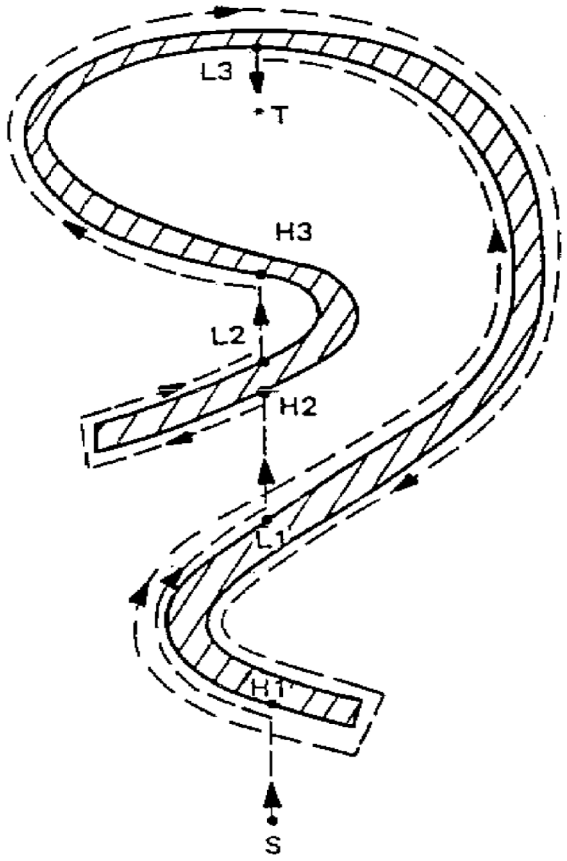
Leaving the obstacle as soon as the line between start and goal is crossed.

+ More efficient than previous.

- Requires knowing the line!

- Inefficient situations still exist.





Obstacle Avoid. – Local Mapless Potential Field

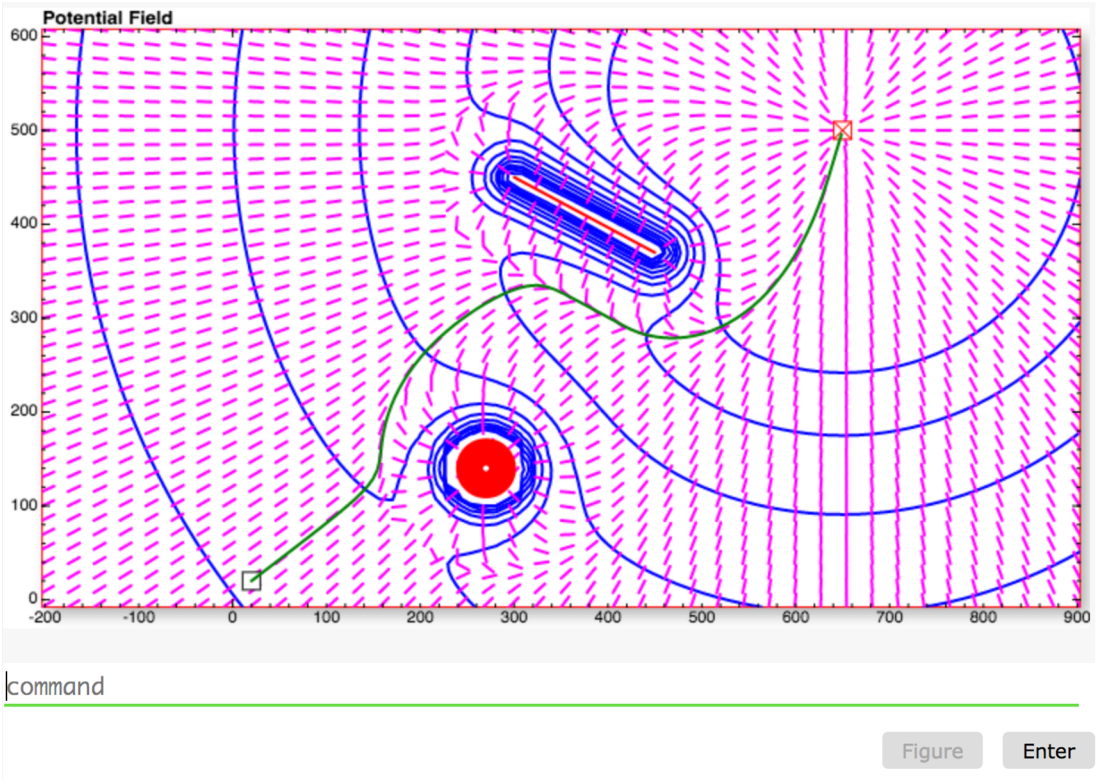
Treats range readings as a repulsive force vector, which will act on the robot (transformation from force to wheel speeds depends on kinematics).

Example of processing:

1. If the magnitude of the sum of the repulsive forces exceeds a certain threshold, the robot stops, turns into the direction of the resultant force vector, and moves on.
2. As soon as an object is detected, the repulsive force is influencing the direction of the robot

Problem of combination between attraction to a target and repulsion from the obstacles (see global navigation).

Demonstration on Potential Field



Local Occupancy Grid

Each cell represents the confidence of the algorithm in the existence of an obstacle at that location, for example:

a) For each range reading, the cell that lies on the acoustic axis and corresponds to the measured distance *d* is incremented, increasing the certainty value of the cell.

b) A histogramic pseudo-probability distribution is obtained by continuous and rapid sampling of the sensors while the robot is moving.

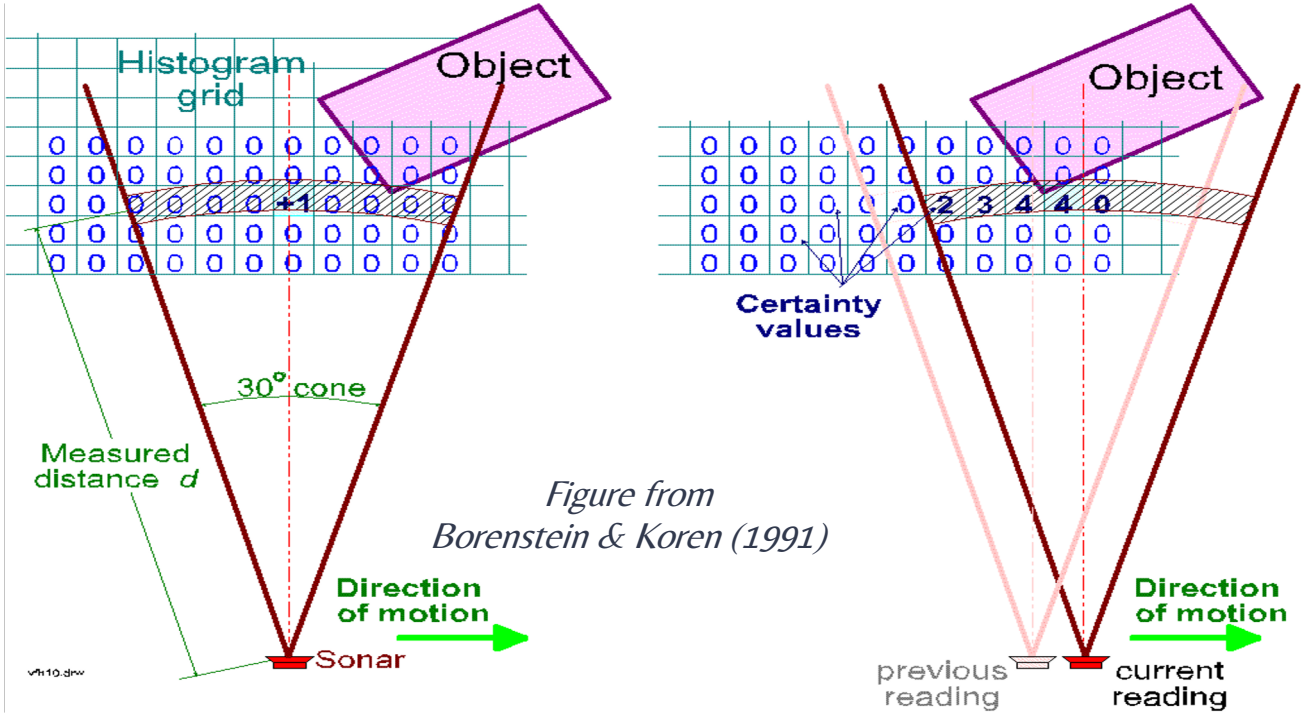


Figure from Borenstein & Koren (1991)

Curvature Velocity Method (CVM)

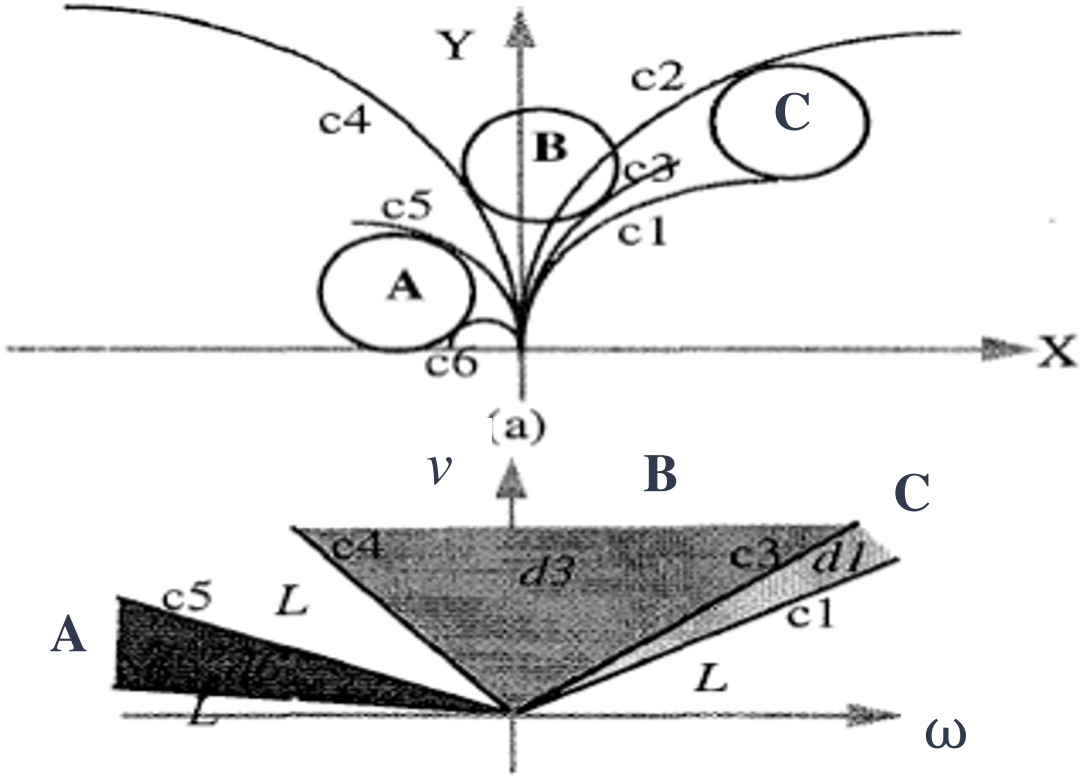
Adds *physical constraints* from the robot and the environment on the *velocity space* (*v*,ω) of the robot.

Assuming that robot is traveling on arcs (c=ω/*v*), which is a good approximation for many wheeled robots.

Obstacles are transformed from a local occupancy grid into the velocity space.

The robot chooses velocity commands that satisfy all the constraints and maximize an objective function that trades off speed, safety and goal directedness.

+ Solution corresponds directly to the commands sent to the robot.



NEW: Evaluation (with project)

In 2 parts

- 60% Project on one of the topics seen during the semester + the exercise sessions (where you want to make it??? -> survey)
- 40% Written take-home exam on moodle during the winter examination session
  - Related to the case studies (multiple choice answer + explanation)

Path Planning : What do you need?

Global Map of the environment to know which positions are accessible or not

Start and End Position to search for the optimal path within the map

Path Planning Algorithm to find the optimal path within the map

Path Following Module to follow the path from the start to the end goal. This requires knowing the position of the robot in order to adjust the motion of the robot. The controller determines the “quality” of the line following

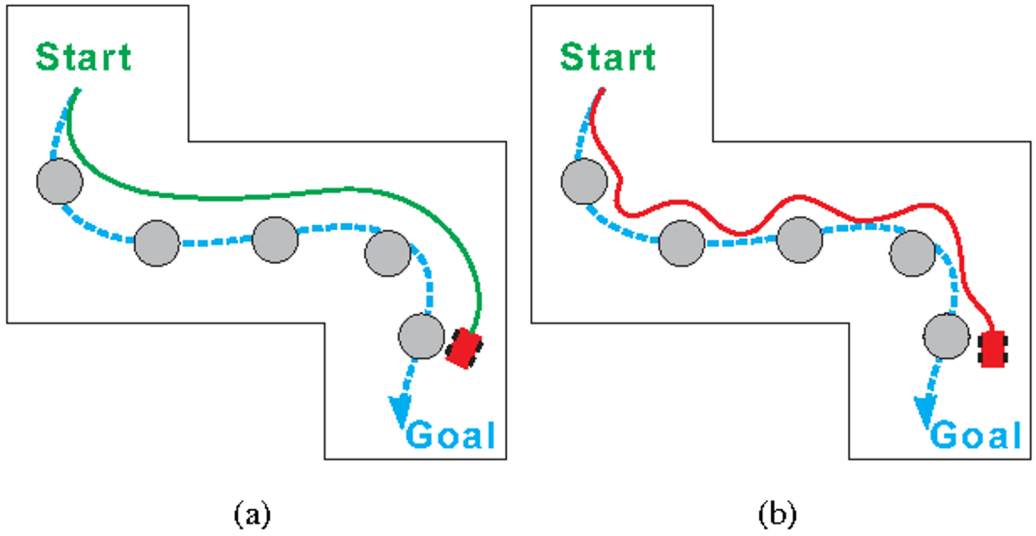


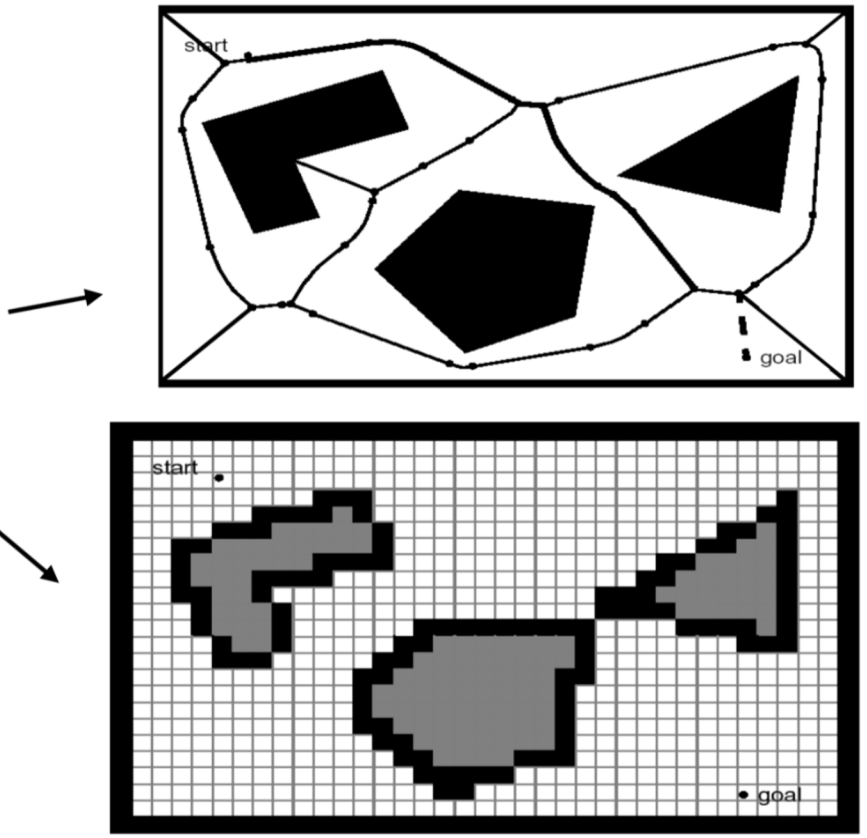
Fig. 1. Dashed blue spline is global path: a) Green spline is ideal local path; b) Red spline is actual local path

https://www.semanticscholar.org/paper/Generalized-Path-Corridor-based-Local-Path-Planning-Wang-Wieghardt/e97fea528a77be550e1262fbffa168e066198a44/figure/0

Different Approaches to Path Planning

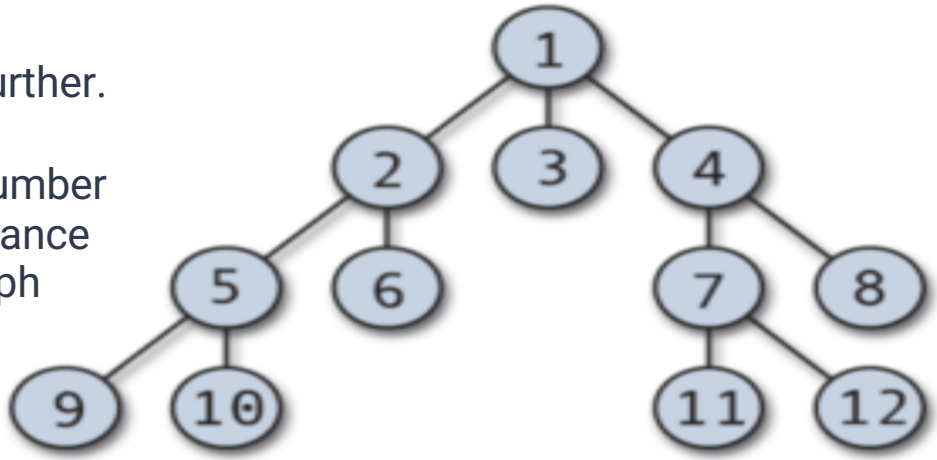
Capture the connectivity of free space into a graph that is subsequently searched for paths:

- *Road-map*: identify a set of routes within the free space
- *Cell decomposition*: discriminate between free and occupied cells -> connectivity graph



Graph Search Strategies

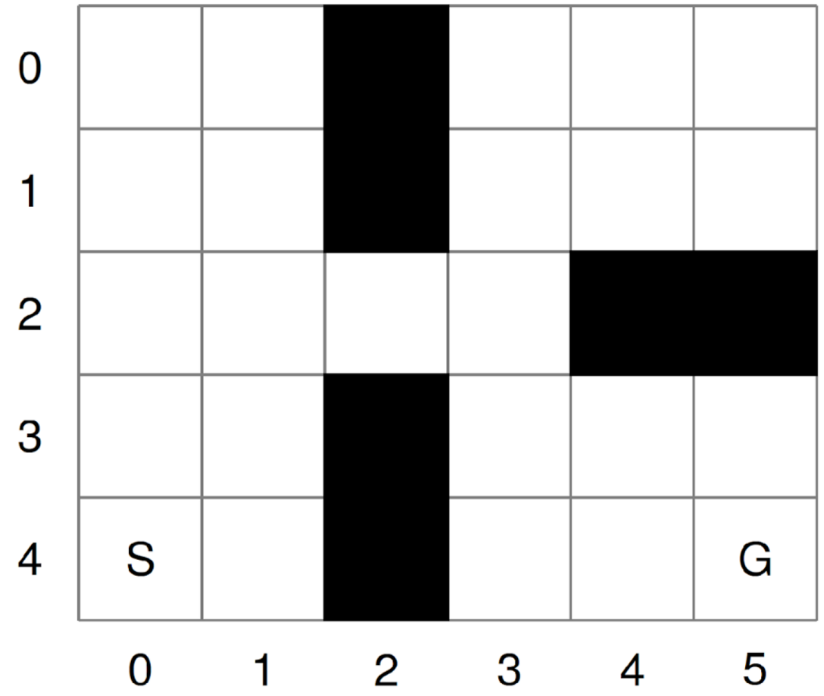
Breadth-first search
• Explore all the nearest nodes before going one step further.
• Limit the risk of exploring deep dead-ends.
• The search always returns the path with the fewest number of edges between the start and the goal (minimum distance path assuming a constant cost of each edge in the graph



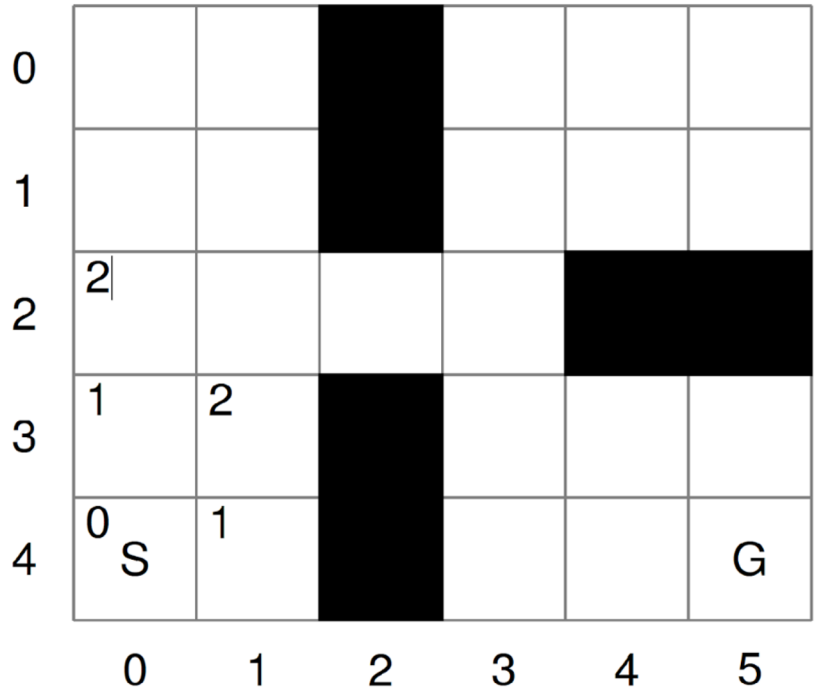
Depth-first search
• Explores as far as possible along each branch before backtracking.
• Memory efficient as completely explored branches may be deleted.

Best-first search (e.g. A*)
• Optimizes search by expanding the most promising node chosen according to some rule.
• Typically used for route finding: the straight-line distance, which is usually an approximation of road distance

Approach 1 Example – Dijkstra's Algorithm

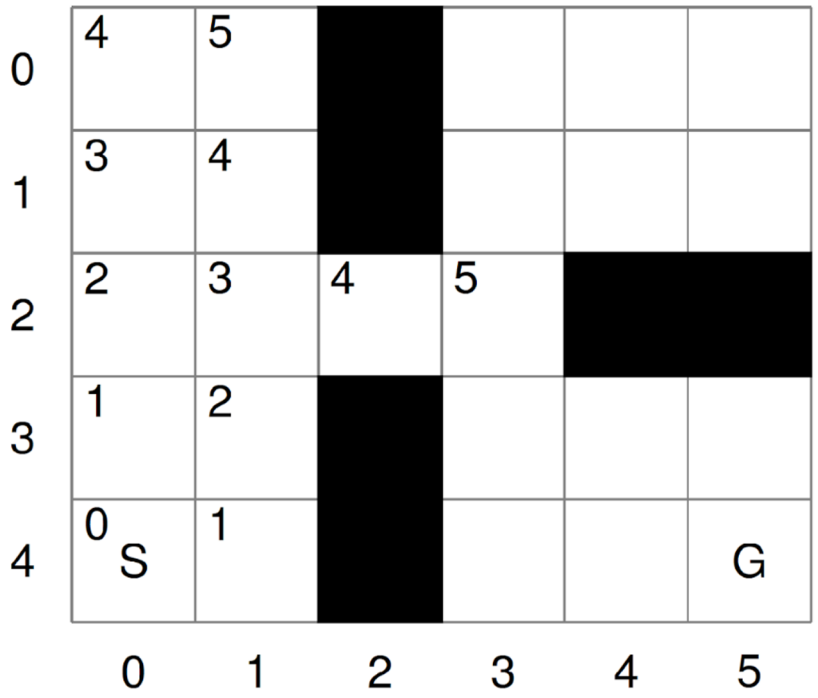


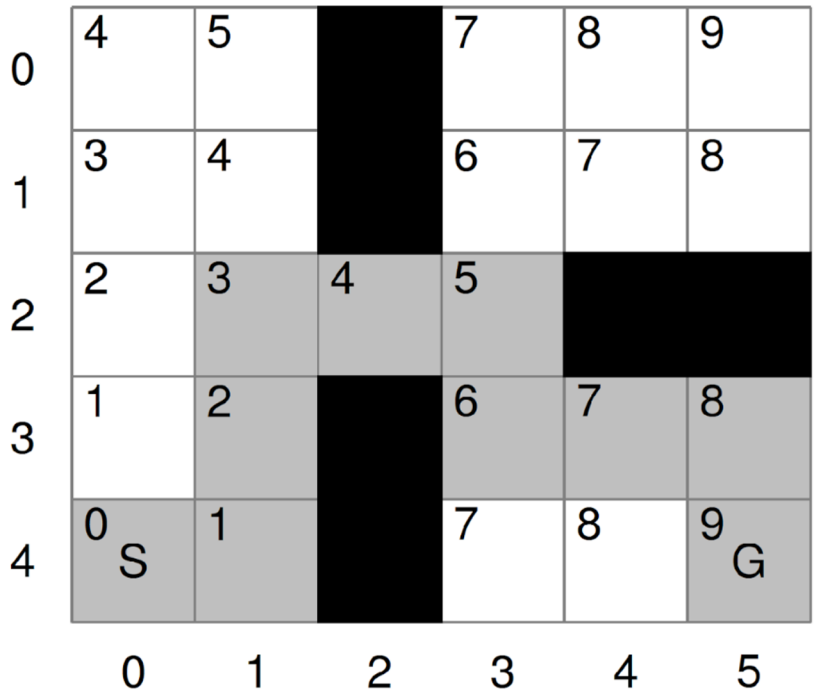
Breadth-first search



Manhattan distance

Approach 1 Example- Dijkstra's Algorithm





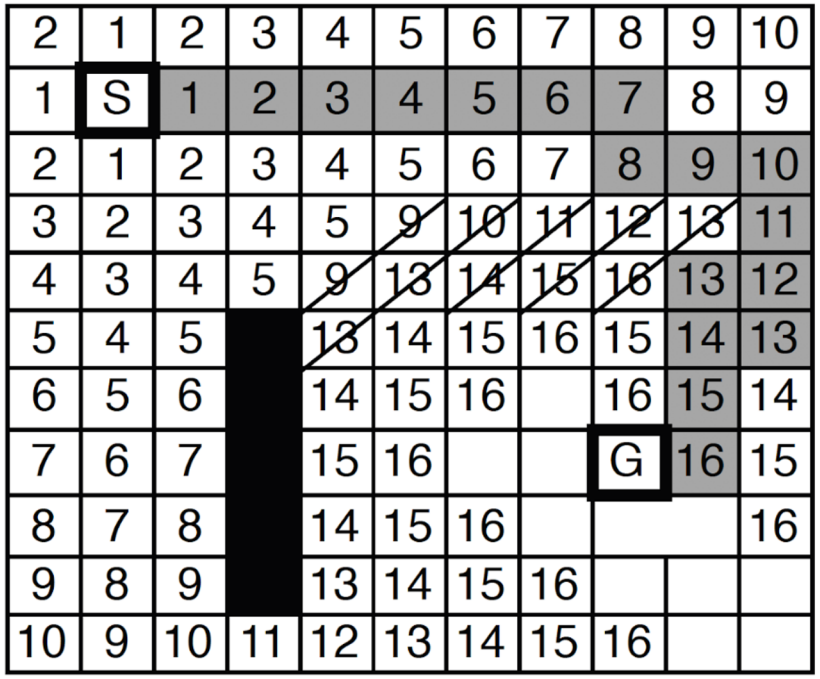
Approach 1 Example- Dijkstra's Algorithm

Algorithm 10.1: Dijkstra's algorithm on a grid map

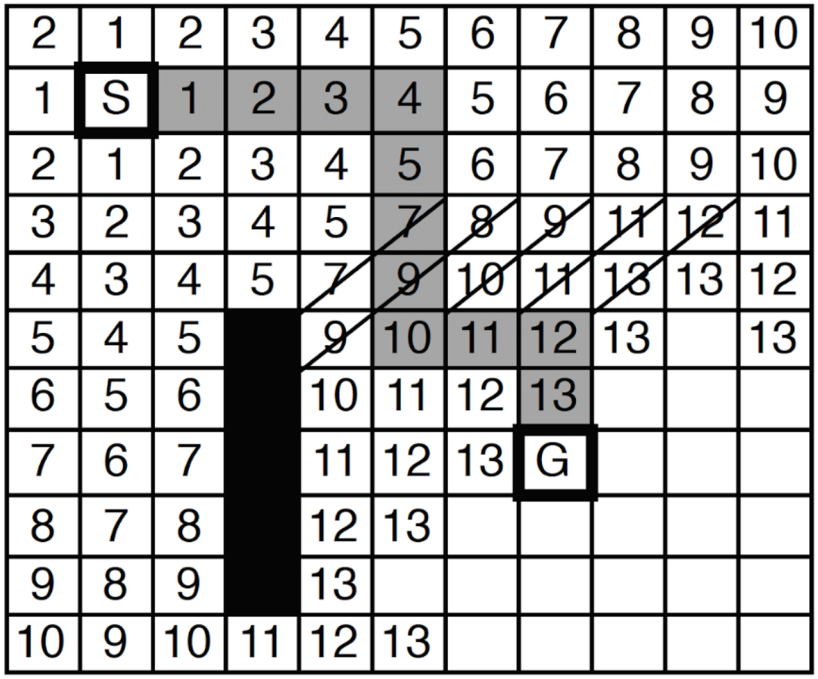
```
integer n ← 0                          // Distance from start
cell array grid ← all unmarked         // Grid map
cell list path ← empty                 // Shortest path
cell current                           // Current cell in path
cell c                                 // Index over cells
cell S ← ···                           // Source cell
cell G ← ···                           // Goal cell
```

```
 1: mark S with n
 2: while G is unmarked
 3:     n ← n + 1
 4:     for each unmarked cell c in grid
 5:             next to a marked cell
 6:         mark c with n
 7: current ← G
 8: append current to path
 9: while S not in path
10:     append lowest marked neighbor c
11:         of current to path
12:     current ← c
```

Approach 1 Example – Dijkstra With Variable Cost



Cost = 4 in central cells



Cost = 2 in central cells

Approach 1 Example – A* : Extending Dijkstra

Best-first search (e.g. A*)

- Optimizes search by expanding the most promising node chosen according to a combination of rules:

  lowest f(n) = g(n)+h(n)

  g(n) : motion cost (like in Dijkstra's algorithm)

  h(n) : heuristic function (e.g. distance to goal)

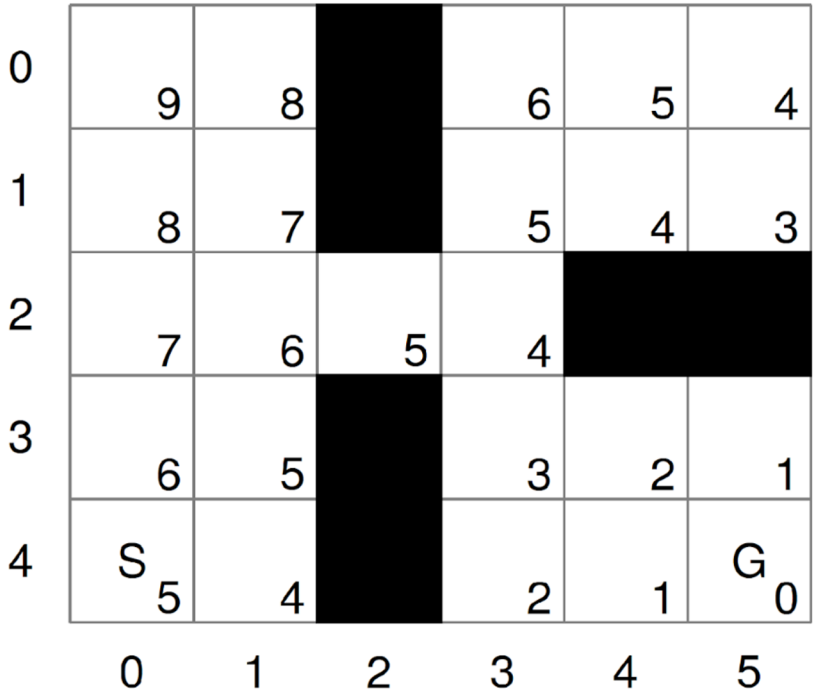
- Typically used for route finding: the straight-line distance, which is usually an approximation of road distance.

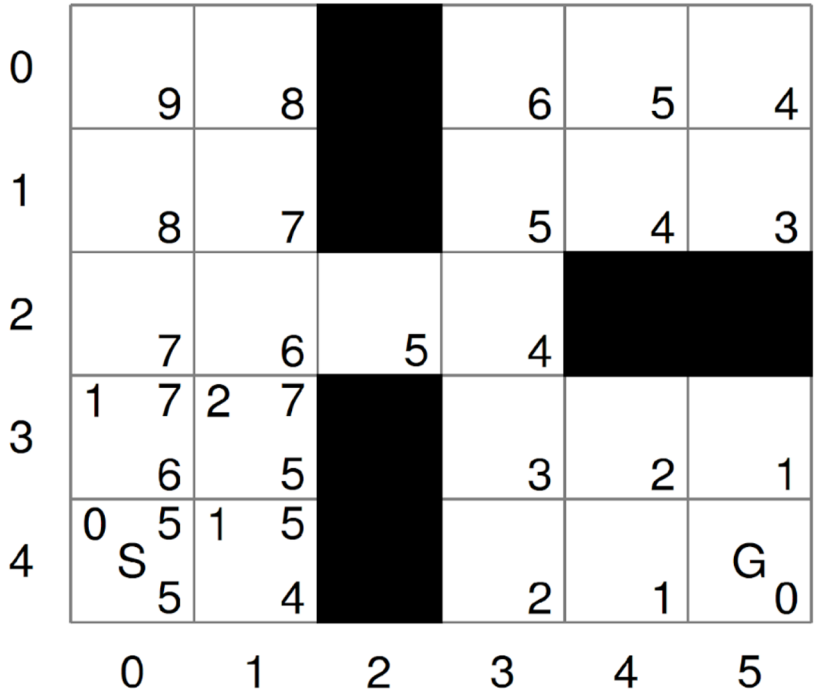
Approach 1 Example – A* : Extending Dijkstra

h(n) : heuristic function (d to goal)
g(n) : motion cost (like Dijkstra)

f(n) = g(n)+h(n)

g f
h



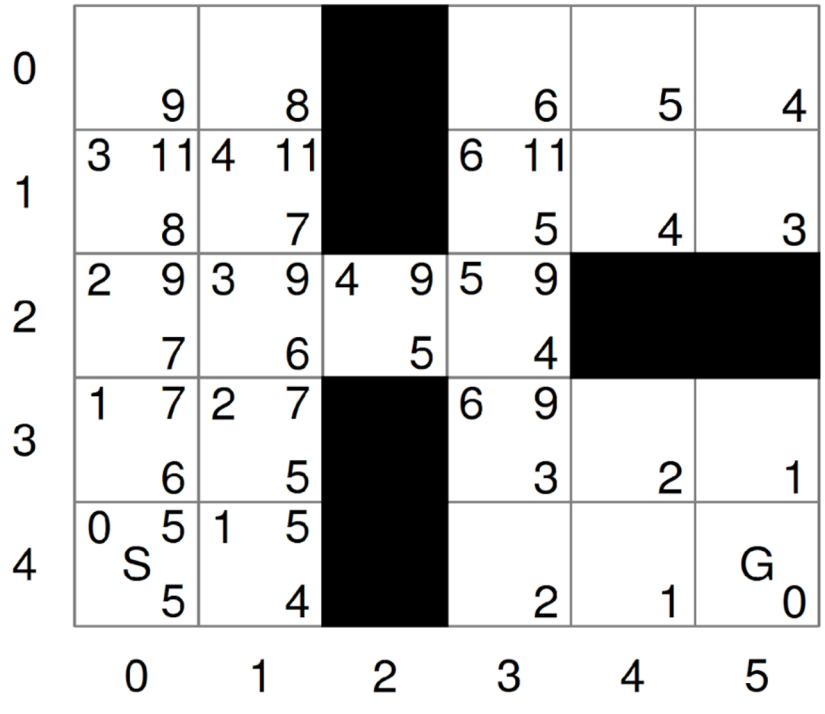


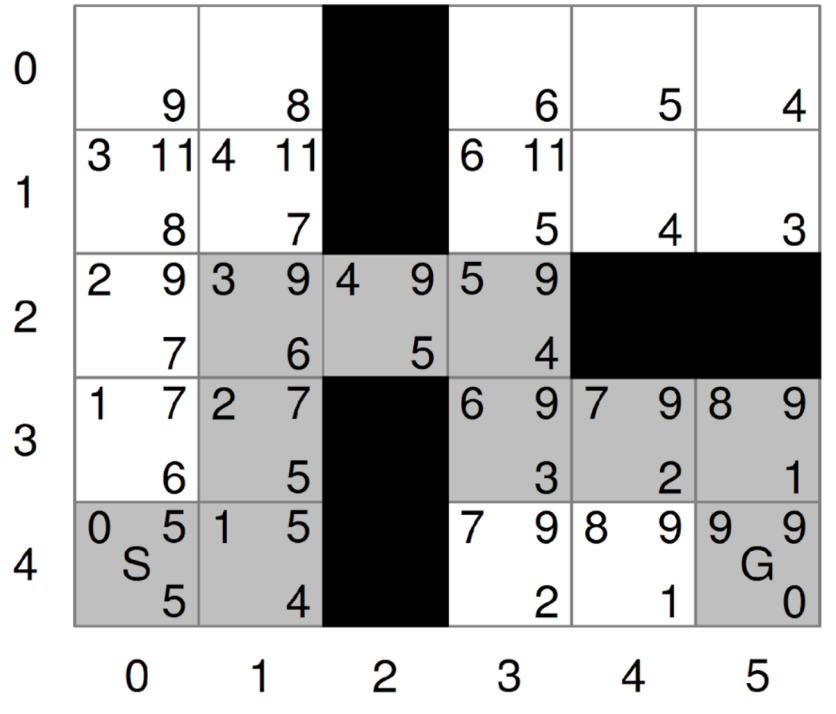
h(n) is simple to compute

Approach 1 Example – A* : Extending Dijkstra

g f

h

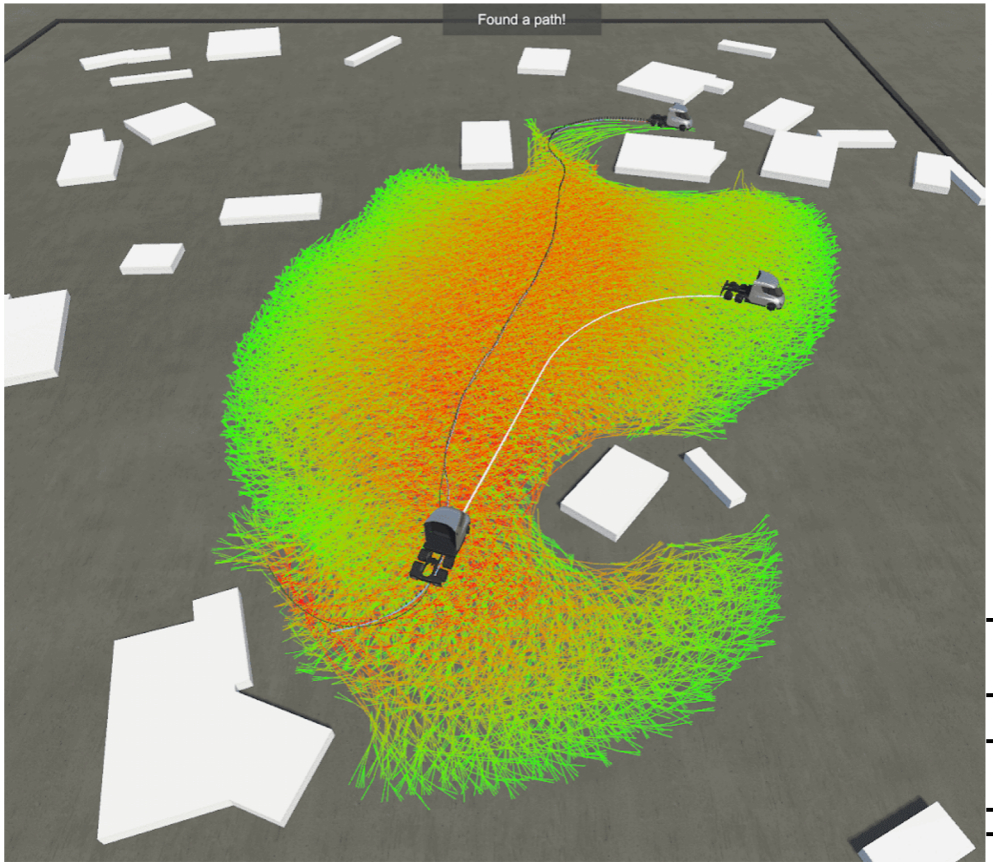




Closer to reality: Hybrid A*

You can add constraints to the choice of cells, for instance

- add a cost when turning
- limit turning angle and access only to some cells, fitting to the possibility of your vehicle (car)
- differentiate forward and backwards motion

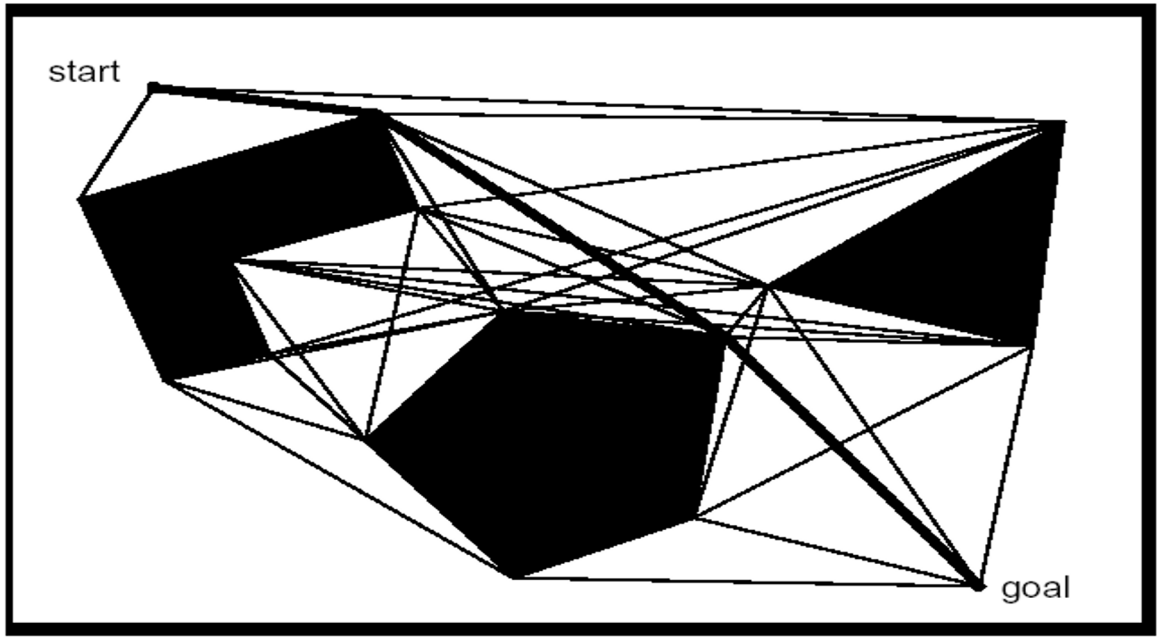


blog.habrador.com

Approach 1 Finding the road – Visibility Graphs

The challenge is to construct a set of roads that enable the robot to go from start to goal, while minimizing the number of total roads.

Joining all pairs of vertices that can see each other (including initial and goal position).



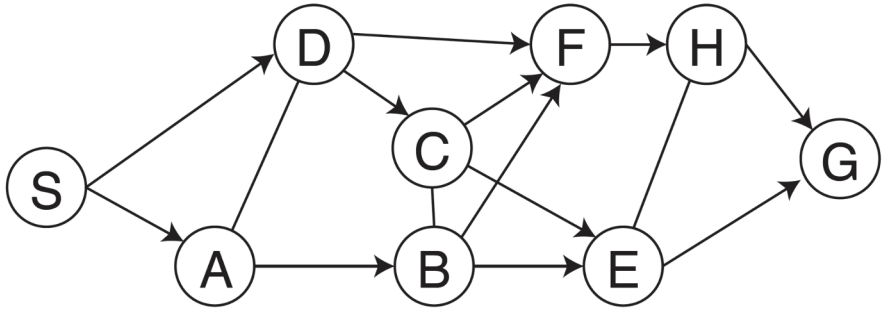
DEMO

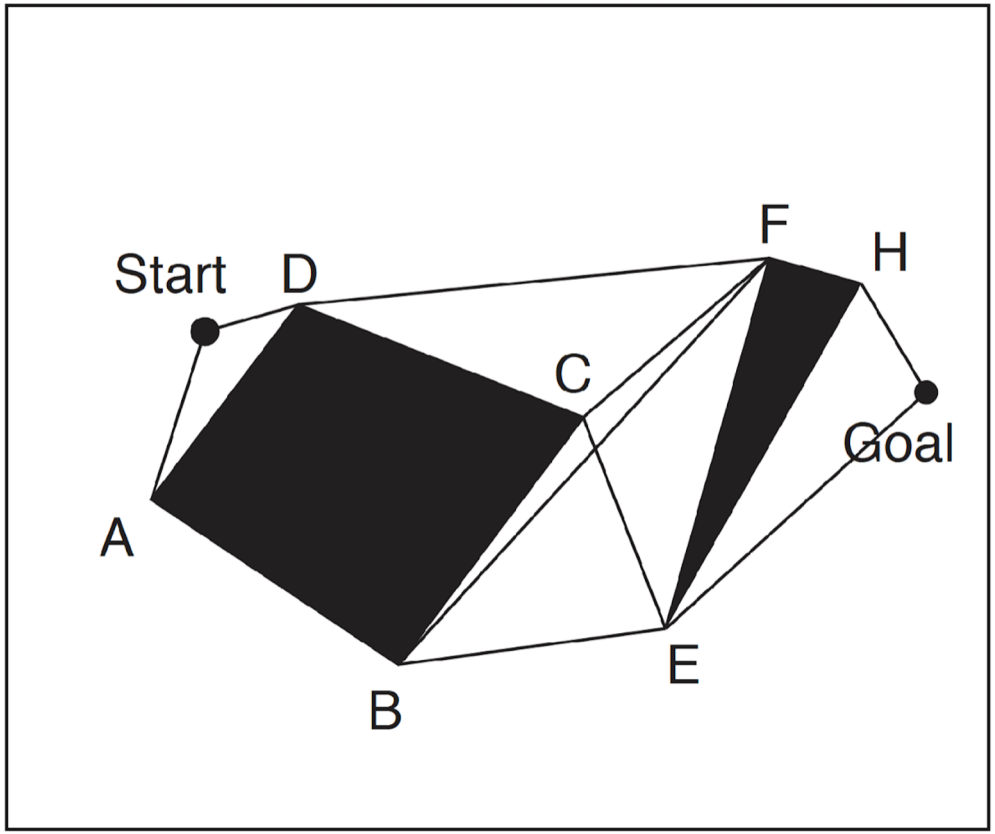
Approach 1 Finding the road – Visibility Graphs

Search for shortest path length using a graph search technique.

+ Complete.

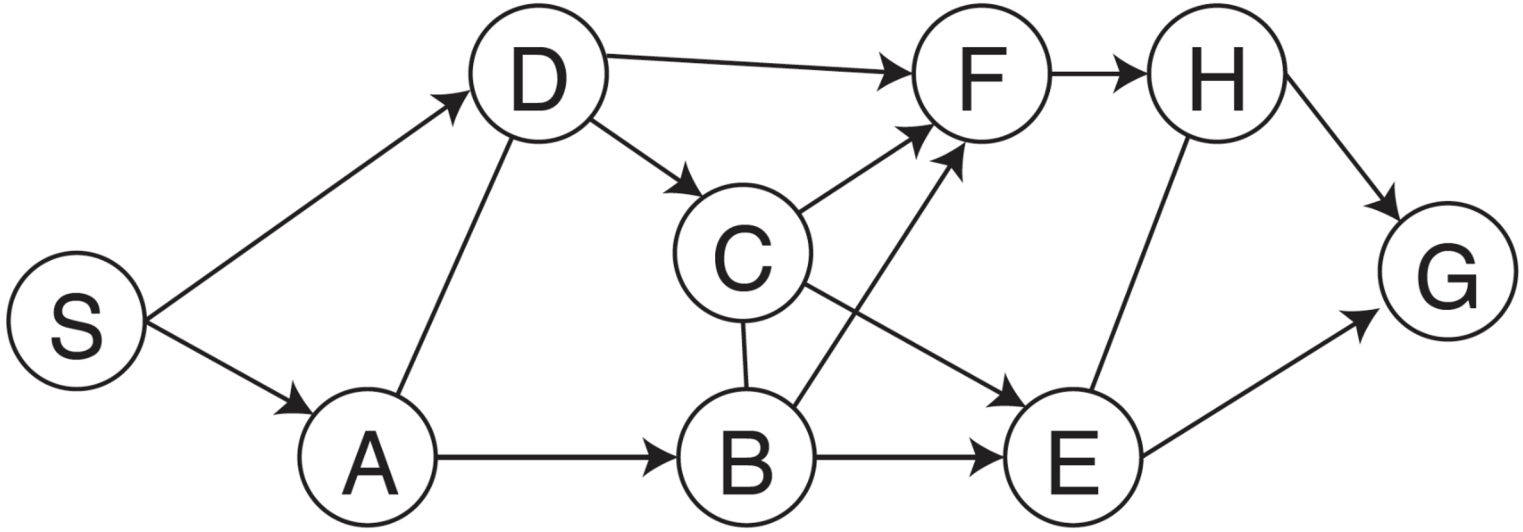
- Tend to graze obstacles
=> requires to grow obstacles to avoid collisions.





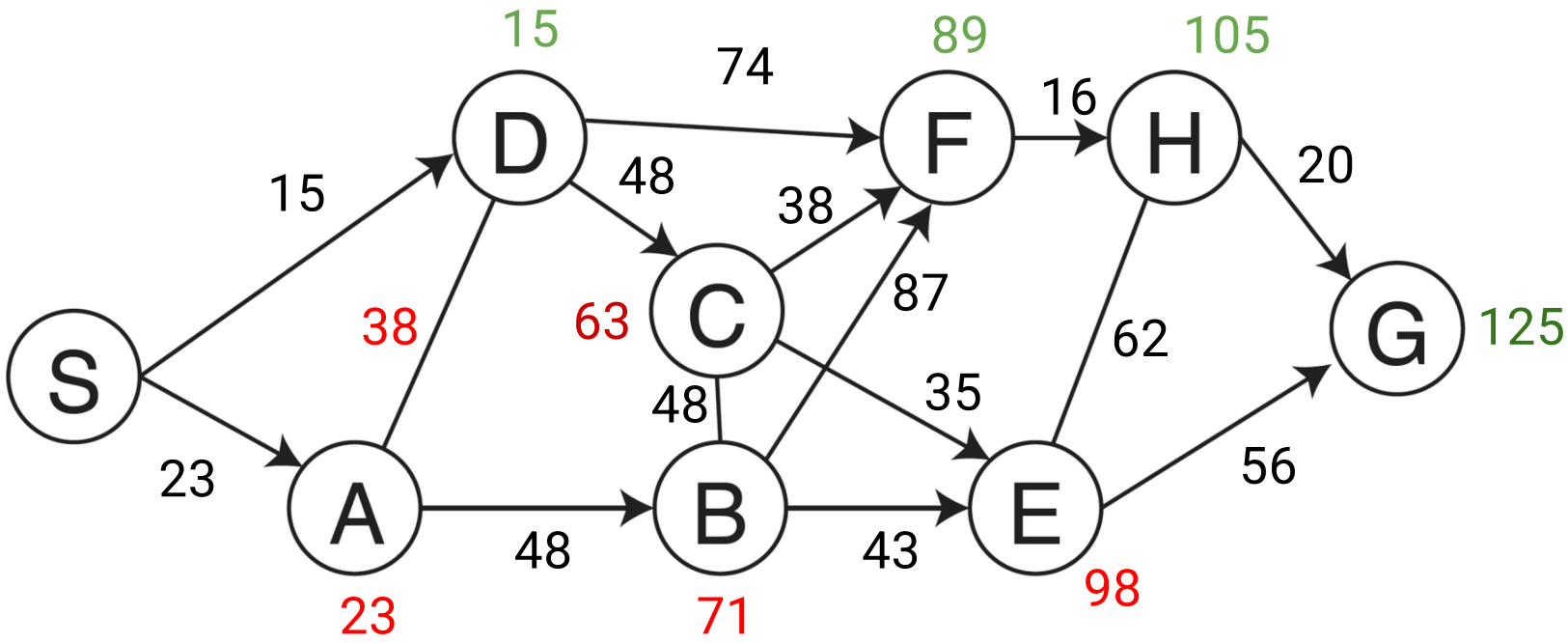
Approach 1 Finding the road – Visibility Graphs

Search for shortest path length using Dijkstra's Algorithm with real distance.



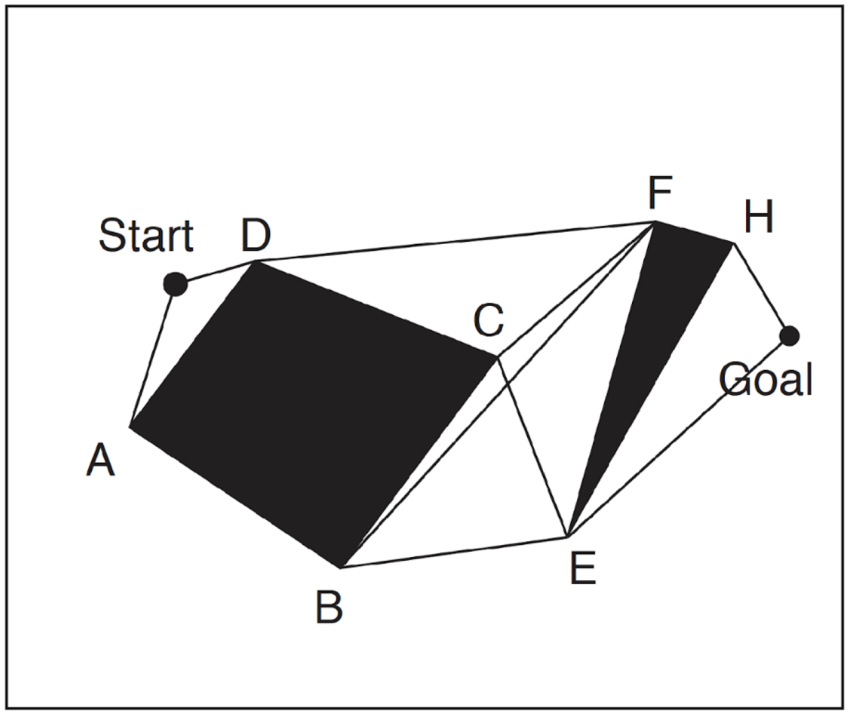
Approach 1 Finding the road – Visibility Graphs

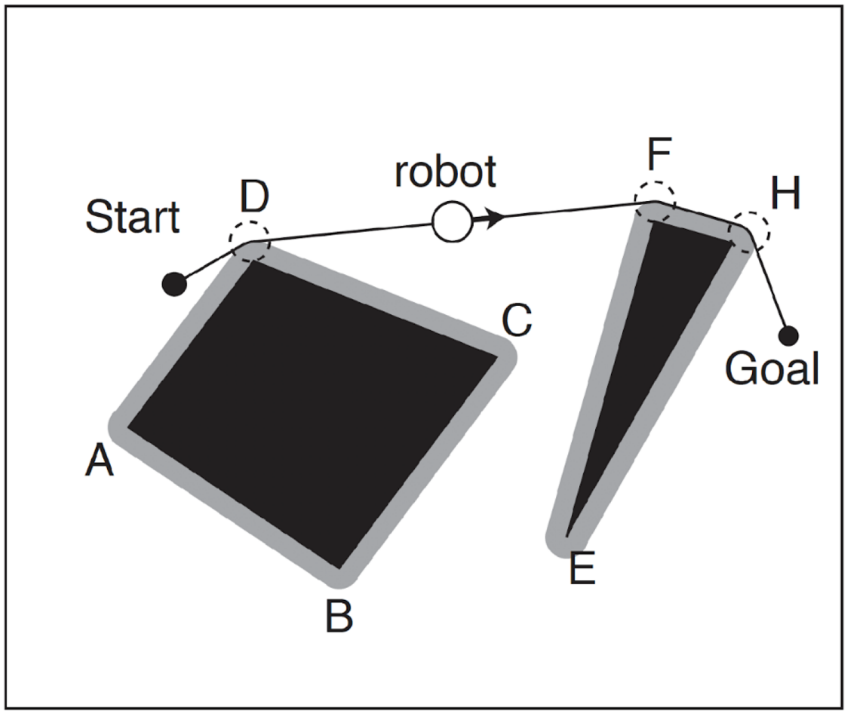
Search for shortest path length using Dijkstra's Algorithm with real distance.



Approach 1 Example – Visibility Graphs

Requires growing obstacles to avoid collisions.





= robot size

Approach 1 Example – Voronoi Diagrams

For each point in the free space compute its distance from the nearest obstacle (growing circles) => results in sharp ridges.

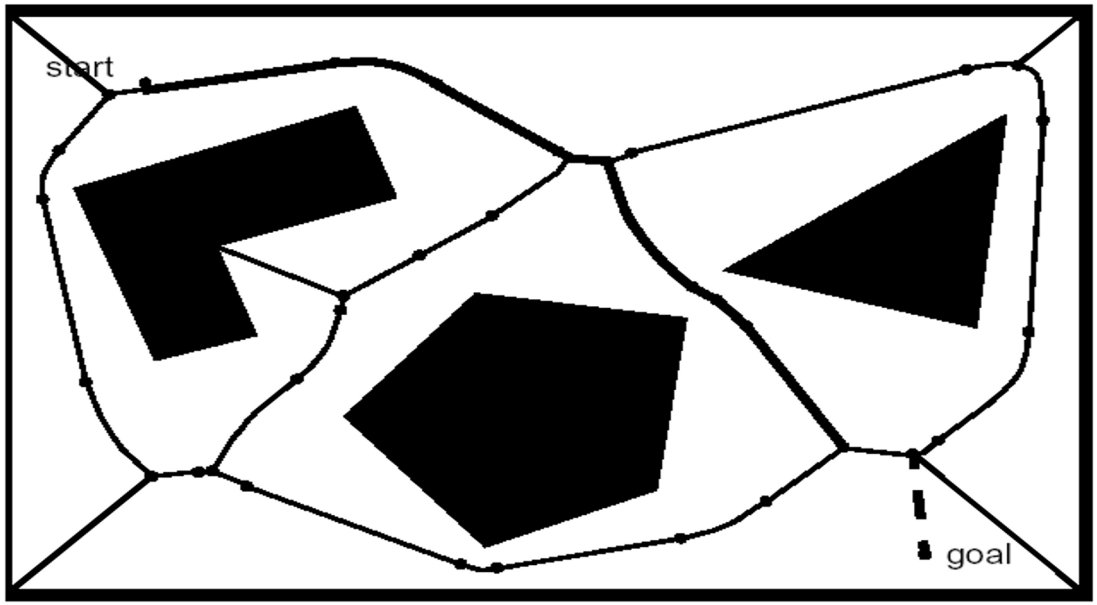
When the configuration space obstacles are polygons, the Voronoi diagram consists of straight and parabolic segments.

Tends to maximize the clearance between the robot and the obstacles.

+ May be straight forward to execute when using range scanner (the robot maximizes the readings of local minima in its sensor values).

- Far from optimal in the sense of total path length.

- May be problematic for localization with short range sensors since no obstacle may be sensed most of the



Approach 2 – Cell Decomposition

Divide free space into simple, connected regions called cells.

Determine which open cells are adjacent and construct a *connectivity graph*.

Find cells in which the initial and goal positions lie and search for a path in the connectivity graph to join them.

From the sequence of cells found with an appropriate search algorithm, **compute a trajectory within each cell**, e.g.

- passing through the midpoints of cell boundaries or
- by sequence of line following movements.

Two families of cell decomposition methods:

- *exact*: cells are either completely free or completely occupied;
- *approximate*: not taking care of the obstacle geometry.

Approach 2 Example – Exact Cell Decomp.

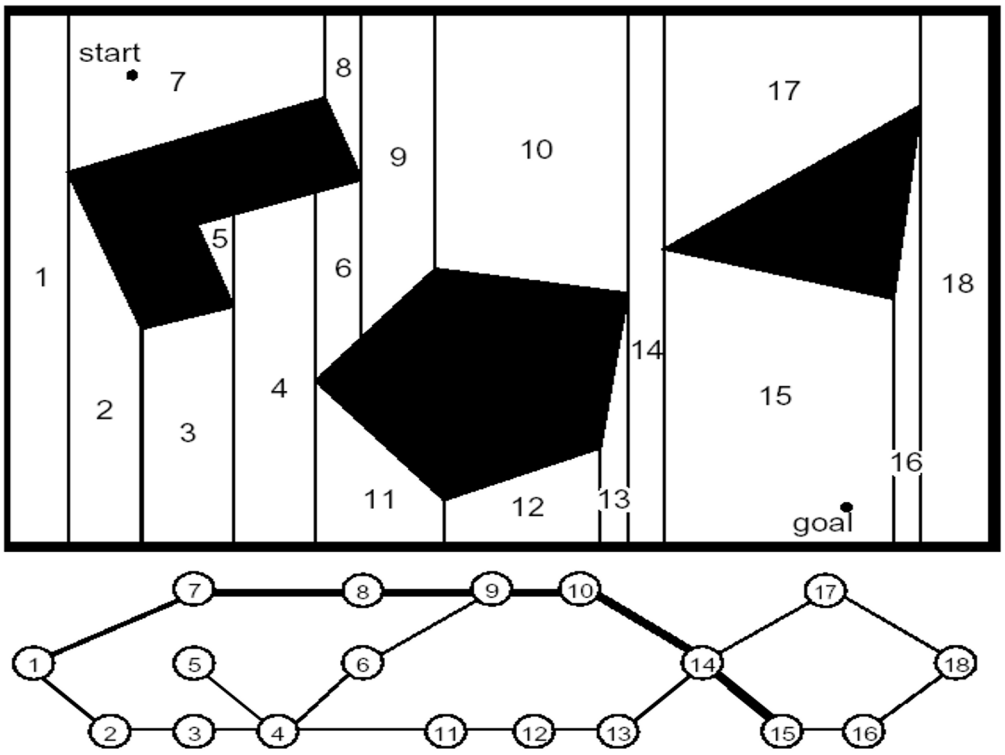
The boundary of the cells is based on geometric criticality, for example the edges of the obstacles.

Underlying assumption: the particular position of the robot within each cell of free space does not matter.

+ Number of cells depend on density and complexity of objects in the environment.

+ Complete.

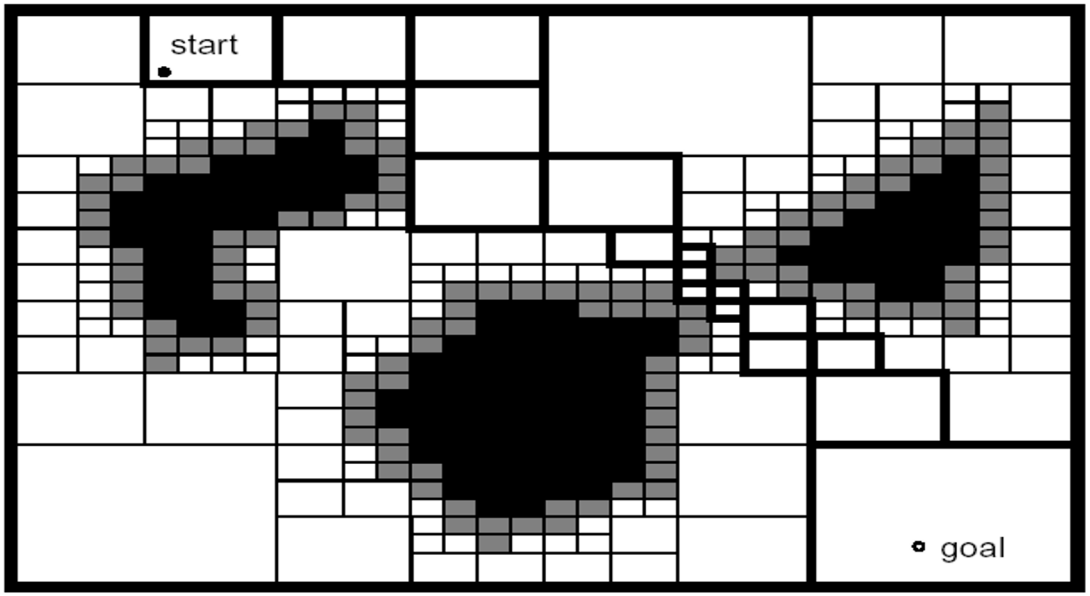
- Dependent on the density and complexity of obstacles.



Approach 2 Example – Approx Cell Decomp. 1

First example: adaptive cell decomposition

- The rectangle is decomposed into 4 identical rectangles.
- If the interior of a rectangle lies completely in free space or on an obstacle, it is not further decomposed. Otherwise, it is recursively decomposed into 4 rectangles, and so on. Stop at a given minimal cell size.



+ Adapted to the complexity of the environment.

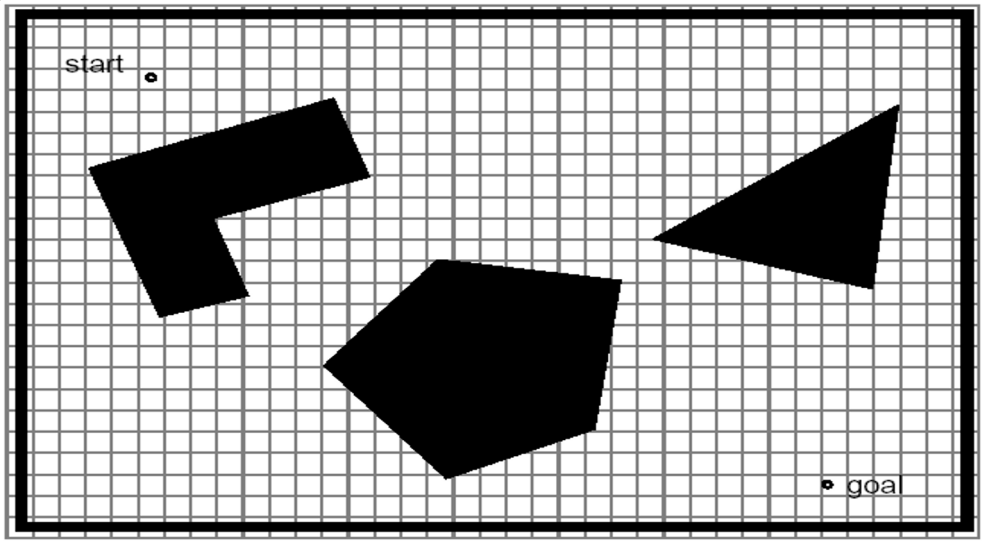
- May not be complete (depending on the minimal cell size).

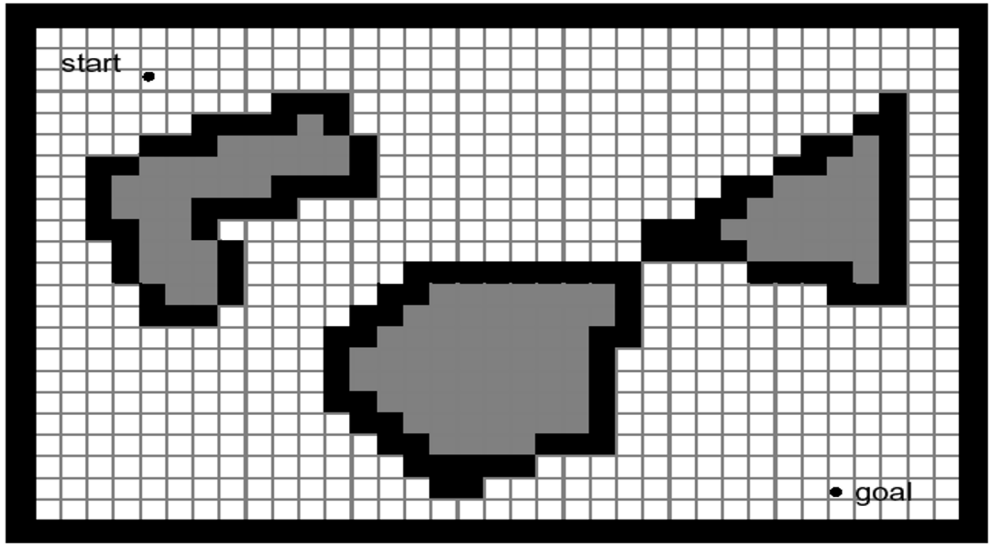
Approach 2 Example – Approx Cell Decomp. 2

2nd example: fixed grid-size decomposition

- Narrow passageways can be lost => may not be complete.

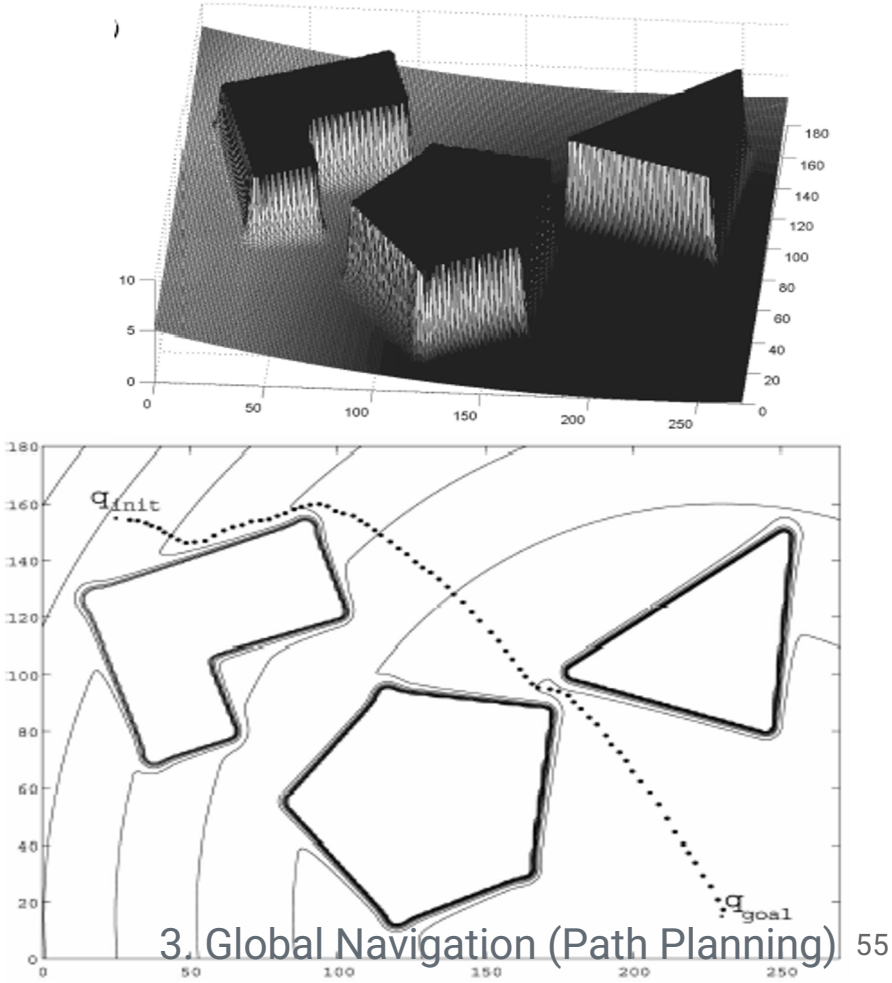
+ Very simple path planning algorithm such as wavefront expansion can be used => computationally efficient.





Approach 2 Example – No graph, Potential Field

- Robot is treated as a particle under the influence of an artificial potential field:
  - Goal generates attractive force.
  - Obstacles are repulsive forces.
- At every location, the direction of the resulting force is considered the most promising direction of motion.



Approach 2 Example – Potential Field Generation

Generation of potential field function $U(q)$, where q is the state variable:

- attracting (goal) and repulsing (obstacle) fields
- summing up the fields
- functions must be differentiable

Generate artificial force field $F(q)$

$$F(q) = -\nabla U(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q) = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix}$$

Approach 2 Example – Potential Field Generation

If the *robot is holonomic*, the control can be: set robot speed (v_x, v_y) proportional to the force $F(q)$ generated by the field (assuming that the dynamics is negligible).

If the *robot is nonholonomic*, a transformation needs to be found. For instance, with a differential-drive, the rotation speed can be set proportional to the angle α between $F(q)$ and the current orientation of the robot, whereas the forward speed can be set proportional to the amplitude of $F(q)$ or $F(q) \bullet \cos(\alpha)$ (in order to avoid moving too fast forward if the current orientation of the robot is not aligned with $F(q)$).

Approach 2 Example – Notes on Potential Field

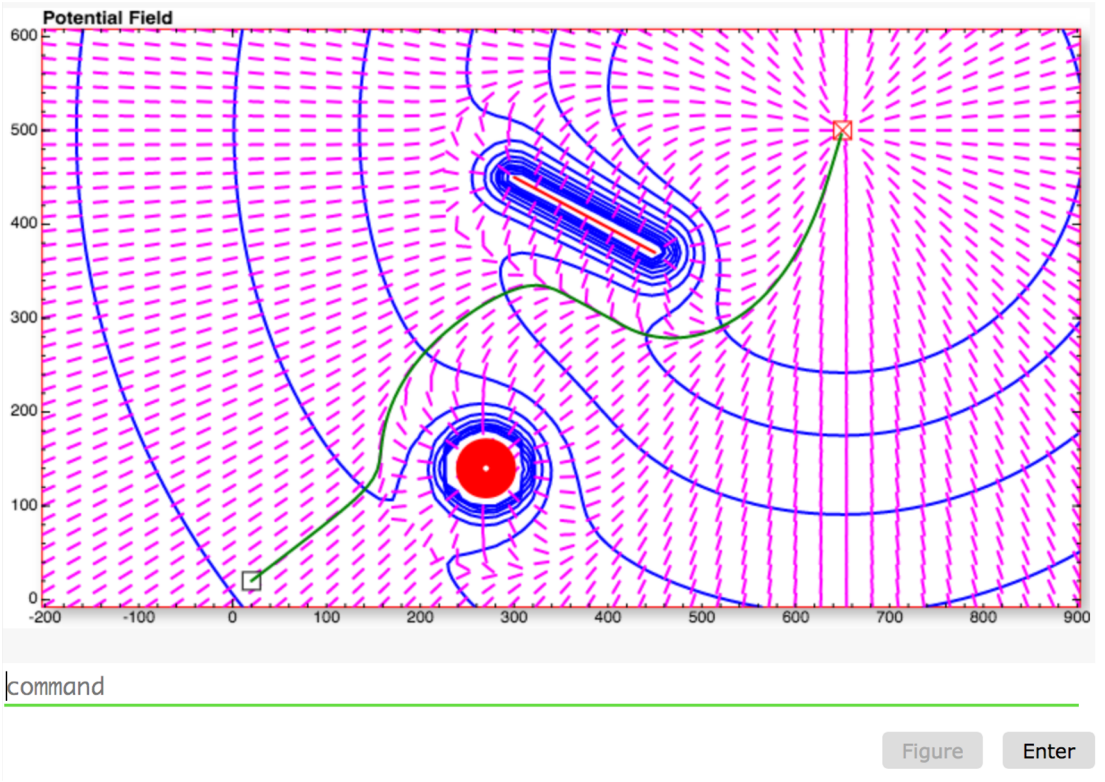
- The potential field methods are usually incomplete.
- Local minima problem exists → can result in oscillations or dead-locks.
- Problem is getting more complex if the robot cannot be considered as a point mass.

+ Easy to implement efficient and reasonably reliable planners.

+ Can be used off-line to draw a complete path that is then followed using a low level controller.

+ Can be used on-line to compute only the force present at current pose and control the robot accordingly.

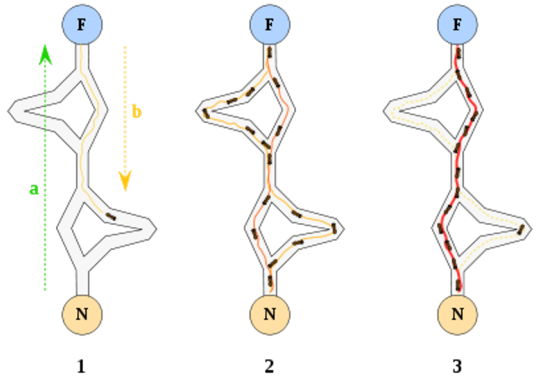
Demo 2 on Potential Field



Stigmergy Based Path Optimisation

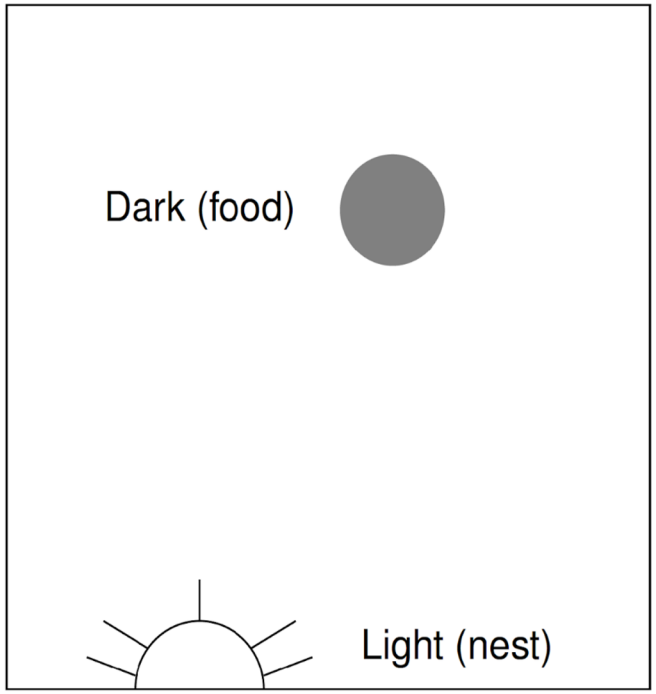
- Create the plan directly in the environment, generating global path with local information

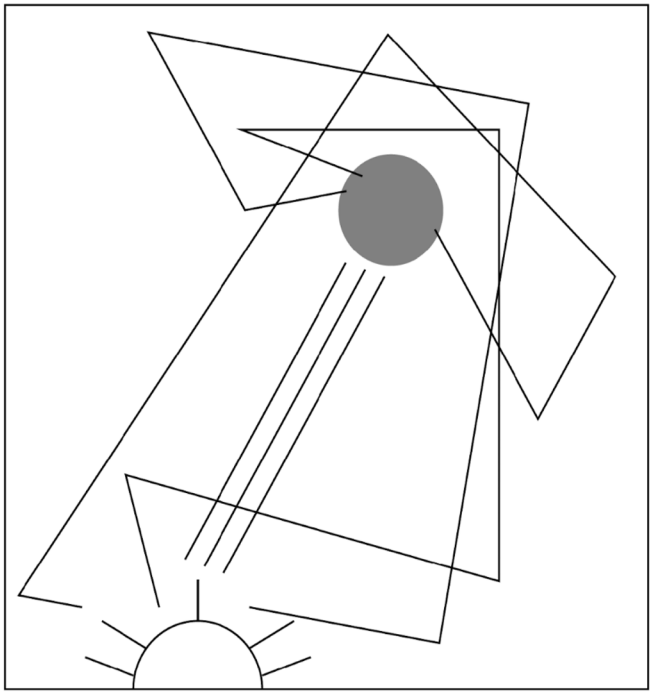
- Marking the environment (simulating pheromones in ants, for instance)
- Can be done by multiple robots or single robots
- A known version: Ant Colony optimisation



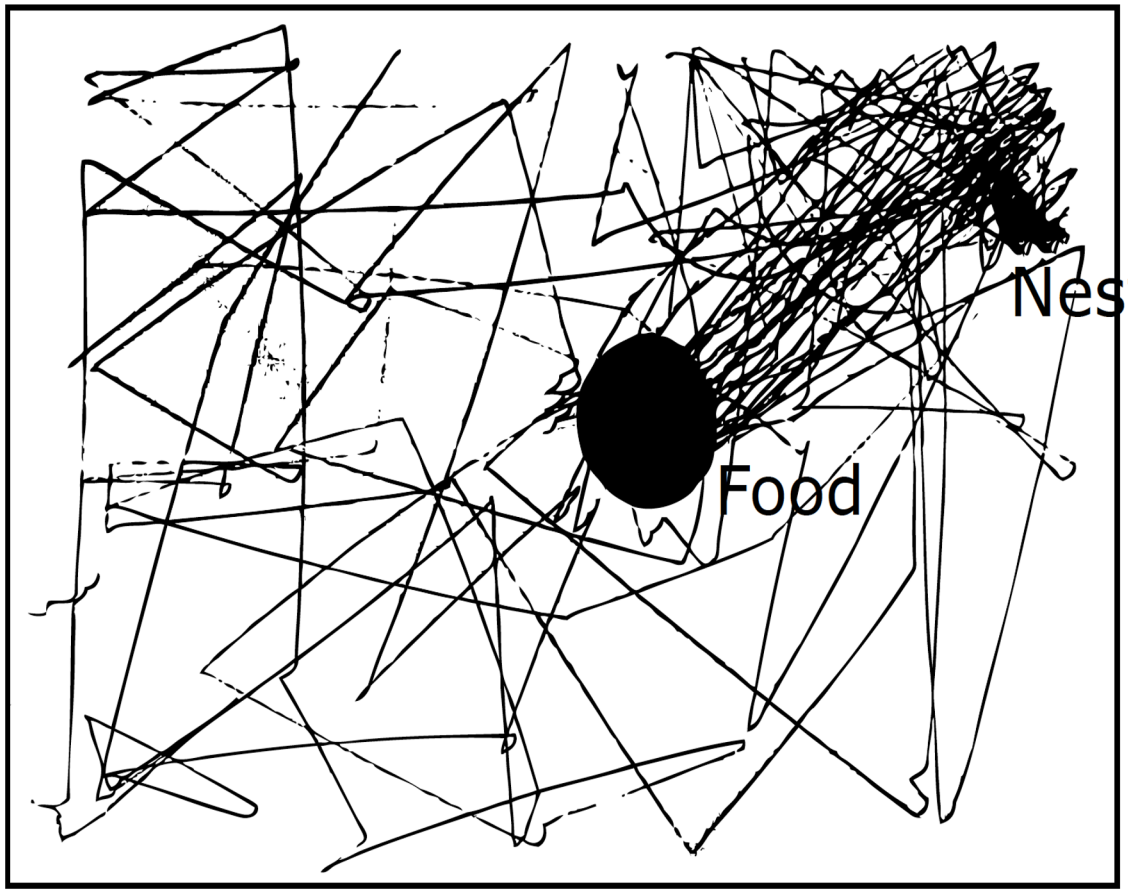
Wikipedia, ACO

Stigmergy Based Path Optimisation



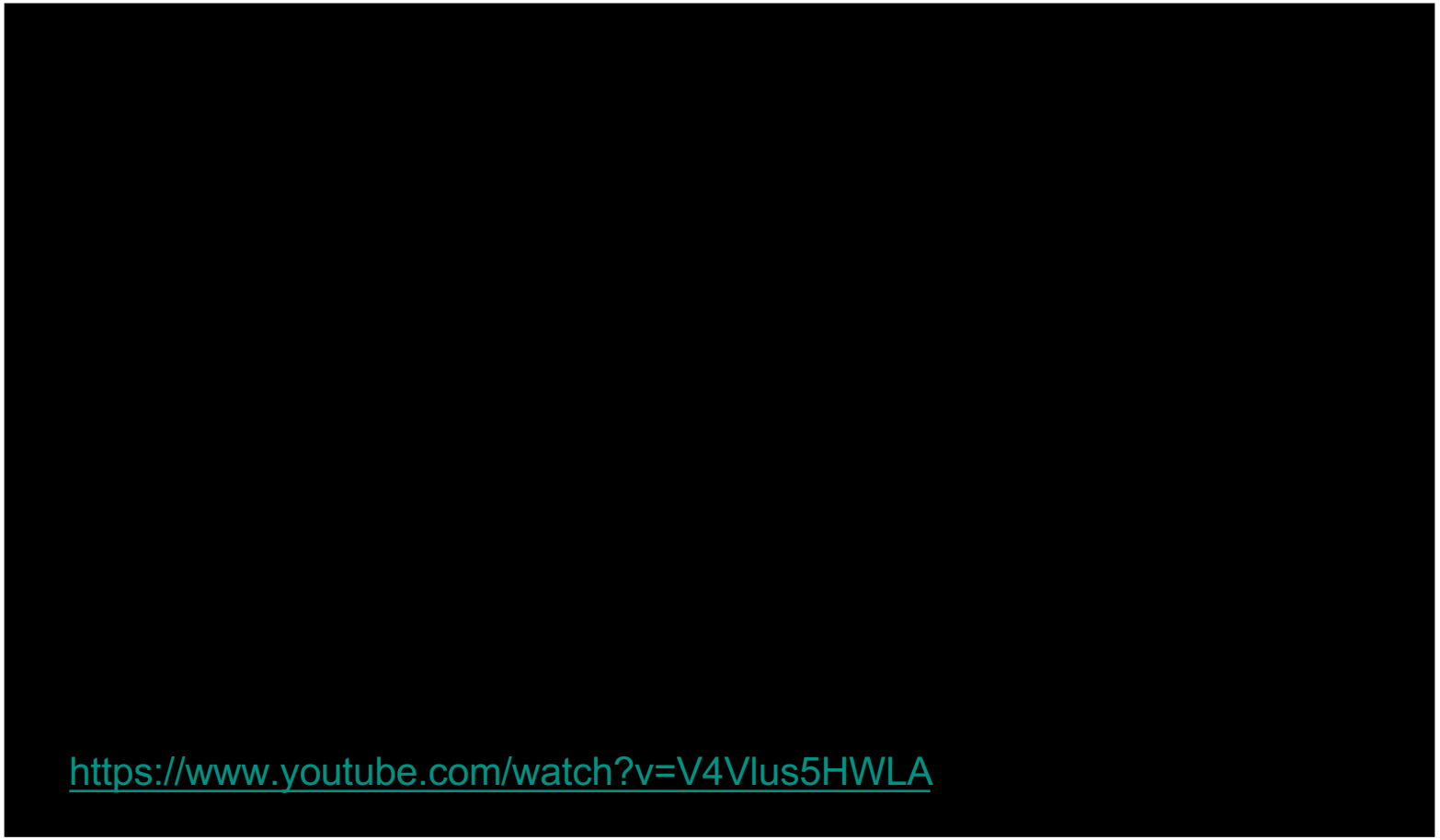


Stigmergy Based Path Optimisation



Stigmergy Based Path Optimisation

- White: looking for food
- Red: food found, back to nest depositing pheromone
- Green: following trail toward food



Ant colony optimization

- Simulation of ants, extracting the optimal path
- Based on pheromone trail with evaporation (unlike previous example)
- Used for optimisation of complex path problem:
  - Vehicle routing
  - Computer network routing
  - Agent scheduling
  - ...

MOBILE ROBOTS 6- Localisation

Prof. Francesco Mondada
Laila El-Hamamsy

EPFL

Bibliography and Sources

Elements of Robotics (ch.5, ch.8 and 9), M. Ben-Ari, F. Mondada, Springer, 2017.

Springer Handbook of Robotics (ch. 45, 46 and 60), B. Siciliano, and O. Khatib (Eds.), 2nd edition, Springer, 2016.

Probabilistic Robotics by Thrun, Burgard, and Fox, MIT Press, 2005

Mobile Robots - EPFL - J.-C. Zufferey, Felix Schill, 2013.

Mobile Robot Localisation

The localization problem is the problem of **estimating a robot's pose** relative to a map of its environment.

Unfortunately, the pose can usually not be sensed directly, i.e., most robots do not possess a noise-free sensor for measuring pose (even GPS is not noise-free and does not directly provide orientation).

Usually, a single sensor measurement is insufficient to determine the pose

-> data from various sources need to be combined (sensor fusion),

-> the robot often has to integrate over time to determine its pose.

Bayesian filtering (probabilistic algorithms) is widely used to recursively merge motion commands and sensor data into an updated pose.

A Taxonomy Of Localisation Problems

Local vs. Global Localisation

- ***Position tracking*** addresses the problem of accommodating the local uncertainty of a robot whose initial pose is known.
- ***Global localization*** is the more general problem of localizing a robot from scratch.
- ***Kidnapping*** is a localization problem in which a well-localized robot is “secretly” deported somewhere else without being told.

Static versus dynamic environments

The difficulty of the localization problem is clearly dependent of the degree to which the environment changes over time.

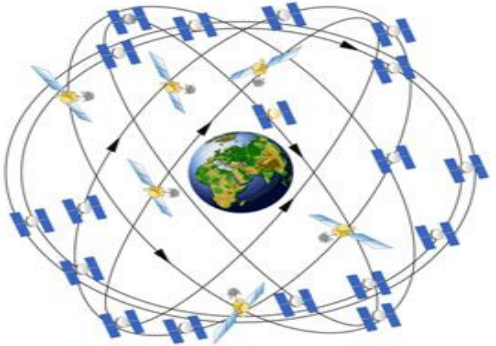
Passive versus active approaches

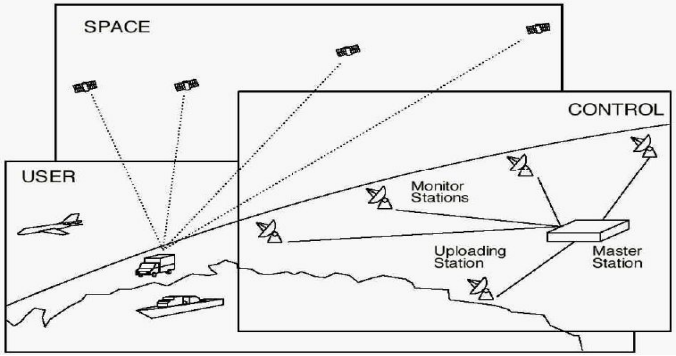
- Passive localization approaches are filters: they process data acquired by the robot but do not control the robot.
- Active techniques control the robot during localization with the purpose of minimizing the robot’s uncertainty (directed exploration).

Table of Contents

Global Positioning System (GPS) – Facts

- Developed for military use by the US (NAVSTAR)
- Became then accessible for commercial applications
- 24 satellites (including three spares) orbiting the Earth every 12 hours at a height of 20’190 km
- 4 satellites are located in each of six planes inclined by 55° with respect to the plane of the Earth’s equator
- Location of any GPS receiver is determined through a time of flight measurement





Most GPS receivers have a position accuracy within 20 m (typ. 10 m) in the horizontal plane and 45 m in the vertical plane, depending on the number of satellites within line of sight and multipath issues.

The update rate is typically between 1 and 10 Hz only.

The European system, Galileo, has similar features in its Galileo Open Service mode. Better precision is available for commercial and governamental purposes.

GPS Technical Challenges

Real time update of the exact location of the satellites:

- monitoring the satellites from a number of widely distributed ground stations
- master station analyses all the measurements and transmits the actual position to each of the satellites

Time synchronization:

- ultra-high precision time synchronization is extremely important
- roughly 0.3m/ns => position accuracy proportional to precision of time measurement
- atomic clocks on each satellite

Precise measurement of the time of flight

- the receiver correlates a pseudocode with the same code coming from the satellite
- the delay time for best correlation represents the time of flight
- quartz clock on the GPS receivers are not very precise
- the range measurement with four satellites allows to identify the three values (x, y, z) for the position and the clock correction ΔT

Interferences with other signals, reflections

Already one of the key sensors for outdoor mobile robotics but not applicable for indoor robots

Beacon Based Positioning

Beacon-based navigation has been used since the humans started to travel

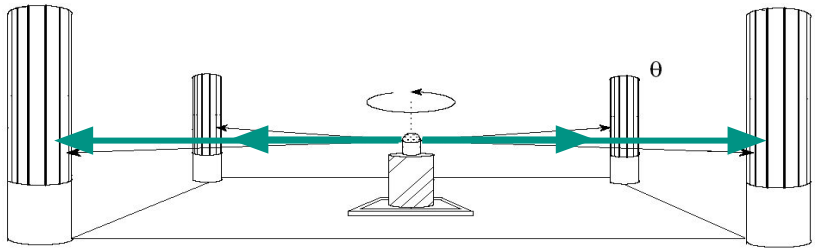
- Natural beacons (landmarks) like stars, mountains or the sun
- Artificial beacons like lighthouses

Beacons = signaling devices of precisely known position.

Major drawback with the use of beacons indoors:

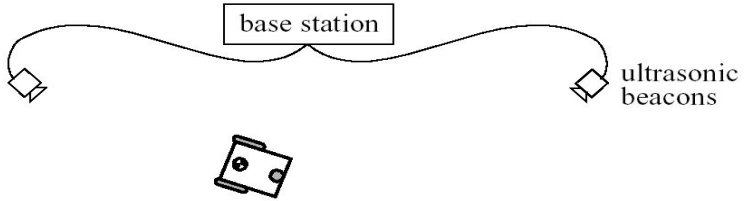
- Beacons require changes in the environment -> costly
- Limit flexibility and adaptability to changing environments

Passive Beacons : optical / retroreflective of known pos.



Distance and heading of at least two beacons are measured (e.g. using a scanning laser range finder)

Active Ultrasonic Beacons



Robots must know emitter locations & deduce their pos based on TOF. Accuracy depends on: 1) Precise location of the beams 2) Time synchronization (e.g. RF/IR) 3) Measurement of TOF

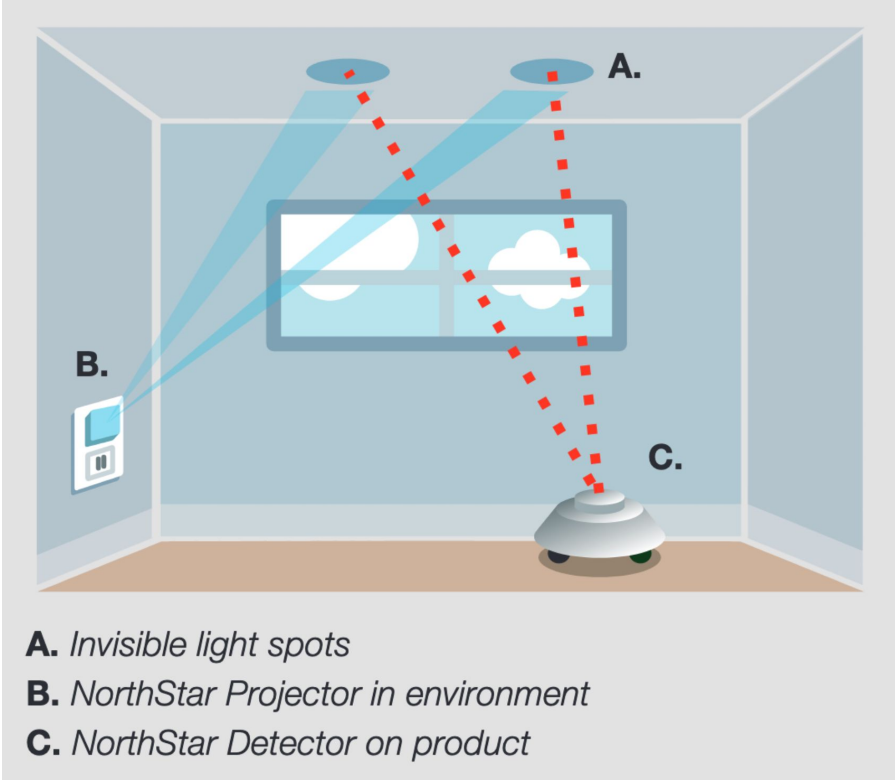
Beacon Based Positioning for home use

Example of system: NorthStar (Evolution Robotics / iRobot)

- IR beacons toward the ceiling: invisible
- One projector: minimally invasive
- Very specific signal: easy to decode

The NorthStar Detector

Example of NorthStar IR Projector



A. *Invisible light spots*
B. *NorthStar Projector in environment*
C. *NorthStar Detector on product*

Example of navigation system for home

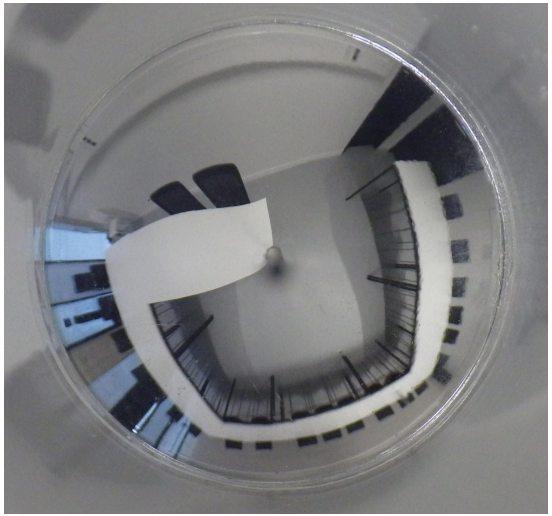
Example of complete system: iRobot Braava using NorthStar

Extremely few systems using this technology: cheaper to localize by vision

Vision-based navigation systems

Several approaches: looking to the ceiling or all around (360 vision)

Samsung: top looking camera



dyson: 360 camera

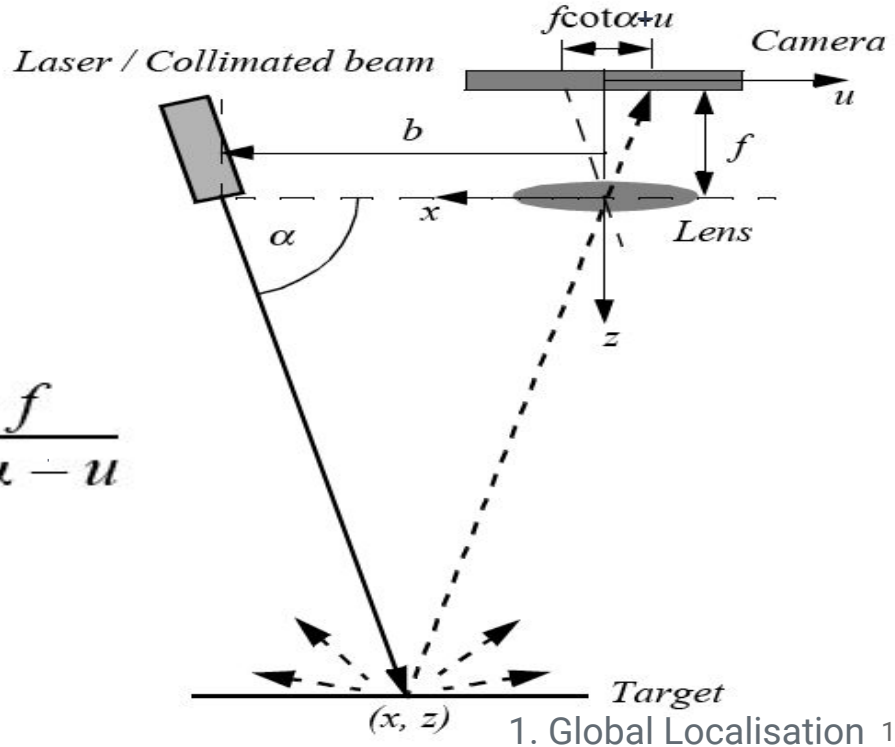
Triangulation – Laser / Collimated Beam

Independent from object detected

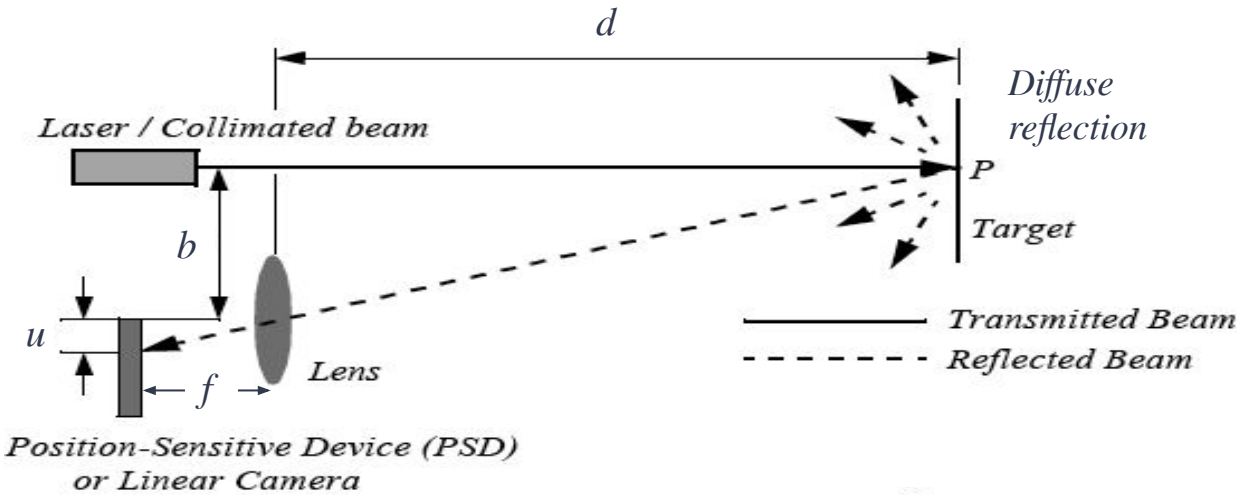
One dimensional schematic of the principle.

From the figure, geometry of similar triangles shows that:

$$x = \frac{b \cdot u}{f\cot\alpha - u} \quad ; \quad z = \frac{b \cdot f}{f\cot\alpha - u}$$



Triangulation – PSD or Linear Camera



$$d = f\frac{b}{u}$$

- Both collimated LED or lasers are used to emit a visible or infrared (IR) beam.
- Position-Sensing Detectors (PSD) are the simplest and fastest way to measure the position of the centroid of a light spot. PSDs are based on a lateral-effect diode that converts the position of incident radiation into signal currents that are directly proportional to the light's position on the active area of the detector.
- Another approach consists of using a linear camera.

Triangulation – application example

Roborock

- Laser sensor
- Triangulation
- Local map

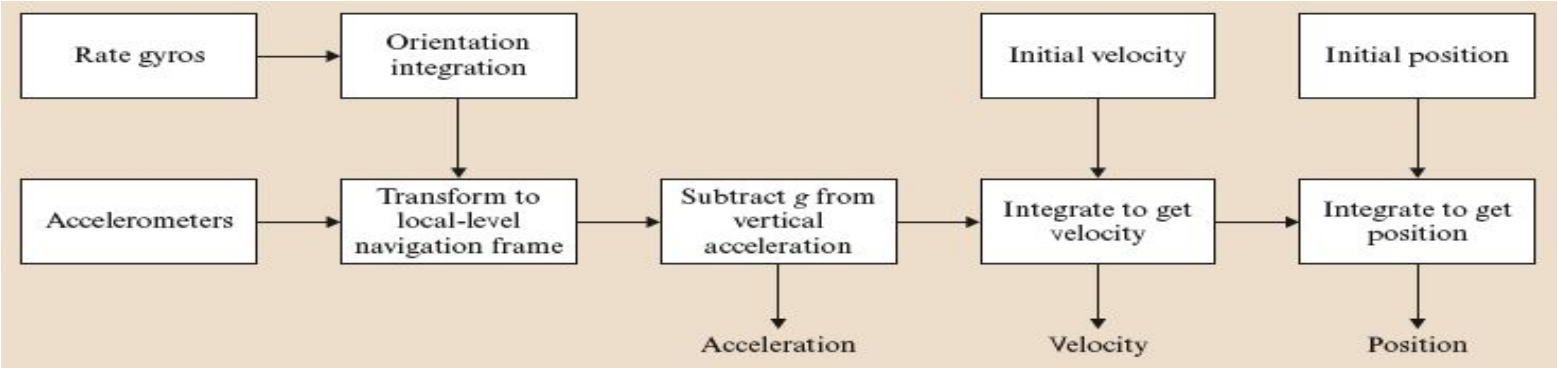
Table of Contents

Inertial Measurement Unit (IMU) Or Inertial Navigation System (INS)

A device that utilizes sensors such as gyroscopes and accelerometers to estimate the relative position (velocity, and acceleration) of a vehicle in motion.

A complete IMU/INS maintains a 6 DOF estimate of the vehicle pose: position (x,y,z) and orientation (roll, pitch, yaw)

This requires integrating information from the sensors in real time.



IMU/INS are extremely sensitive to measurement errors => an external source of information such as GPS and/or magnetometer is often required to correct this (cf. lesson on uncertainties for more information on sensor fusion).
List of IMU/INS products: http://damien.douxchamps.net/research/imu/

Wheel / Motor Incremental Encoders

Measure position or speed of the wheels or steering.

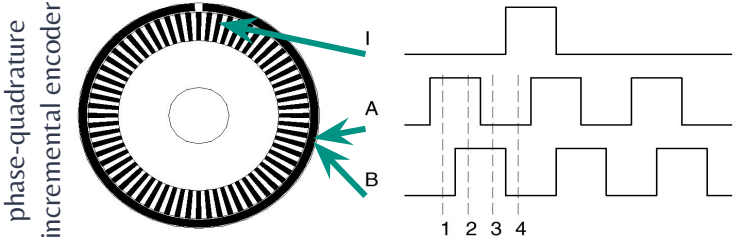
- Measurement types: resistive, inductive, magnetic, optical, etc.
- Optical method is cheap, robust and provides simple digital output.
- Many systems based on magnets and hall sensors (no need of light protection)

Integrate wheel movements to get an estimate of the position -> odometry

Encoders are proprioceptive sensors thus the position estimation in relation to a fixed reference frame is only valuable for short movements.

Incremental Encoders (e.g. Optical)

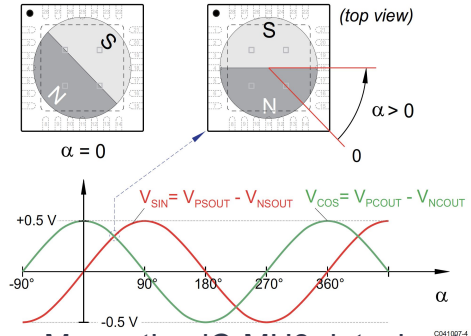
Two signals (90 degrees shift) for direction detection



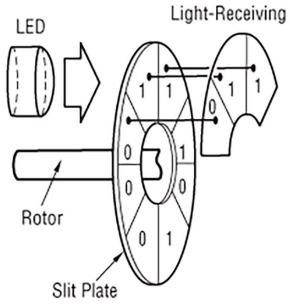
State	Ch A	Ch B
S_1	High	Low
S_2	High	High
S_3	Low	High
S_4	Low	Low

Typical resolutions: 64 - 2048 increments per revolution.

Absolute Encoders (e.g. Magnetic, Optical)



Magnetic - iC-MH8 datasheet

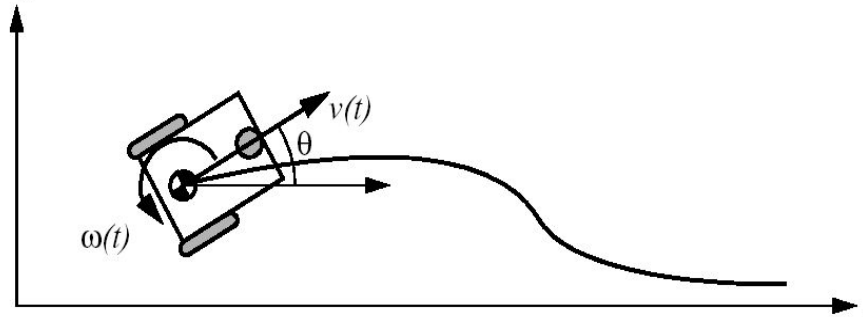


Sensor Output	Angle
011	45°
001	90°
000	135°
100	180°
010	225°
101	270°
110	315°
111	360°

Optical

Odometry of a Differential Drive Robot

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \qquad p' = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}$$



The small displacements (translation and rotation) in the robot frame can be estimated from the encoders as follows:

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \qquad \Delta\theta = \frac{\Delta s_r - \Delta s_l}{b}$$

b : distance between the two wheels.

Expressed in the external fixed frame:

$$\Delta x = \Delta s \cos(\theta + \Delta\theta/2)$$
$$\Delta y = \Delta s \sin(\theta + \Delta\theta/2)$$

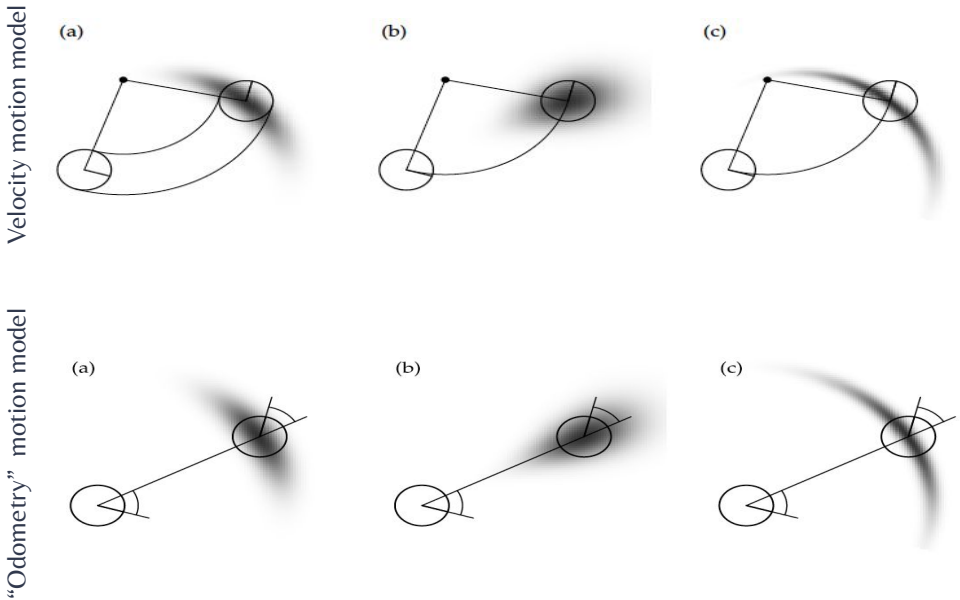
Motion Models $P(X_T \mid U_T, X_{T-1})$

The *motion model* provides a way of estimating in a probabilistic sense the effect of motion on the state.

In mobile robotics, this is usually achieved using odometry (note that encoders are not part of the measurement z !) or low-level speed control.

Examples:

- Velocity motion model: the robot is controlled using two velocities v and ω and follows circular trajectories during Δt
- "Odometry" motion model: u_t is given by the difference between x_t (as provided by odometry) and x_{t-1} ; u_t is transformed into a sequence of 3 steps: rotation, linear motion, rotation



Motion models with various noise parameter settings

Gaussian Error Model

Proposed encoder error model (Gaussian):

$$\Sigma_\Delta = covar(\Delta s_r, \Delta s_l) = \begin{bmatrix} k_r|\Delta s_r| & 0 \\ 0 & k_l|\Delta s_l| \end{bmatrix}$$

Assumptions:

- The two errors of the individually driven wheels are not correlated.
- The errors are proportional to the absolute value of the traveled distances.

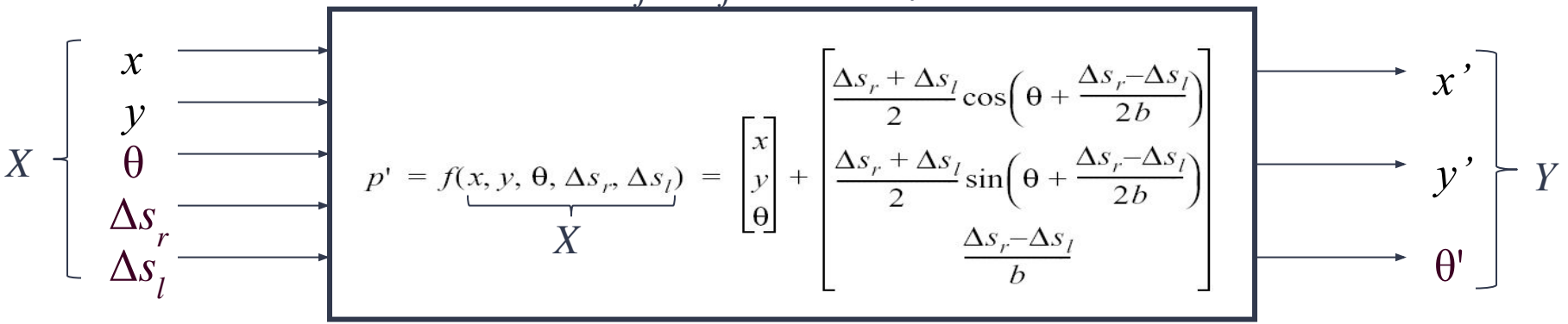
Note that a detailed discussion of odometric errors and a method for calibration can be found in Borenstein and Feng (1995). Another method for on-the-fly odometry error estimation is presented in Martinelli and Siegwart (2003).

⇒ How does error propagate through odometry integration?

⇒ How can we compute uncertainties and visualize them?

Uncertainty in Propagation

$X \sim N(\mu, \Sigma)$ $\quad Y_j = f_j(X_1 \ldots X_n)$ $\quad Y \sim N(f(\mu),\ J(f)\Sigma J(f)^T)$



⇒ Output mean is the image of the input mean.

⇒ Output covariance (uncertainty) can be estimated using the Jacobian of the transformation:
$\Sigma' = J(f)\ \Sigma\ J^T(f)$, where the input covariance matrix is given by:

$$\Sigma = \begin{bmatrix} \Sigma_p & 0 \\ 0 & \Sigma_\Delta \end{bmatrix}_{5\times 5}$$

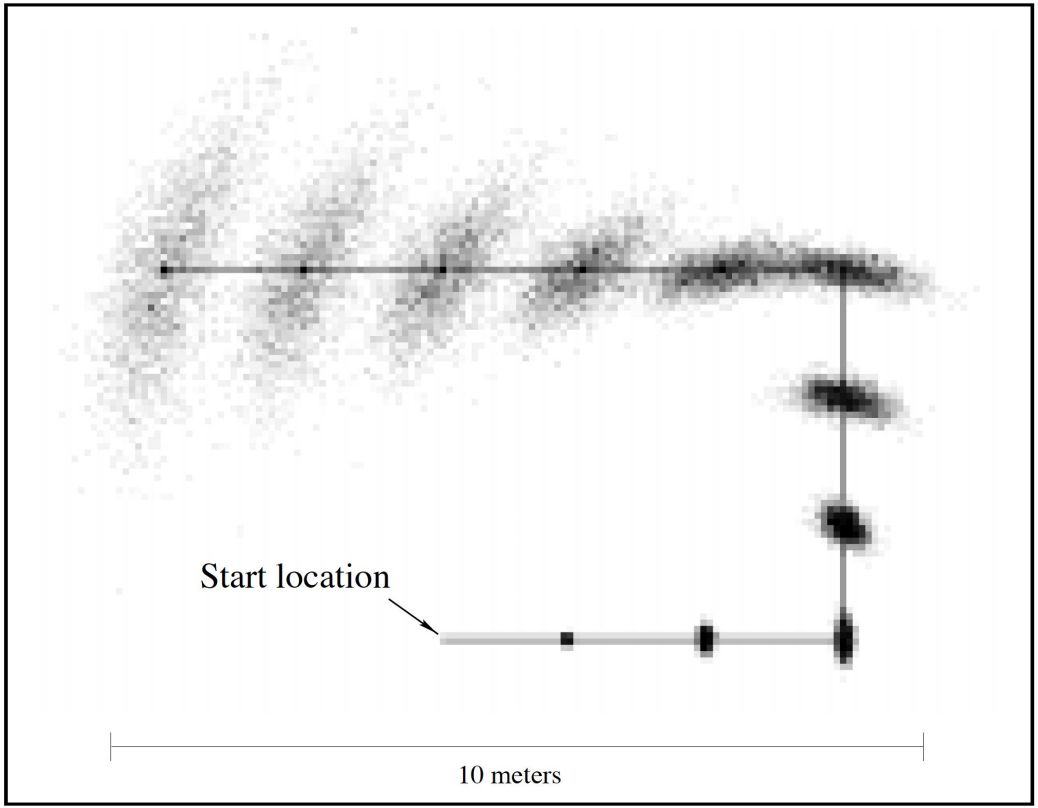
see previous slide

previous covariance matrix on the pose (note that $\Sigma' = \Sigma_p$,)

Note that $\Sigma' = J(f)\Sigma J(f)^T = J_p(f)\Sigma_p J_p(f)^T + J_\Delta(f)\Sigma_\Delta J_\Delta(f)^T$

Effect of Non Deterministic Errors

This graph shows how the odometric uncertainty grows along a straight or curved path. The error is bigger in angle than straight



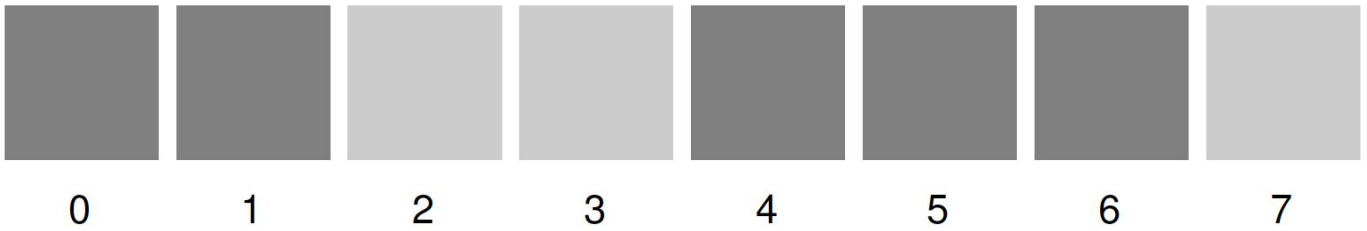
Fox, D., Burgard, W., Dellaert, F., & Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, *1999*(343-349), 2-2.

Table of Contents

Example

Robot moving in a known map, simplified here to 8 positions. The robot has the map and can sense the color of the position (for instance a ground sensor).

Map:

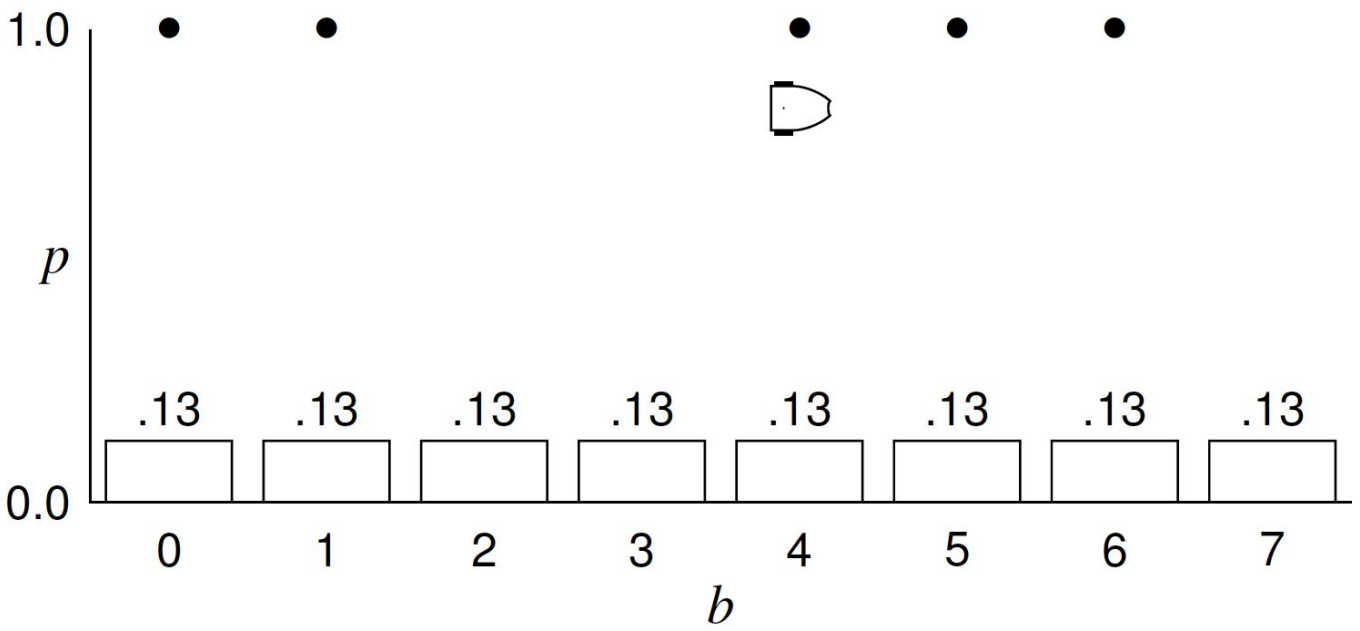


Example

Robot moving in a known map, simplified here to 8 positions. The robot has the map and can sense the color of the position (for instance a ground sensor).

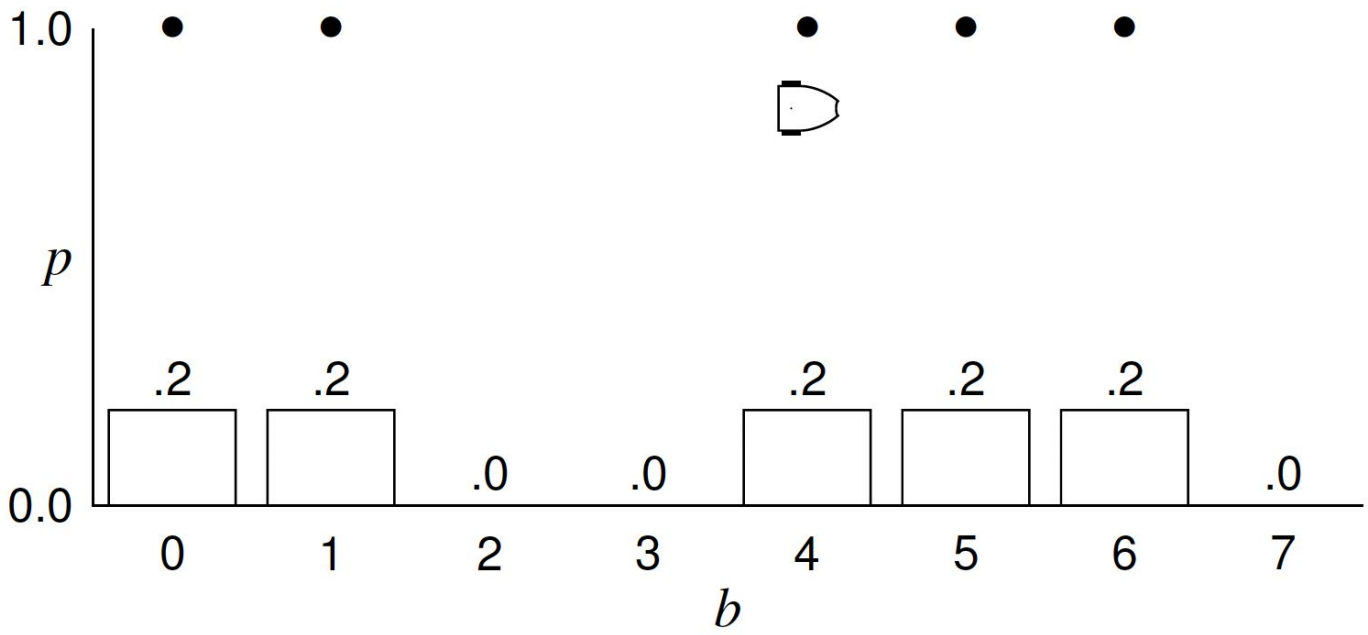
Sensing

Start, no info on position. Probability p same in all positions b.



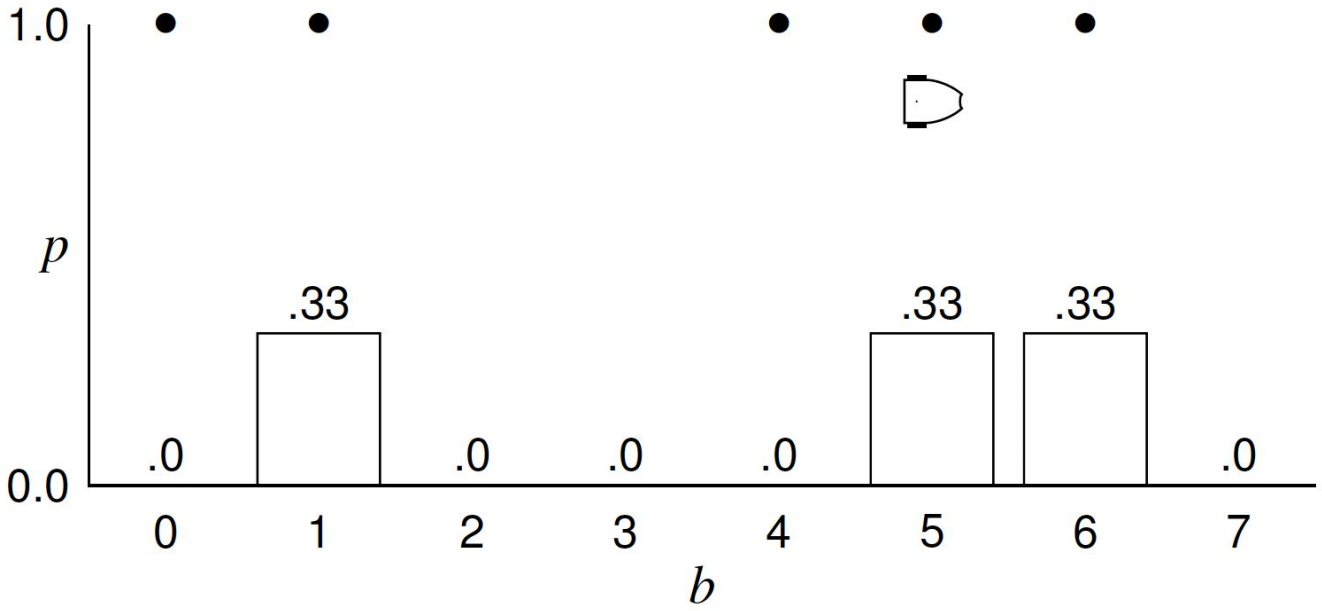
Sensing

Robot senses the ground and get an information on potential position:



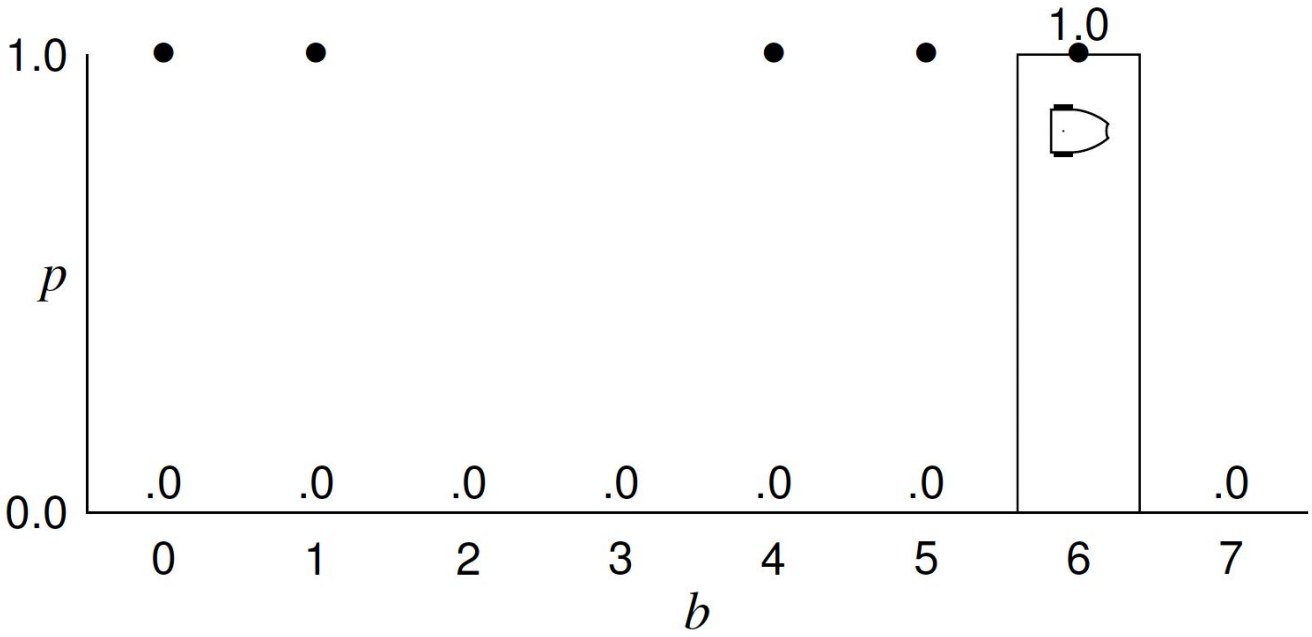
Action

Robot moves and senses again:



After Last Move

Robot moves and senses again:



How to Compute the Probability?

Sense:

p(b|z)=p(z|b).p(b)

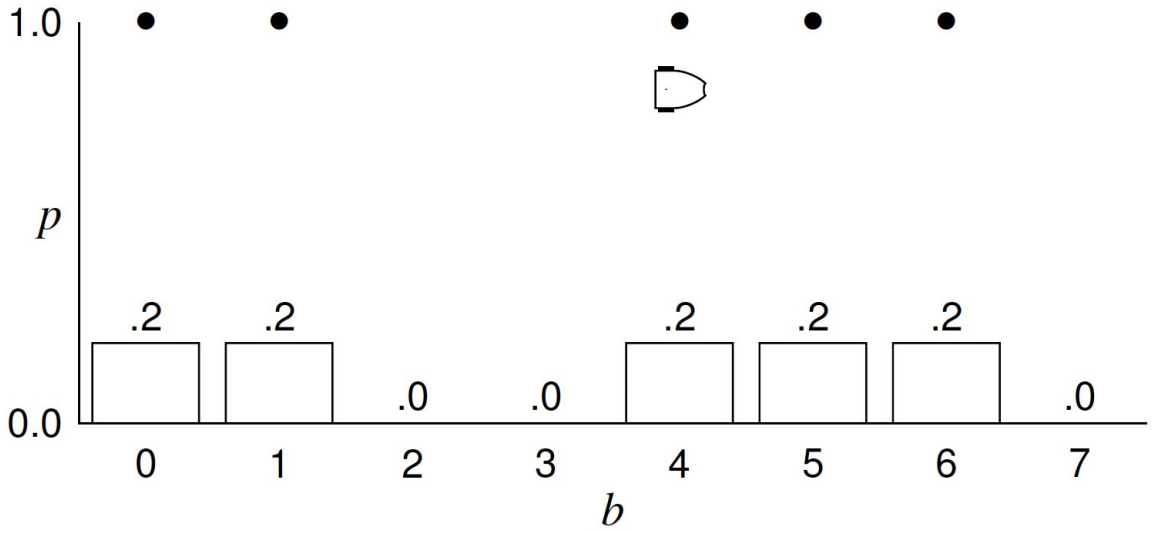
z: measurement

b: state

p(z|b) = 1
if map = measure

p(z|b) = 0
if map ≠ measure

then normalization



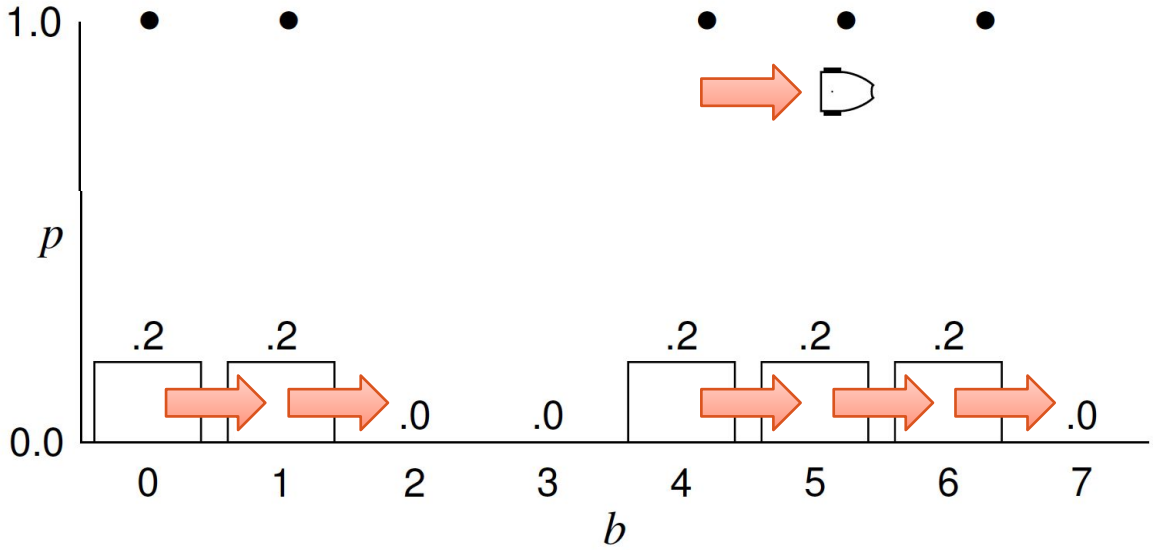
How to Compute the Probability?

Movement:

$p(b_t)=p(b_t|u,b_{t-1}).p(b_{t-1})$

u: action

$p(b_t|u,b_{t-1})$: shift



Uncertainty in Sensing

Measure:

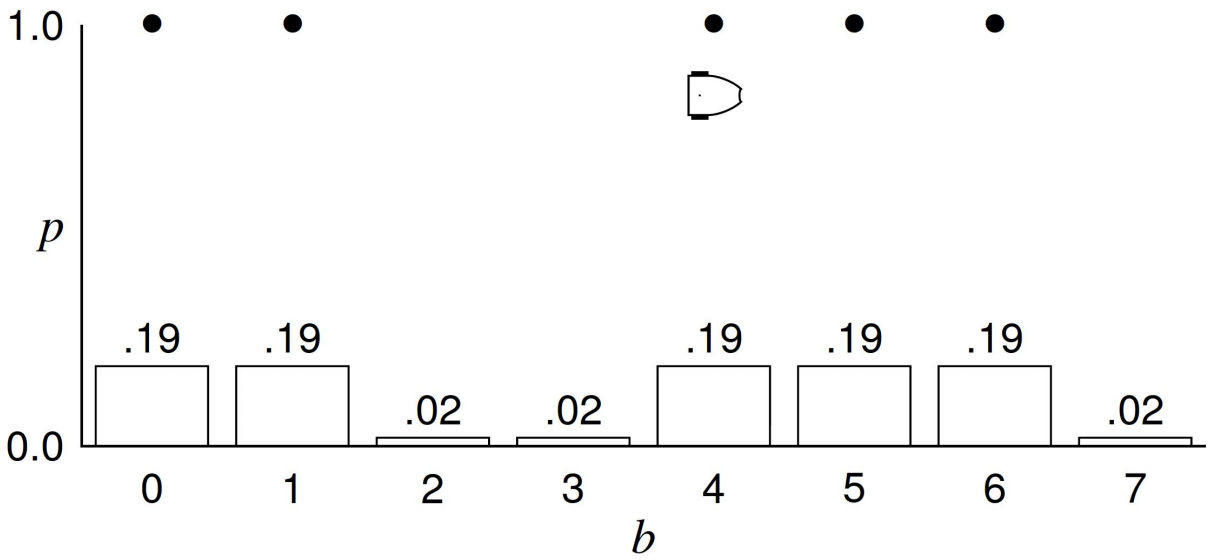
p(b|z)=p(z|b).p(b)

z: measurement

p(z|b) = **0.9**
if map = measure

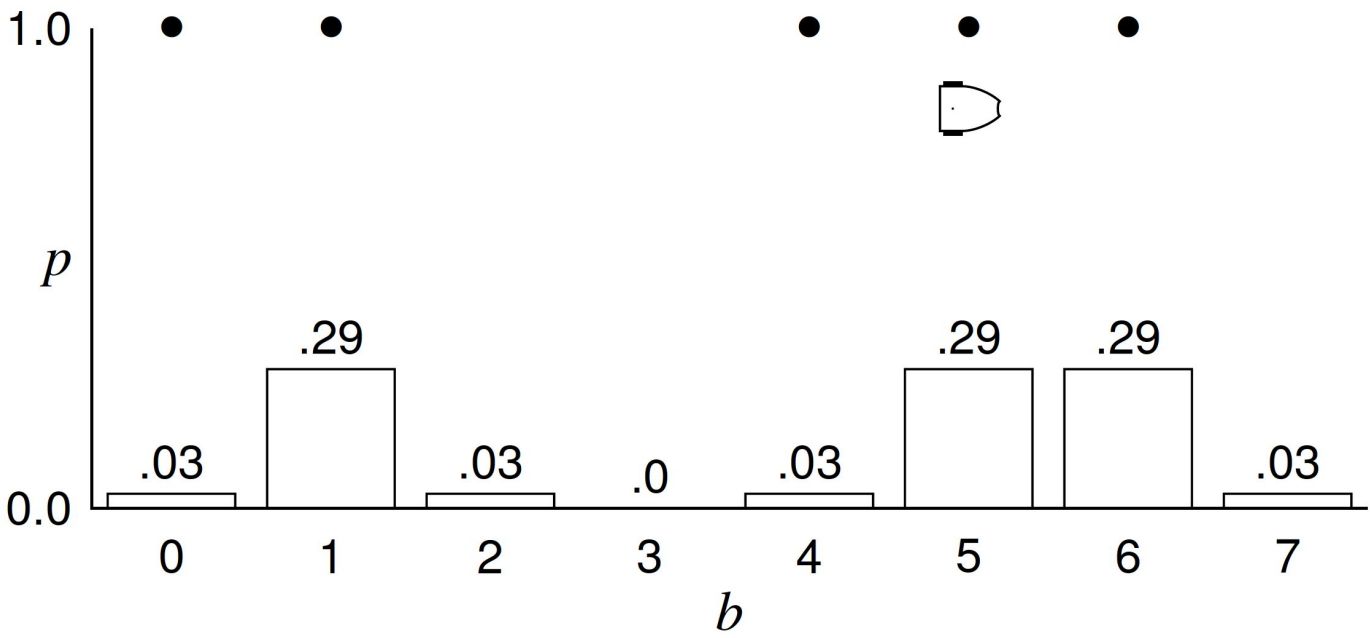
p(z|b) = **0.1**
if map ≠ measure

then normalization



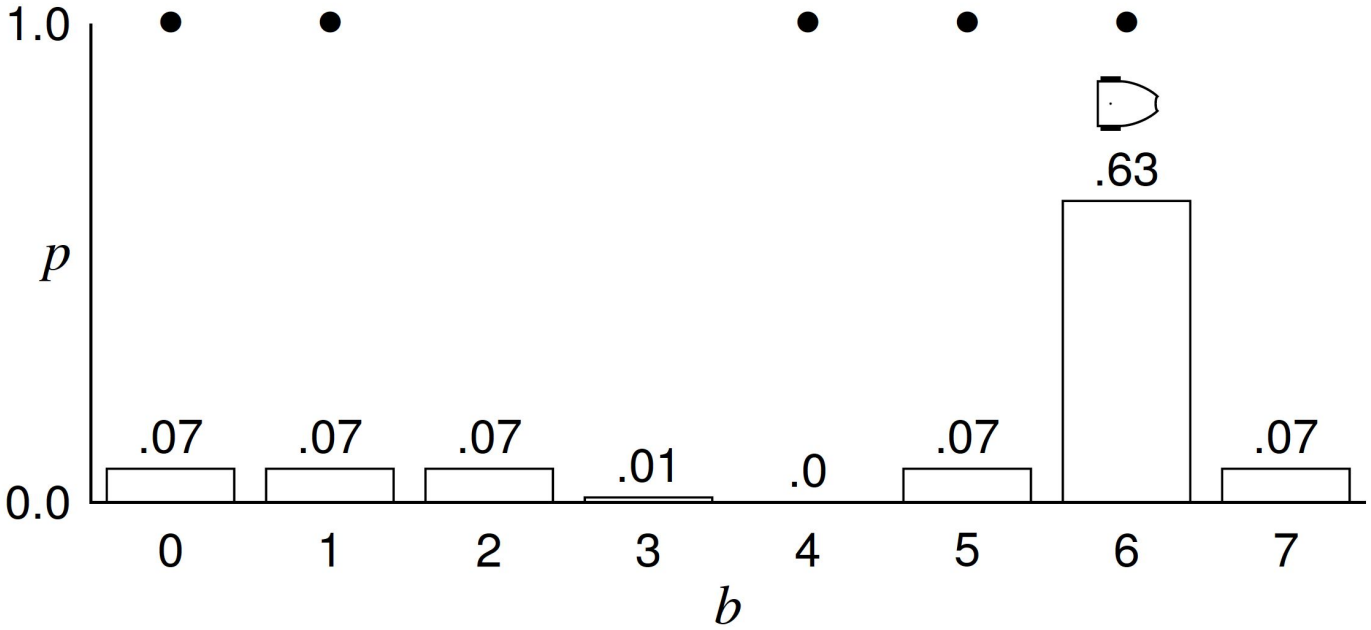
Uncertainty in Sensing

After move:



Uncertainty in Sensing

After move:



Uncertainty in Motion

Movement:

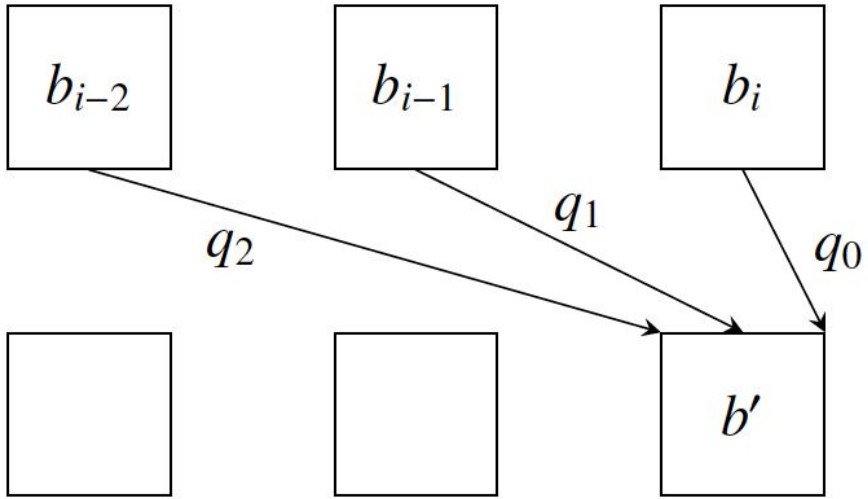
$p(b_t)=p(b_t|u,b_{t-1}).p(b_{t-1})$

u: action

q_1 = **0.8**

$q_0 = q_2$ = **0.1**

$$b'_i = p_i\,(b_{i-2}\,q_2 + b_{i-1}\,q_1 + b_i\,q_0)$$



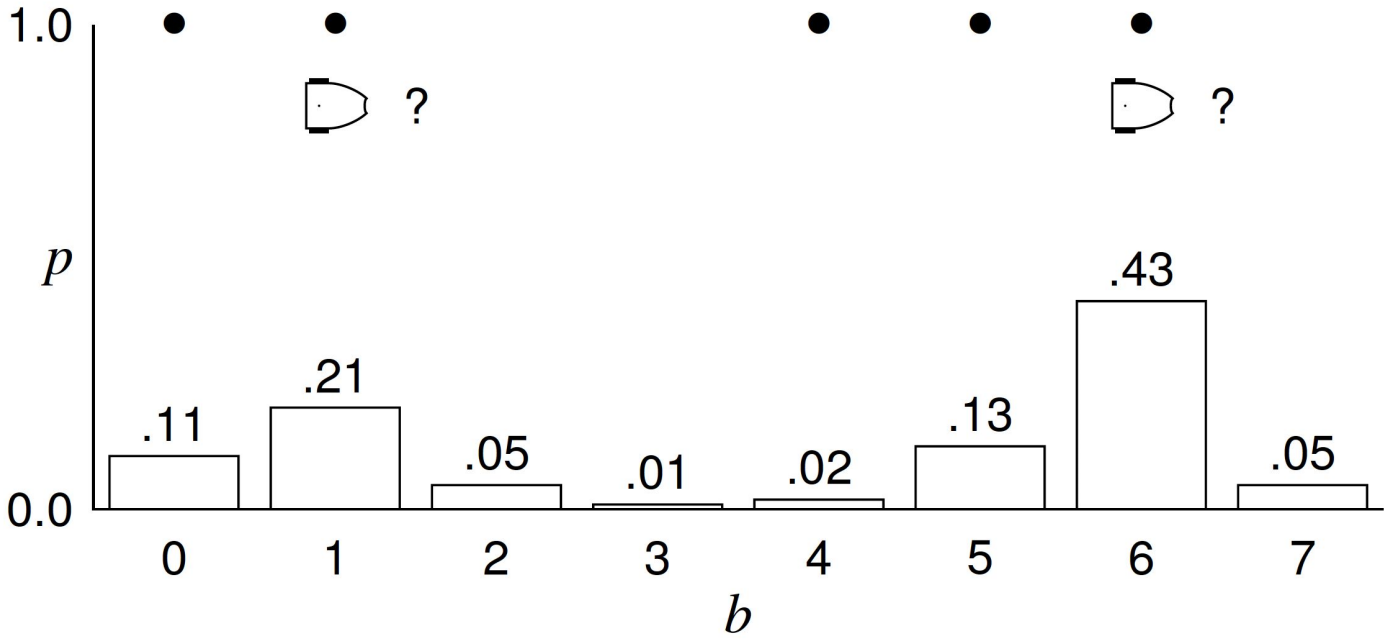
Uncertainty in Motion & Sensing

Table 8.2 Localization with uncertainty in sensing and motion
sensor=after multiplying by the sensor uncertainty
norm=after normalization
right=after moving right one position

	position	0	1	2	3	4	5	6	7
	door?	•	•			•	•	•	
	initial	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13
sensing	sensor	0.11	0.11	0.01	0.01	0.11	0.11	0.11	0.01
	norm	0.19	0.19	0.02	0.02	0.19	0.19	0.19	0.02
moving	right	0.05	0.17	0.17	0.04	0.04	0.17	0.19	0.17
sensing	sensor	0.05	0.17	0.02	0.00	0.03	0.15	0.17	0.02
	norm	0.08	0.27	0.03	0.01	0.06	0.25	0.28	0.03
moving	right	0.06	0.12	0.23	0.05	0.01	0.07	0.23	0.25
	sensor	0.05	0.10	0.02	0.01	0.01	0.06	0.21	0.02
	norm	0.11	0.21	0.05	0.01	0.02	0.13	0.43	0.05

Uncertainty in Motion & Sensing

Final state:



MOBILE ROBOTS 7/8- Uncertainties

Prof. Francesco Mondada

EPFL

Bibliography and Sources

***Elements of Robotics* (ch. 8),** M. Ben-Ari, F. Mondada, Springer, 2017.

***Springer Handbook of Robotics* (ch 5),** B. Siciliano, and O. Khatib (Eds.), 2nd edition, Springer, 2016.

Probabilistic Robotics by Thrun, Burgard, and Fox, MIT Press, 2005

Mobile Robots - EPFL - J.-C. Zufferey, Felix Schill, 2013.

Table of Contents

Motivation

How can we represent a world

- which is perceived with errors,
- on which we do actions that do not correspond exactly to our orders,
- with maps that are uncertain?

Let use probabilistic (random) variables

Conditional Probability

Random variables often carry information about other random variables. Suppose we already know that *Y* value is *y*, and we would like to know the probability that *X* value is *x* conditioned on that fact:

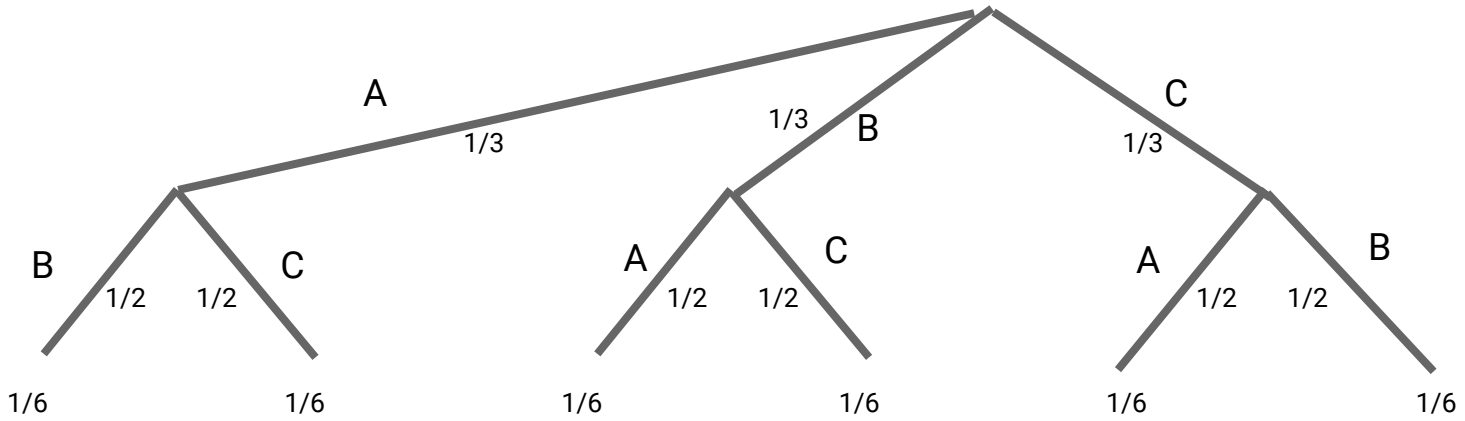
$$p(x \mid y) = \frac{p(x, y)}{p(y)}$$

Where *p(x,y)* is the probability of having *x* and *y*.

Theorem of *total probability* (often called *marginalization*):

$$p(x) = \int p(x, y)dy = \int p(x \mid y)p(y)dy$$

Conditional Probability



$$p(x \mid y) = \frac{p(x, y)}{p(y)}$$

$$p(x) = \int p(x, y)dy = \int p(x \mid y)p(y)dy$$

Bayes Rule

The Bayes rule relates a conditional of the type $p(x \mid y)$ to its "inverse" $p(y \mid x)$:

$$p(x \mid y) = \frac{p(y \mid x) \cdot p(x)}{p(y)}$$

Bayes rule plays a predominant role in probabilistic robotics (and probabilistic inference in general).

Bayes Rule

$$p(x \mid y) = \frac{p(y \mid x) \cdot p(x)}{p(y)}$$

If ***x* is the robot's state** that we would like to infer from ***y* (the sensor data)**:

- $p(x)$ is referred to as *prior probability distribution*, which summarizes the knowledge (or ignorance) we have regarding the robot state X **prior** to incorporating the sensor data y.
- $p(x \mid y)$ is called the *posterior probability distribution* over X meaning the knowledge we have regarding the robot state X **after** incorporating the sensor data y.
- $p(y \mid x)$ is often coined *likelihood* or *generative model*, since it describes how state variables X impacts on sensor measurements Y. This information can typically come from a map that gives information on the environment based on the state of the robot (position).

Bayes Rule and Normalisation

Note that the denominator of the Bayes rule, $p(y)$, the general probability to have a given sensor readings, does not depend on x. For this reason, $p(y)^{-1}$ is often seen as a normalizer:

$$p(x \mid y) = \eta \cdot p(y \mid x) \cdot p(x)$$

The posterior integral just needs to be equal to 1.

To sum up, the Bayes rule provides a convenient way to compute a posterior $p(x \mid y)$ using the "inverse" conditional probability $p(y \mid x)$ along with the prior probability $p(x)$.

Table of Contents

State variable

In probabilistic robotics, the *state x* is the collection of all aspects of the robot and its environment (both constituting a single dynamical system) that can impact the future of the robot (defined in the state).

State variables in mobile robotics typically include:

- *robot pose*: location & orientation relative to a global coordinate frame
- configuration of the robot's manipulators
- robot velocity (dynamic state)
- sensor status or parameters (e.g. inertial sensor biases)
- location and properties of surrounding objects in the environment
- location and velocities of moving objects in the environment
- etc. (the list is endless!)

In this chapter, we assume a static world and focus on estimating only the pose of a kinematic mobile robot (without dynamic objects, nor changing sensor parameters).

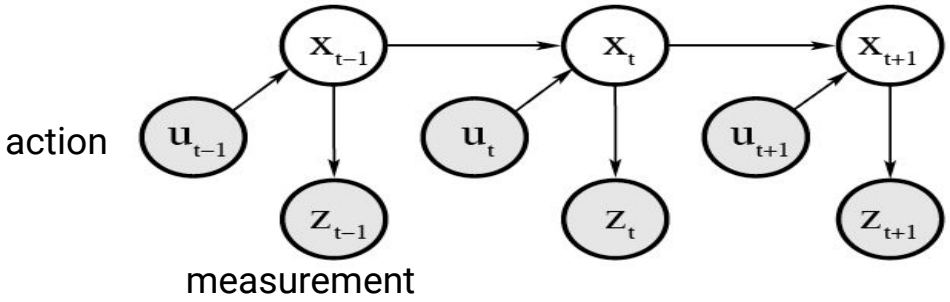
Complete State and Markov Chain

A state x_t is said to be *complete* if it fully captures all the information that could influence its future evolution.

Completeness entails that knowledge of past states, measurements, or controls carry no additional information that would help us predict the future more accurately.

Complete State and Markov Chain

A *Markov chain* is a temporal process that meets this condition of state completeness. A Markov chain describes at successive times the (complete) state of a system.



Note: The notion of state completeness is mostly of theoretical importance (i.e. required to derive the Bayes filter). In practice, it is impossible to specify a complete state for any realistic robot system. A complete state includes not just all aspects of the environment that may have an impact on the future, but also the robot itself, the content of its computer memory, the brain dumps of surrounding people, etc.

Types of Robot–Environment Interactions

Environment sensor **measurements** (=observation, percept)

- Process by which the robot uses its sensors to obtain information about the state of its environment.
- Environment *measurement data* will be denoted z_t where

$$z_{t1:t2} = z_{t1}, z_{t1+1}, z_{t1+2}, \ldots, z_{t2}$$

denotes the set of all measurements acquired from time t_1 to t_2.

Types of Robot–Environment Interactions

Control actions (or motion)

- Process by which the robot changes the state of the world by actively asserting forces on the robot's environment.
- *Control data* u_t carry information about the *change of state*. Typical control data include: velocity of robot actuators or odometer data (!)
- As before, a sequence of control data will be denoted:

$$u_{t1:t2} = u_{t1}, u_{t1+1}, u_{t1+2}, \ldots, u_{t2}$$

Types of Robot–Environment Interactions

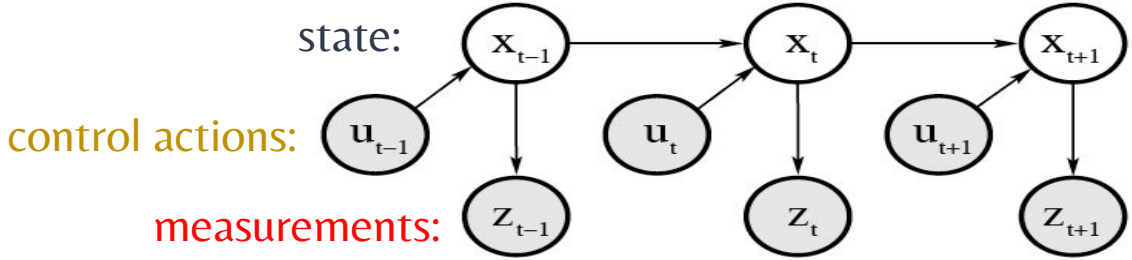
The distinction between measurement and control is a crucial one, as both types of data play fundamentally different roles in the material yet to come.

- **Perception** provides information about the environment's state, hence it **tends to increase the robot's knowledge**.

- **Motion**, on the other hand, **tends to induce a loss of knowledge** due to the inherent noise in robot actuation and the stochasticity of robot environments; (although sometimes a control makes the robot more certain about the state.)

Evolution of State and Measurement

The evolution of state and measurements is governed by probabilistic laws:



The *state transition probability* is the probabilistic law characterizing the evolution of state:

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) \quad = \quad p(x_t \mid x_{t-1}, u_t)$$

The *measurement probability* is the process by which measurements are generated:

$$p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) \quad = \quad p(z_t \mid x_t)$$

Such a generative model is also known as dynamic Bayes network (DBN), which belongs to the class of **hidden** Markov model (HMM).

Belief Distribution

A *belief* reflects the robot's internal knowledge about the state, *which cannot be measured directly (hidden state)*.

A belief distribution assigns a probability to each possible hypothesis with respect to the true state ("what is the probability to be somewhere?").

Belief distributions are *posterior probabilities* over state variables conditioned on the available data:

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t})$$

If this posterior is calculated before incorporating the latest measurements, it is referred to as *prediction* and denoted:

$$\overline{bel(x_t)} = p(x_t \mid z_{1:t-1}, u_{1:t})$$

This terminology reflects the fact that $\overline{bel(x_t)}$ predicts the state at time t based on the previous state posterior, before incorporating the measurement at time t. Calculating $bel(x_t)$ from $\overline{bel(x_t)}$ is called *correction* or *measurement update*.

Table of Contents

Bayes Filter

The most general algorithm for calculating beliefs is the *Bayes filter*. It is a recursive filter and the following pseudo-code depicts a single iteration of it:

Algorithm Bayes_filter$(bel(x_{t-1}), u_t, z_t)$**:**

for all x_t *do*

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

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

endfor

return $bel(x_t)$

prediction

measurement update

action model

measurement model

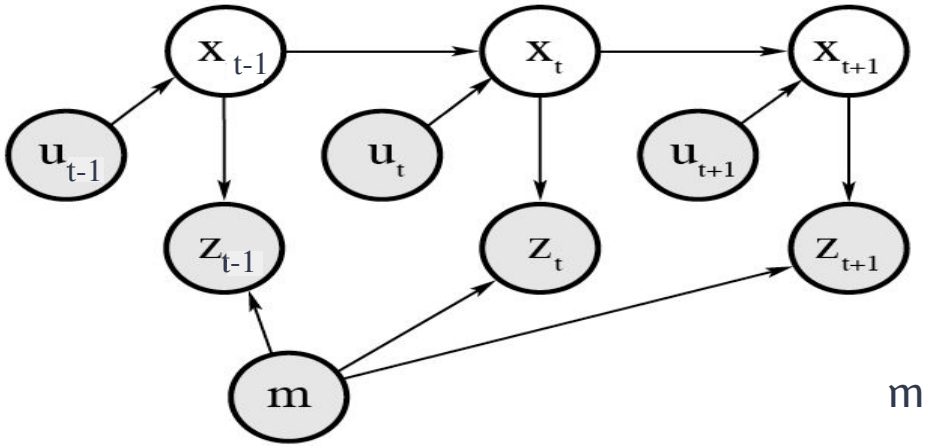
Two steps:

1. *Control update* or *prediction* (based on the theorem of total probability).
2. *Measurement update* or *correction* (based on the Bayes rule).

This algorithm can only be implemented in the form stated here for very simple estimation problems. One either needs to be able to carry out the integration in step 1 and the multiplication in step 2 in closed form, or one needs to restrict himself to finite state spaces, so that the integral becomes a finite sum.

Localisation : the Bayesian Perspective

The robot is given a map m of its environment and its goal is to determine its position relative to this map given the perceptions of the environment and its movements.



state ~ robot pose

control actions

measurements

map

In probabilistic robotics, this problem is solved using a variant of the Bayes filter.

Markov Localisation

Markov localization is just a different name for the Bayes filter applied to the mobile robot localization problem:

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

for all x_t *do*

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

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

endfor

return $bel(x_t)$

motion model (not always map dependent)

prediction

measurement update

measurement model (map dependent)

Markov localization can address the global localization problem (initial belief is uniform), the position tracking problem (initial belief is typically a tight Gaussian), and the kidnapped robot problem in static environments.

1D Example

a) Initial belief is uniform over all poses (global localization).

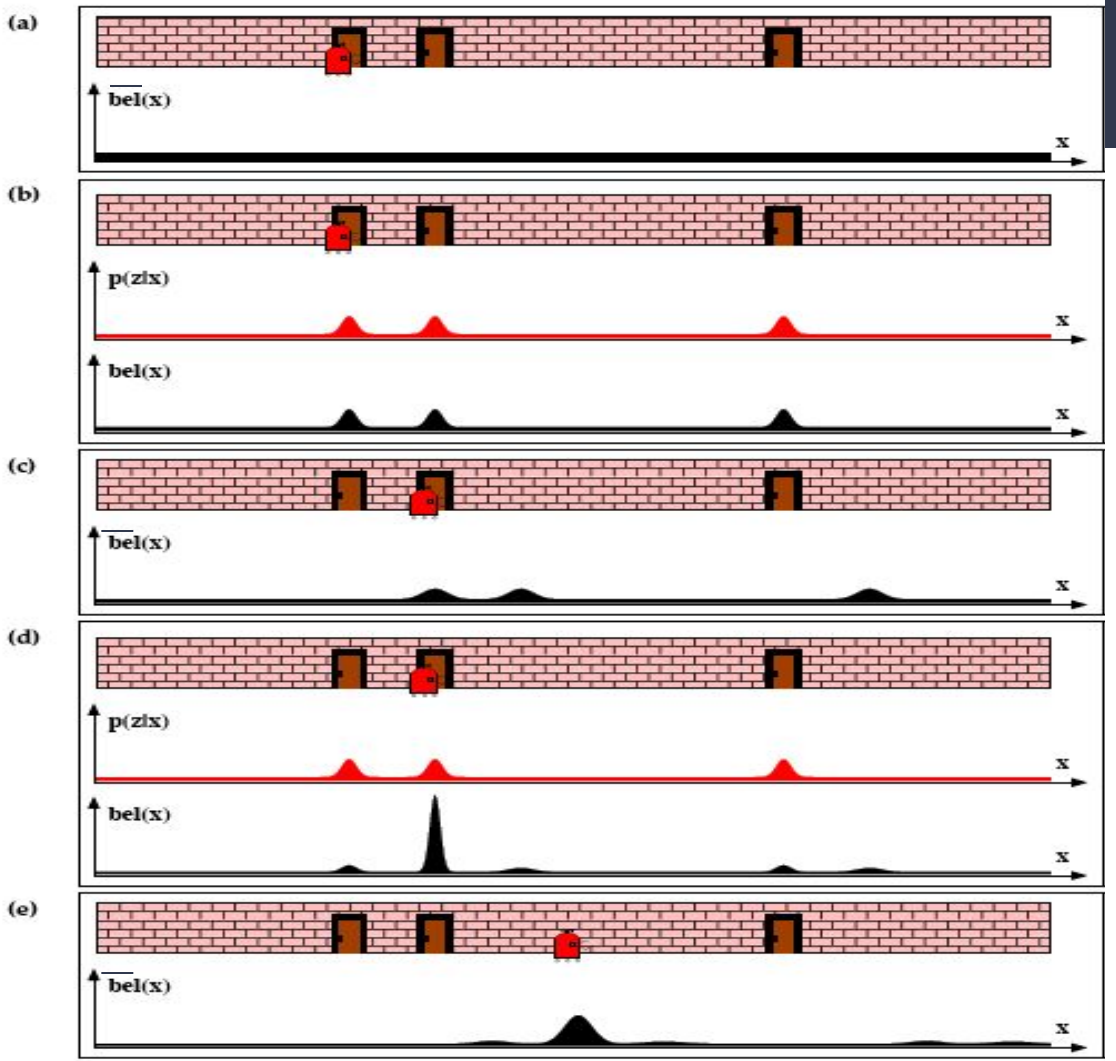
As the robot moves to the right, the 1st step of Bayes filter convolves its belief with the motion model $p(x_t \mid u_t, x_{t-1})$, not indicated here at the beginning.

b) As the robot queries its sensors and notices that it is adjacent to one of the doors, it multiplies its belief by $p(z_t \mid x_t, m)$ according to the 2nd step of the Bayes filter.

c) As the robot moves to the right, the 1st step of Bayes filter is applied again.

d) The 2nd measurement allows to correct the previous prediction. Now the robot is quite confident of having localized itself.

e) Robot belief after having moved further down the hallway (without further measurements).



Tractability Of Bayes Filter For Localisation

Since the Bayes filter is not a practical algorithm (numerical computation is feasible only for very specific cases), probabilistic algorithms for robot localization use approximations.

The nature of approximation has important ramifications on the complexity of the algorithm and the type of localization (e.g. global vs tracking).

Example of widely-used approximations:

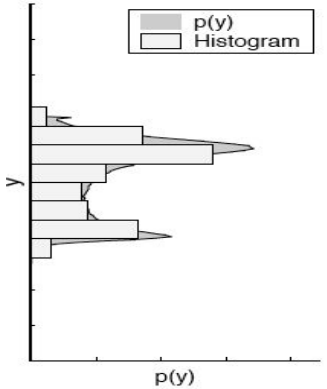
- Discretization of the belief space -> nonparametric, discrete filters
  - *Histogram filter -> grid localization:*
    - belief is discretized into an histogram (finite state spaces)
  - *Particle filter -> Monte Carlo Localization (MCL):*
    - represents the belief by a set of random state samples drawn from the belief
- Linearization and parameterization -> Gaussian filters
  - *Extended Kalman filter -> EKF localization:*
    - belief is represented using Gaussian(s)
    - motion and measurement models are linearized (using the Jacobian matrix)

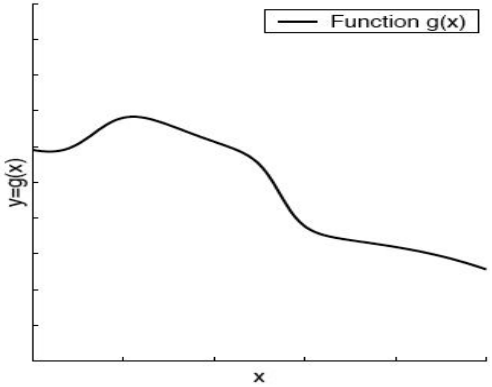
Histogram Filter (Markov Localization)

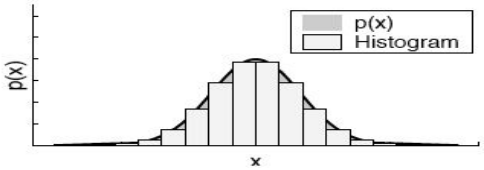
Belief is discretized into a n-dimensional histogram.

Simplest to understand.

Becomes intractable for large or high dimensional state spaces.







Algorithm Discrete_Bayes_filter$(\{p_{k,t-1}\}, u_t, z_t)$:

for all k *do*

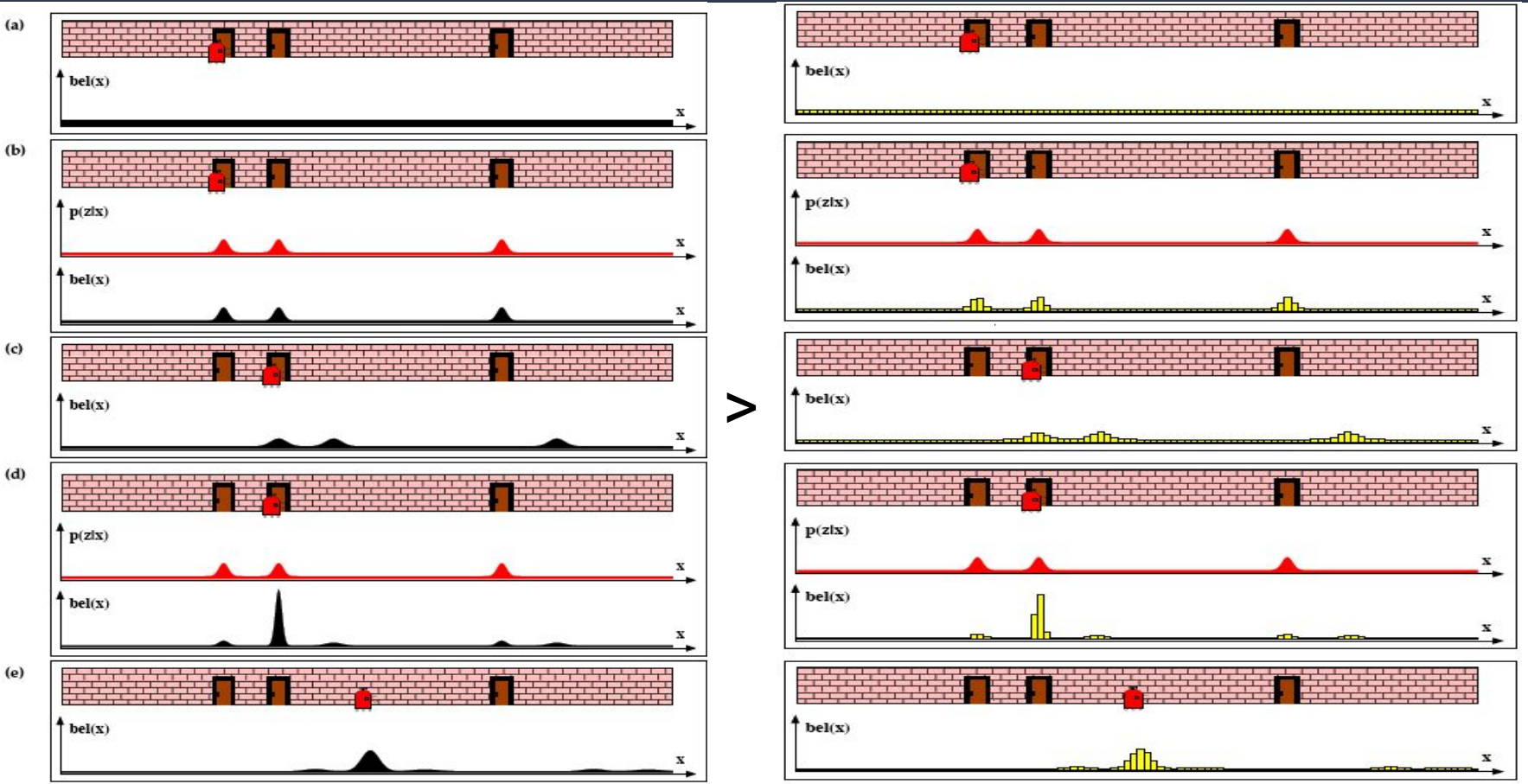
$$\bar{p}_{k,t} = \sum_i p(X_t = x_k \mid u_t, X_{t-1} = x_i)\ p_{i,t-1}$$

$$p_{k,t} = \eta\ p(z_t \mid X_t = x_k)\ \bar{p}_{k,t}$$

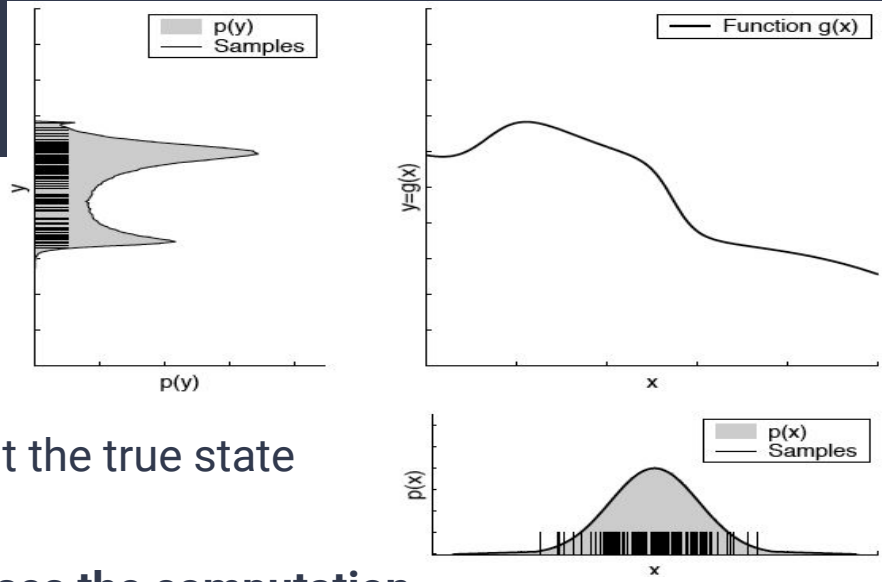
endfor

return $\{p_{k,t}\}$

1D Grid Example



Particle Filter



Particle filters represent a distribution by a set of samples.

The denser a subregion of the state space is populated by samples, the more likely it is that the true state falls into this region.

Such a representation is approximate, but focuses the computation on the most probable location, "forgetting" the locations that have very low probability.

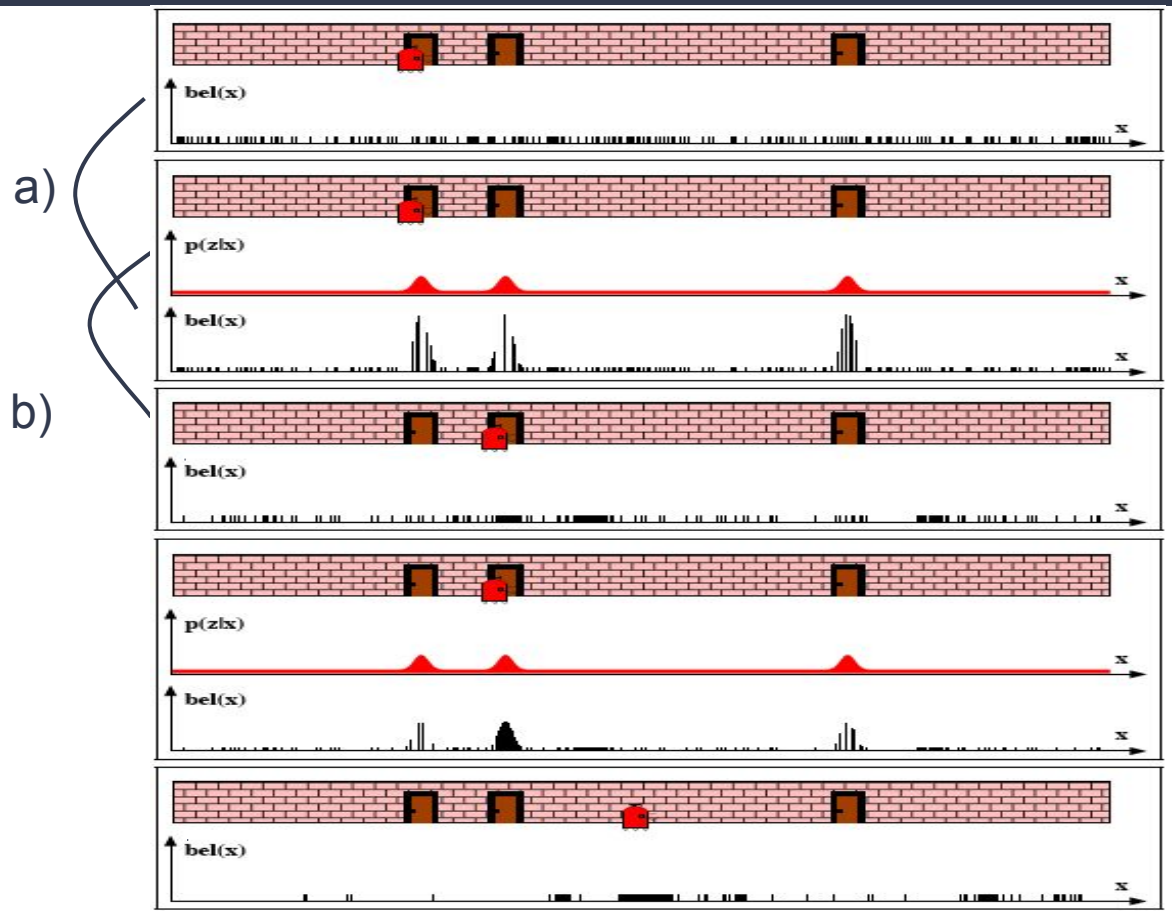
Weights are assigned to particles through the measurement model and resampling allows to redistribute particles approximately according to the posterior $bel(x_t)$.

The resampling step is a probabilistic implementation of the Darwinian idea of *survival of the fittest*: it refocuses the particle set to regions in state space with high posterior probability.

1D Particle Filter (MCL) Example

a) weighing of the particles depending on the measurement model & resampling according to the weights

b) application of the motion model



Particle Filter – Example Algorithm

In particle filters, the samples of a posterior distribution are called *particles* and are denoted:

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}$$

1: **Algorithm Particle_filter**($\mathcal{X}_{t-1}, u_t, z_t$):
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to M do
4: sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to M do
9: draw i with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

Sampling generation of new particles from the old one using the probabilistic motion model (spreading)

Evaluation of the **weight** by incorporating the measurement z_t into the particle set. The weight is thus the probability of the measurement z_t under the particle $x_t^{[m]}$. Particles that have a position that fits well with the measurement get higher weight.

Particle Filter – Example Algorithm

In particle filters In particle filters, the samples of a posterior distribution are called *particles* and are denoted:

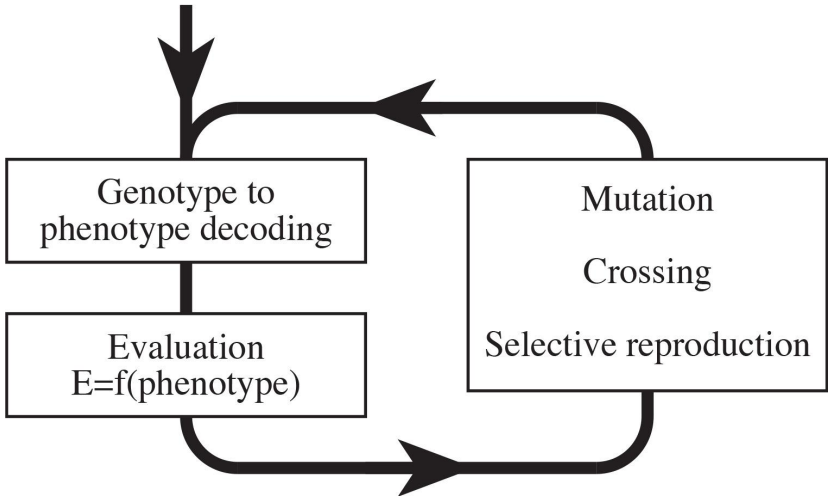
$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}$$

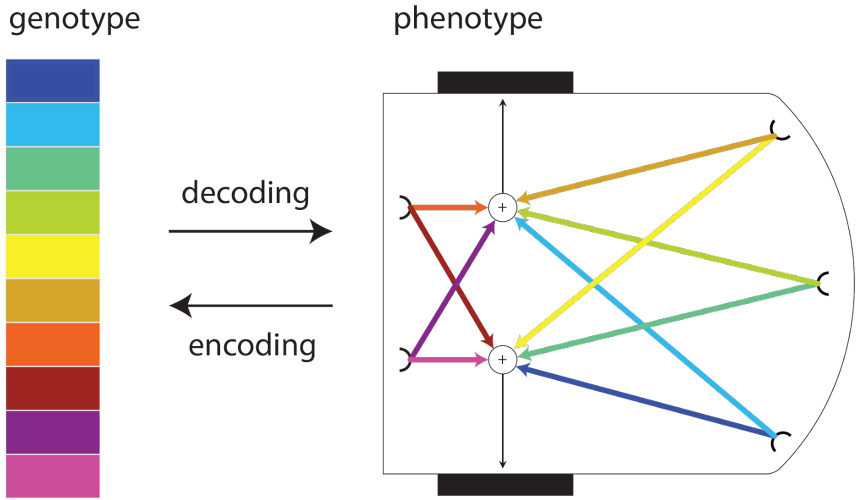
1: **Algorithm Particle_filter**($\mathcal{X}_{t-1}, u_t, z_t$):
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to M do
4: sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to M do
9: draw i with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

Resampling. The algorithms draws with replacement M particles from the temporary belief $\overline{X_t}$. The drawing probability is proportional to the weight. Whereas before the resampling step, particles were an approximation of $\overline{bel(x_t)}$, after they are an approximation of $bel(x_t)$

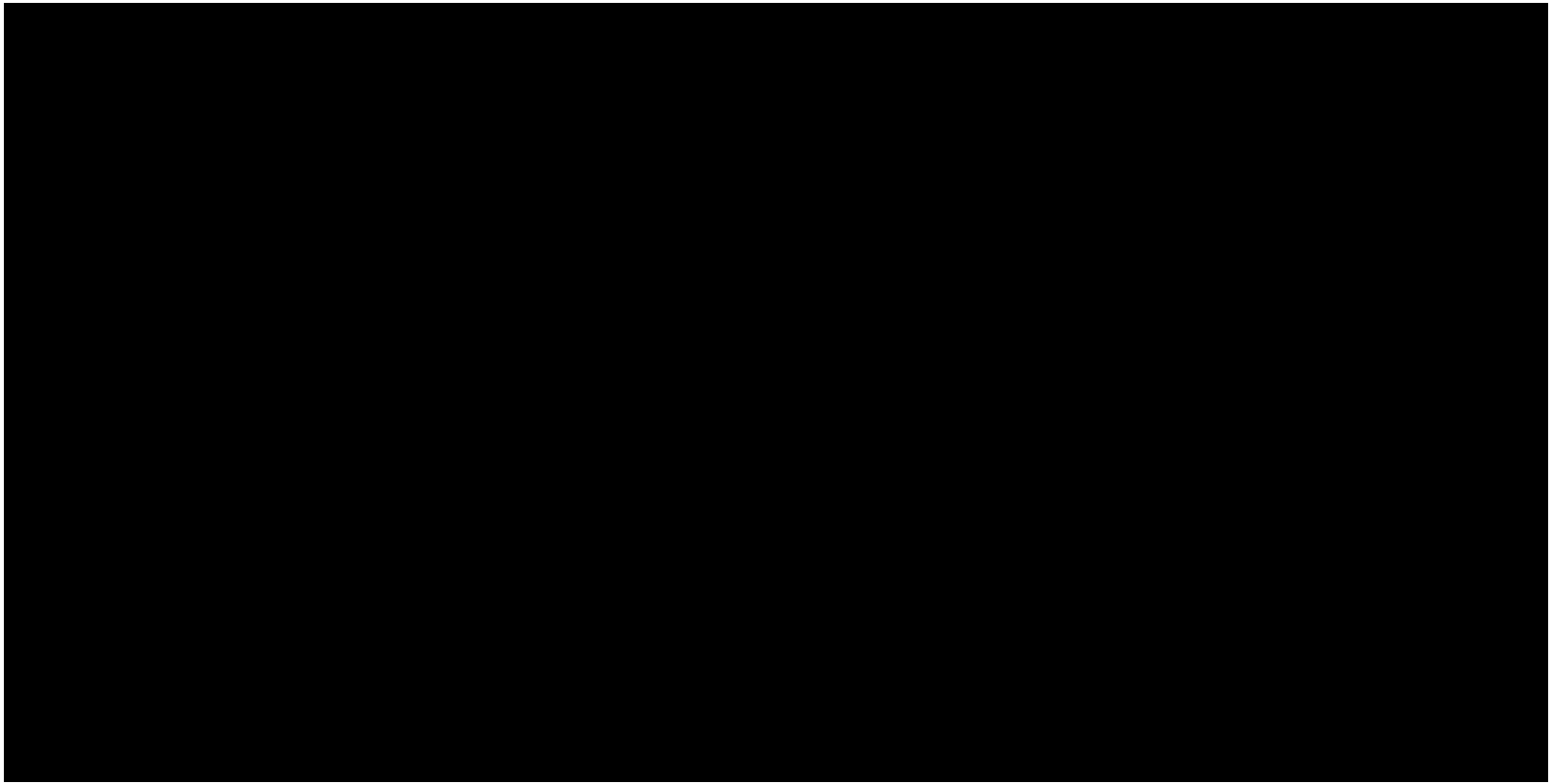
Population-based: Genetic Algorithm

The genetic algorithm (GA) is an engineering implementation of the concept of Darwinian evolution. An example for the evolution of the weights of an ANN:





2D Localisation on Thymio



Wang, S., Colas, F., Liu, M., Mondada, F., & Magnenat, S. (2018). Localization of inexpensive robots with low-bandwidth sensors. In *Distributed Autonomous Robotic Systems* (pp. 545-558). Springer, Cham.

Course Topics (reminder from week 1)

Week 1	Components of a mobile robot	**Week 8**	Uncertainties
Week 2	Vision	**Week 9**	Localisation 2 + Project week 1
Week 3	Vision & ANN & ML	**Week 10**	Project week 2
Week 4	Navigation	**Week 11**	Project week 3
Week 5	Navigation	**Week 12**	Project week 4 ~~+ Project presentations~~
Week 6	Localisation 1	**Week 13**	Project presentations
Week 7	Uncertainties	**Week 14**	~~Pr. presentations +~~ Conclusion + Dry Exam

Project phase (starting next week)

Beginning: next week

Registration of groups on moodle starting today 19:00 (176 students, 44 groups, 55 slots)

Deadline for the registration: nov 16, 17:00

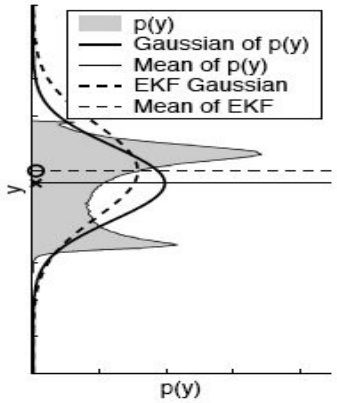
Deadline for project submission: Sunday December 12, 23:00

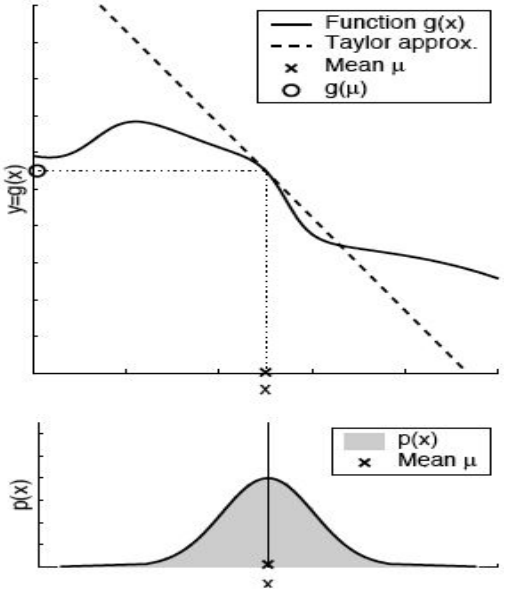
Project phase (starting next week)

A	B	C	D	E	F
	Monday December 13th on zoom	Tuesday December 14th on campus (MEA331)	Wednesday December 15th on zoom	Thursday December 16th on zoom	Friday December 17th on zoom
8:30 - 9:00		1	1		
9:00 - 9:30	1		1		1
9:30-10:00	1		1		1
10:00-10:30	1		1		1
10:30-11:00					
11:00-11:30	1		1		1
11:30-12:00	1		1		1
12:00-12:30	1		1		1
12:30-13:00	1	1	1		1
13:00-13:30		1			
13:30-14:00		1			
14:00-14:30	1		1		1
14:30-15:00	1	1	1		1
15:00-15:30	1	1	1		1
15:30-16:00	1	1	1		1
16:00-16:30		1		1	
16:30-17:00	1		1	1	1
17:00-17:30	1		1	1	1
17:30-18:00	1		1	1	1
	Give in each **green** cell the names of the 4 group members				

Kalman Filters

Kalman filters represent the belief by means of multinormals, i.e.mean vector and covariance matrix.





This approach allows to fuse information from several sensors in an “intelligent way”, by combining their statistical properties.

The aim of the Kalman filters is to minimize the a posteriori (after sensor integration) error covariance.

A Simple Scalar Example

Let consider a model of the position of a robot moving in 1D at constant speed. The system model is the following:

$$x_{t+1} = x_t + d + w_t$$

$$y_t = x_t + v_t$$

Where

x_t is the position at time t,
d is the displacement between two steps,
w_t is the displacement noise with variance q,
y_t the measured position,
v_t the measurement noise of variance r.

A Simple Scalar Example

Let compute the **prediction** of the mean, based on the previous mean:

$$\bar{\mu}_t = \mu_{t-1} + d$$

… and compute the **prediction** of the error variance:

$$\bar{\Sigma}_t = \Sigma_{t-1} + q$$

A Simple Scalar Example

If we now take a **measurement** we can have a look to the difference between measure and prediction, called *innovation*:

$$i_t = y_t - \bar{\mu}_t$$

… and compute an optimal gain for the correction (not developed here):

$$K_t = \frac{\bar{\Sigma}_t}{\bar{\Sigma}_t + r}$$

A Simple Scalar Example

Based on the optimal gain, we can generate an estimation *a posteriori* that takes into account the **measurement** through the innovation and the gain:

$$\mu_t = \bar{\mu}_t + K_t i_t$$

… and compute the corresponding variance:

$$\Sigma_t = (1 - K_t)\bar{\Sigma}_t$$

Towards a Generic Kalman

Let consider a model of the state of a robot having two variables, a and b, components of the vector x :

$$x_{t+1} = Ax_t + Bu_t + w_t$$

$$y_t = Cx_t + v_t$$

Where x_k is the state of the system at sample k,
A is the matrix defining how the system evolves,
u is the known input to the system, impacting x through B,
w_k is the state stochastic perturbation with covariance matrix Q,
y_k the measurement, related to the state by matrix C,
v_k the measurement noise with covariance matrix R.

Towards a Generic Kalman

As an example we could consider a simple system with no external input:

$$x_{t+1} = Ax_t + w_t$$

$$y_t = Cx_t + v_t$$

Where:

$$A = \begin{bmatrix} 1 & -0.9 \\ 1 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad Q = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \quad R = \begin{bmatrix} 0.1 \end{bmatrix}$$

Towards a Generic Kalman

Let compute the expectation of the **predicted** state, based on the previous mean:

$$\bar{\mu}_t = A\mu_{t-1} + Bu_{t-1}$$

… and compute the covariance matrix of the **predicted** state:

$$\bar{\Sigma}_t = A\Sigma_{t-1}A^T + Q$$

Towards a Generic Kalman

If we now take a **measurement** we can have a look to the difference between measure and prediction, called *innovation* (and its variance):

$$i_t = y_t - C\bar{\mu}_t \qquad S_t = C\bar{\Sigma}_t C^T + R$$

… and compute an optimal gain for the correction (not developed here):

$$K_t = \bar{\Sigma}_t C^T S_t^{-1}$$

Towards a Generic Kalman

Based on the optimal gain, we can generate an estimation *a posteriori* that takes into account the **measurement** through the innovation and the gain:

$$\mu_t = \bar{\mu}_t + K_t i_t$$

… and compute the corresponding the covariance matrix:

$$\Sigma_t = (I - KC)\bar{\Sigma}_t$$

Extended Kalman Filter (EKF)

The Extended Kalman Filter is a Kalman filter applied to a non-linear

system linearized at each discrete time *k*

Algorithm Extended_Kalman_filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

$$\bar{\mu}_t = g(u_t, \mu_{t-1})$$
$$\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$$
$$K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$
$$\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$$
$$\textbf{return } \mu_t, \Sigma_t$$

prediction: computation of mean and covariance using linearization of the motion model *g*

measurement update: computation of final mean and covariance using Kalman gain K_t

g is the motion model : $x_t = g(u_t, x_{t-1}) + \varepsilon_t$ where $\varepsilon_t \sim N(0, R)$ is a multinormal (cf. ch. 4) that models the uncertainty introduced by the state transition.

G is the Jacobian (cf. ch. 4) of the motion model *g*.

Extended Kalman Filter (EKF)

The Extended Kalman Filter is a Kalman filter applied to a non-linear

system linearized at each discrete time *k*

Algorithm Extended_Kalman_filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

$$\bar{\mu}_t = g(u_t, \mu_{t-1})$$
$$\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$$
$$K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$
$$\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$$
$$\textbf{return } \mu_t, \Sigma_t$$

prediction: computation of mean and covariance using linearization of the motion model g

measurement update: computation of final mean and covariance using Kalman gain K_t

h is the measurement model: $z_t = h(x_t) + \delta_t$ where $\delta_t \sim N(0, Q)$ is a multinormal describing the measurement noise.

H is the Jacobian of the measurement model *h*.

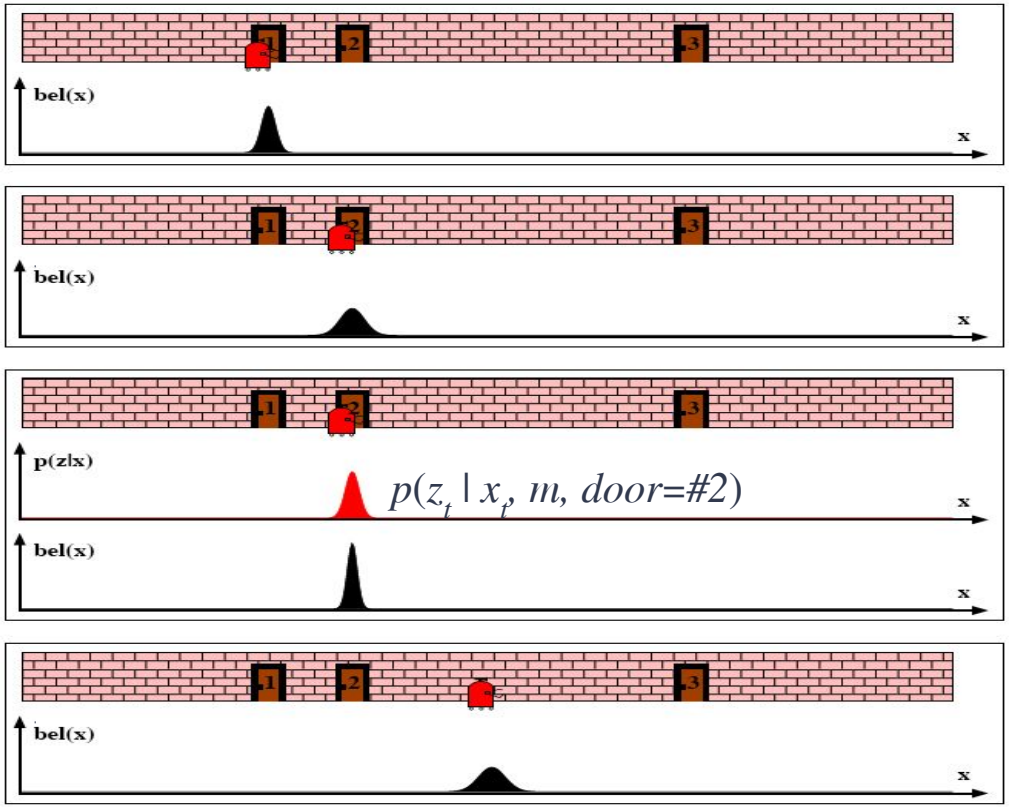
K is the *Kalman gain*. It specifies the degree to which the measurement is incorporated into the new state estimate. It is computed so to minimize the a posteriori error covariance.

1D EKF Example

Initial belief.

Initial belief is convolved with the motion model (prediction step).

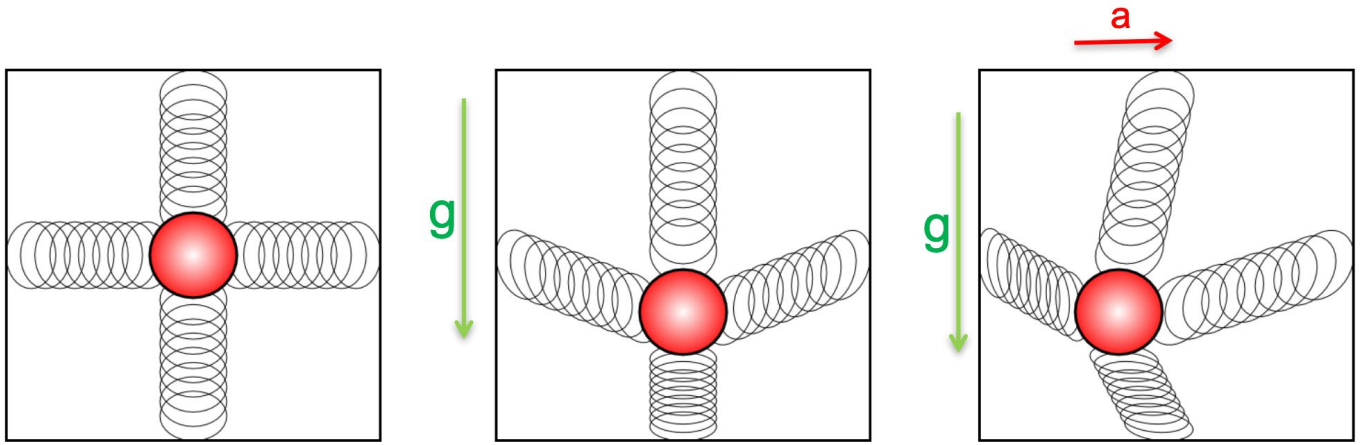
Here we assume that the robot can identify the feature it sees as door #2 (known correspondence).
Resulting belief is tighter than the variances of both the robot's previous belief and the observation density



Concrete example sensor fusion (from Adrien Briod)

Accelerometers

- Measure linear accelerations (including g!)



Free fall

Gravity

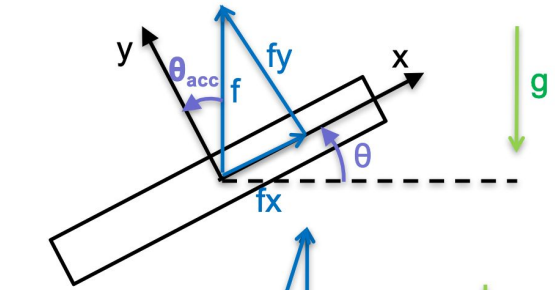
Gravity & linear accelerations

Concrete example sensor fusion (from Adrien Briod)

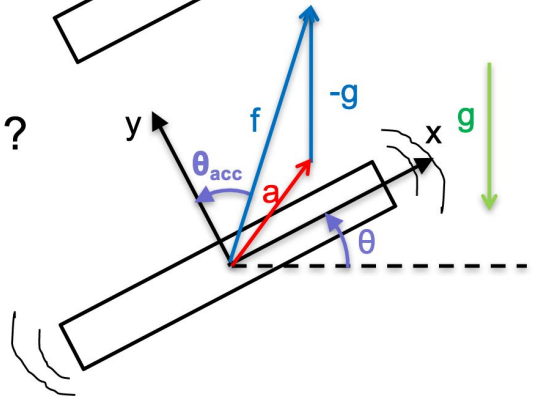
Orientation from accelerometers

- At constant speed, the accelerometers only measure gravity **g**
  → Measurement **f** = **-g**

Θ_{acc} = atan2(**fx**, **fy**)



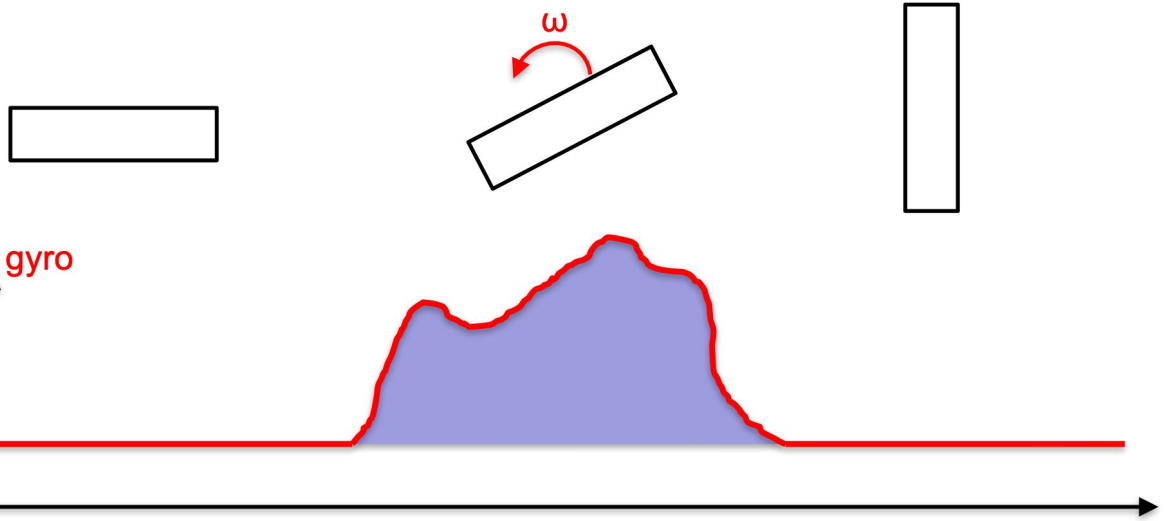
- What if additional accelerations **a** ?
  → Measurement **f** = **-g** + **a**
  → Θ_{acc} is perturbed by **a**



Concrete example sensor fusion (from Adrien Briod)

Rate gyroscopes

- Measure rotation speed (or angular velocity)
  - Pure self-motion. No absolute information.



Concrete example sensor fusion (from Adrien Briod)

Orientation from gyroscopes

- Integration of angular speed (ω)

$$\Theta = \int \omega \, dt$$

→ Θ can be computed incrementally as follows :

$$\Theta_k = \Theta_{k-1} + \omega \, \Delta t$$

- In reality, ω is noisy :

$$\omega = \omega_{true} + r$$

r: gyroscope noise

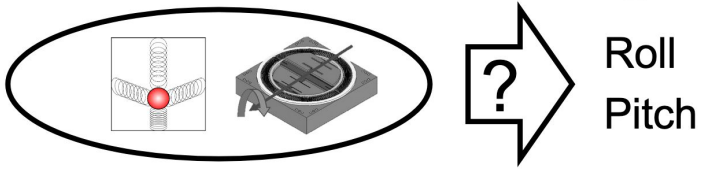
→ The angle error is increasing over time
(e.g: if constant error r=1˚/s → angle error is 60˚ after 1 minute)

Concrete example sensor fusion (from Adrien Briod)

The problem

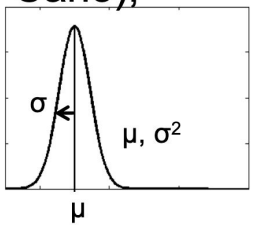
- Accelerometers or rate gyroscopes alone don't provide an accurate orientation

  → How to fuse both sensors to obtain a good estimation ?



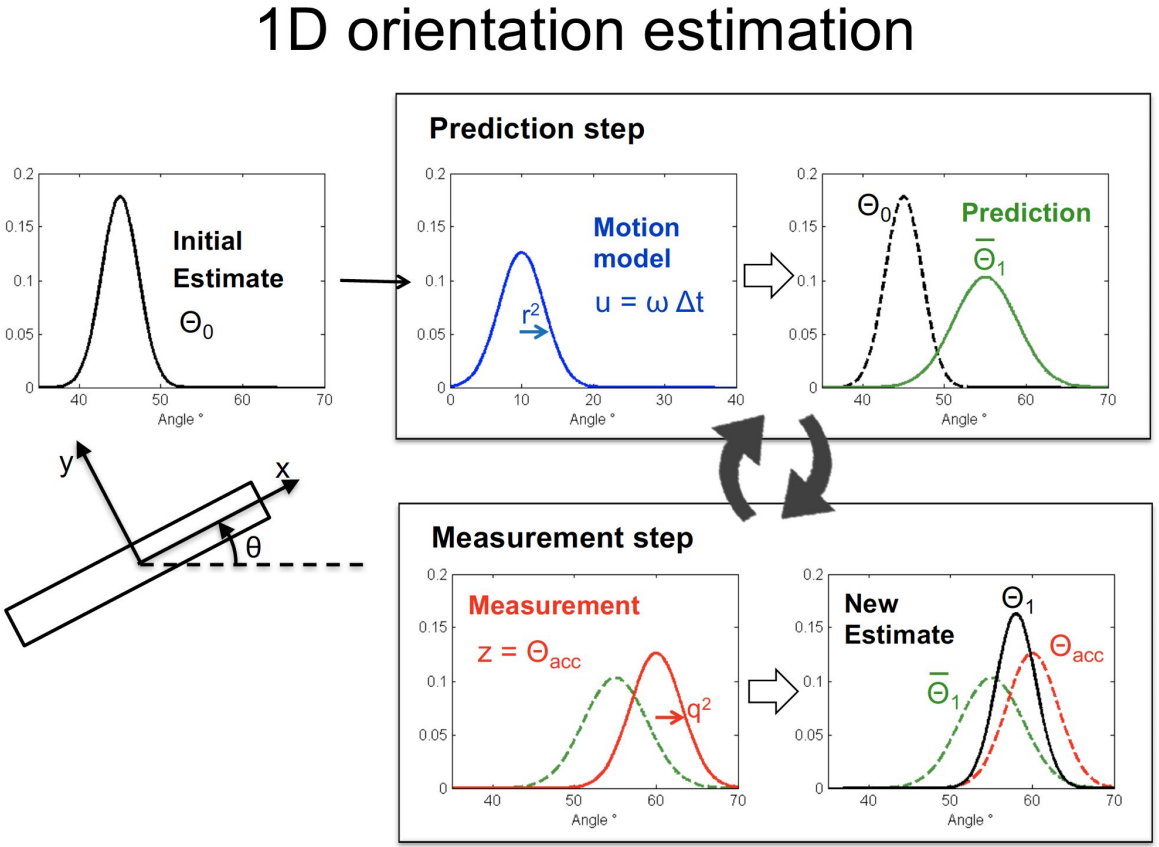
- Answer: Sensor Fusion algorithm
  - Bayes, Histogram (Grid), Particle (Monte-Carlo), Kalman,… which one ?

→ An Extended Kalman filter



  - Unimodal probability distribution (Gaussian)
    - → Advantage: Computationally efficient
    - → Drawback: Needs correct initialization, can make only one hypothesis, can diverge

Concrete example sensor fusion (from Adrien Briod)



MOBILE ROBOTS 9- SLAM

Prof. Francesco Mondada
Laila El-Hamamsy

EPFL

Table of Contents

Simultaneous Localisation and Mapping (SLAM)

SLAM

Can a robot placed at an unknown location in an unknown environment incrementally build a consistent map of this environment while simultaneously determining its location within the map?

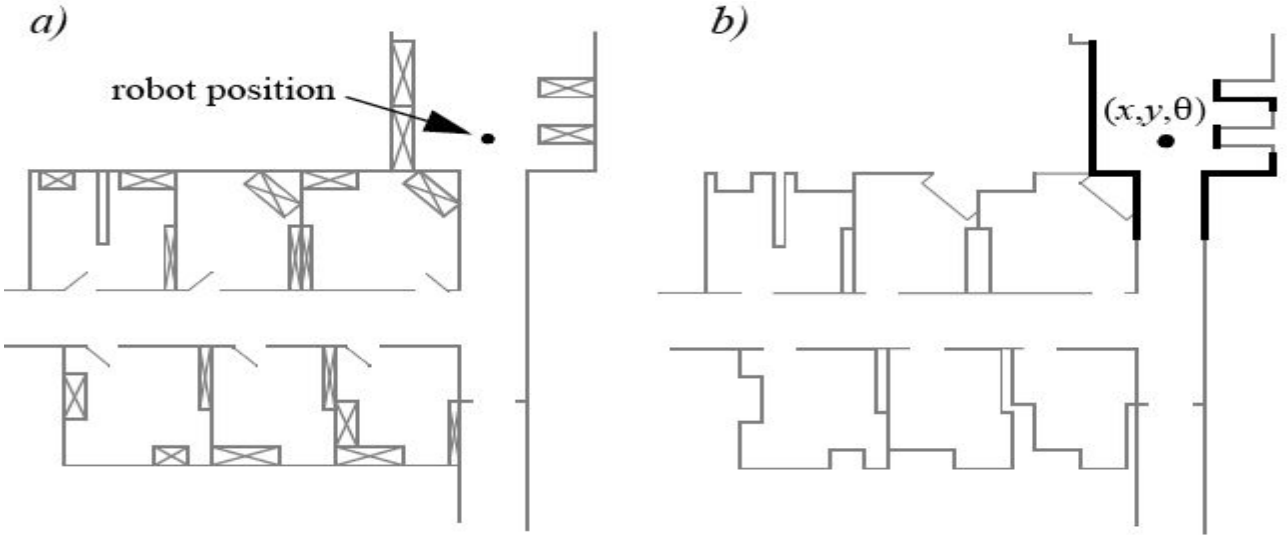
Sensors?

- Lidar
- 2D Cameras
- Depth cameras

Map Types

a) Real map with walls, doors and furniture.

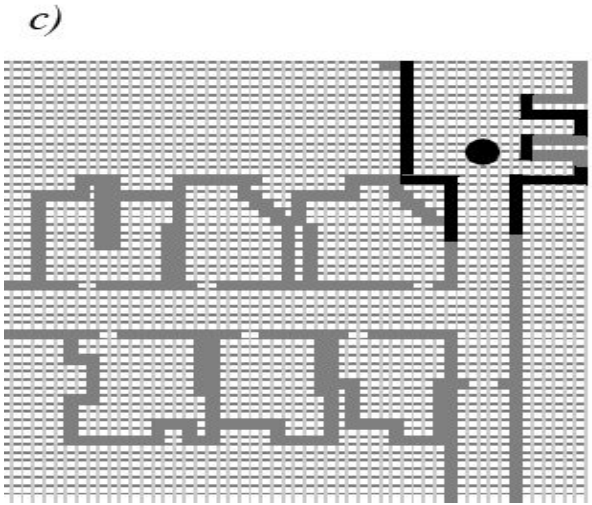
b) Line-based map: example ~100 lines with two parameters.

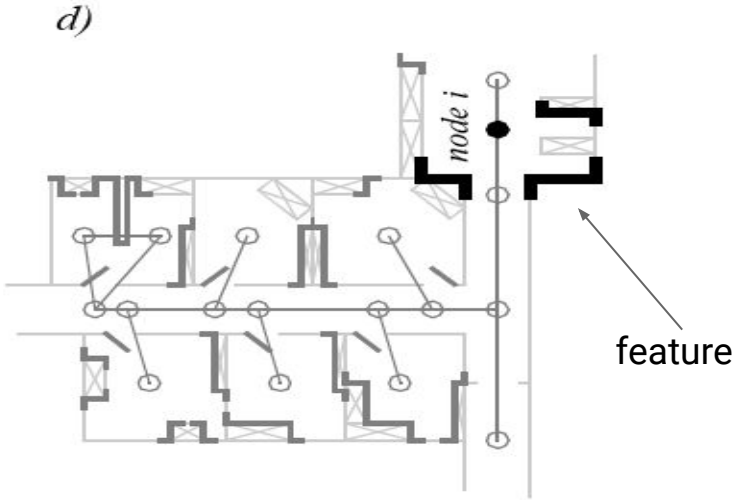


Map Types

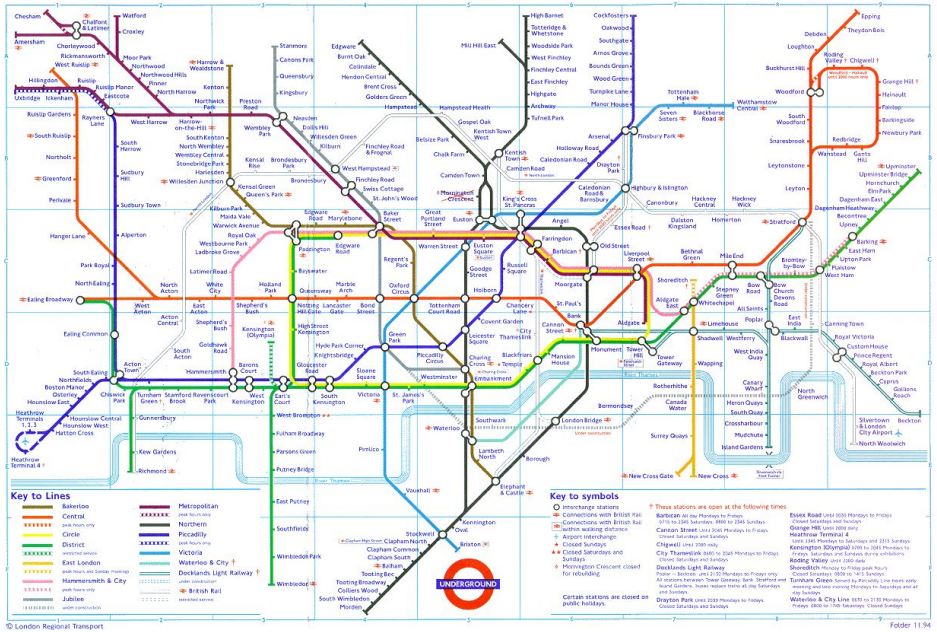
c) Occupancy grid map: example below ~3000 grid cells of 50 x 50cm.

d) Topological map e.g. using line features and doors: example ~18 nodes and 50 features.

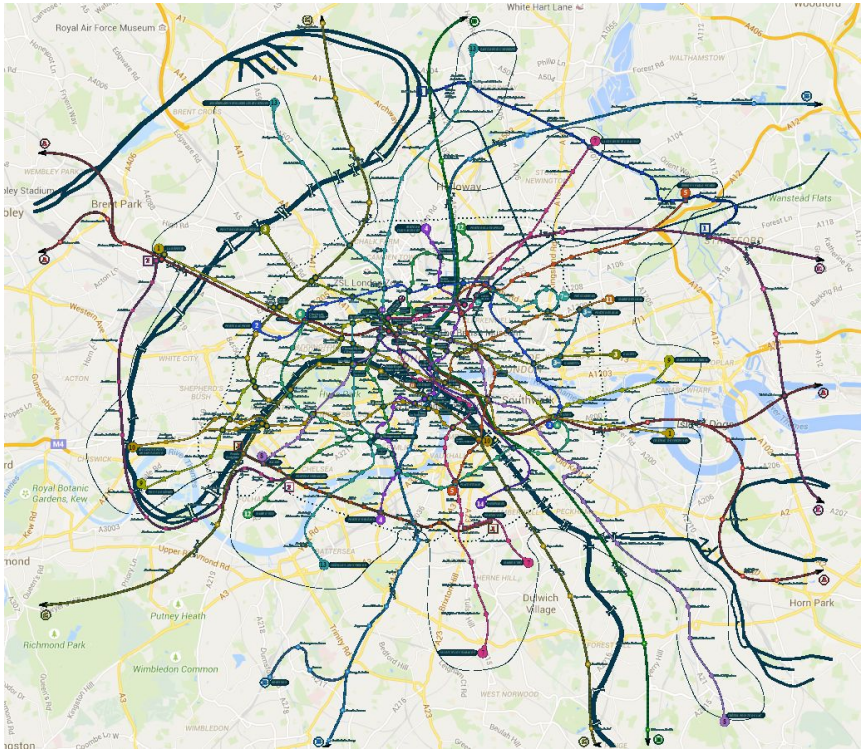




Maps – Example : London Metro



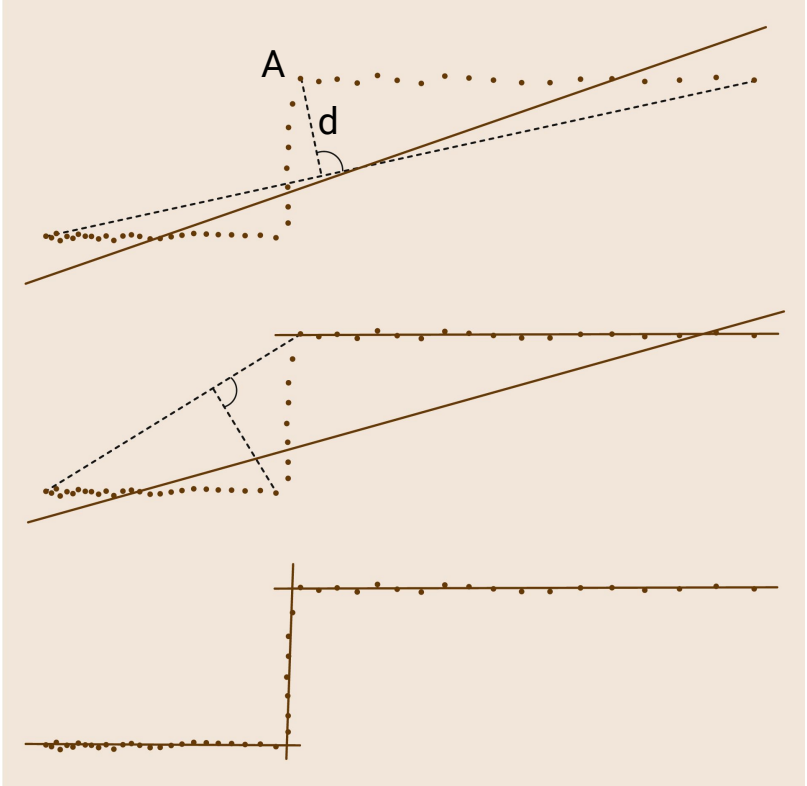
Topological map



Geographic map

Map Creation– E.g. Split & Merge Technique

1. Consider all measurements points
2. Approximate the points with a line
3. To evaluate the quality of this approximation, connect the first to the last point (dotted line) and find the point (A) that has the largest distance (d) to this dotted line.
4. If this distance (d) is larger than a given threshold, use the point (A) to separate the group of points into two clusters
5. Reapply recursively the approach to each cluster of points restarting from item 2, until the distance (d) is below the threshold.



Handbook of robotics, Springer

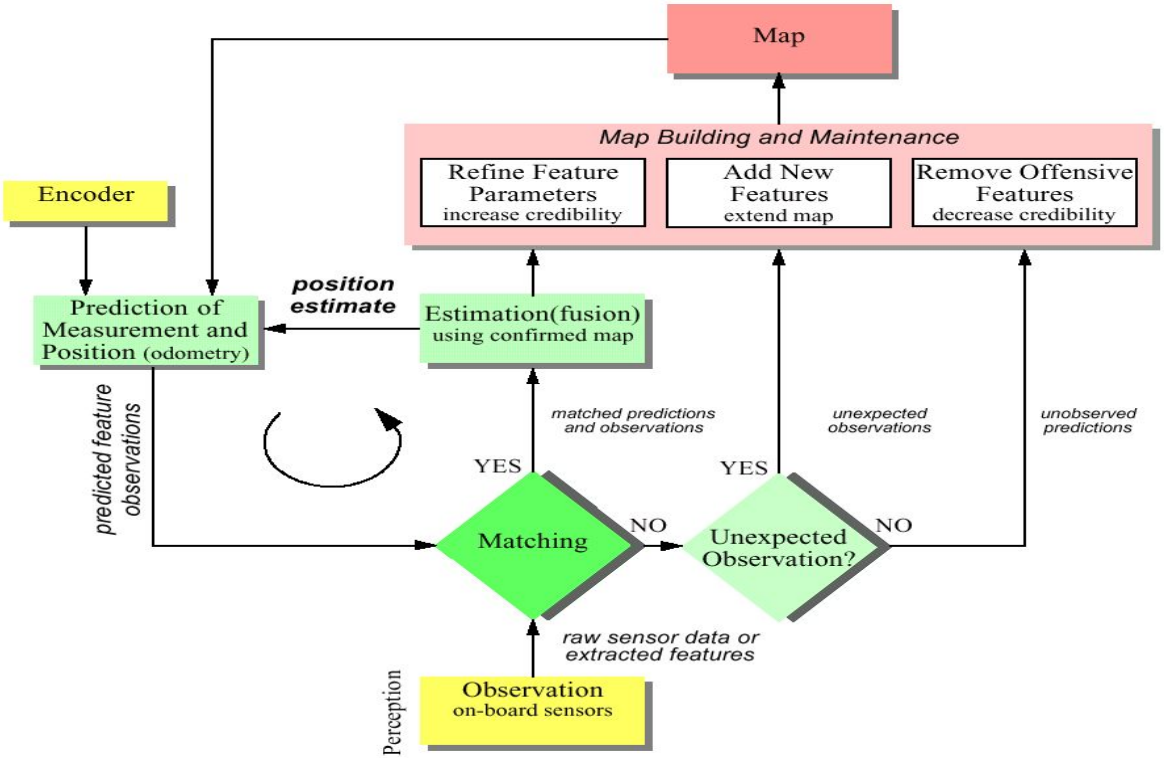
SLAM – Map Building and Maintenance

Unexpected observations will effect the creation of new features in the map.

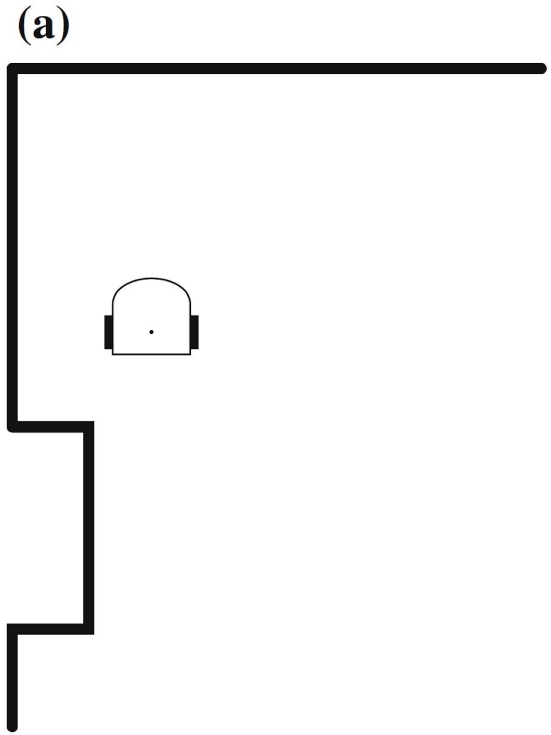
Unobserved measurement predictions may induce the removal of features from the map.

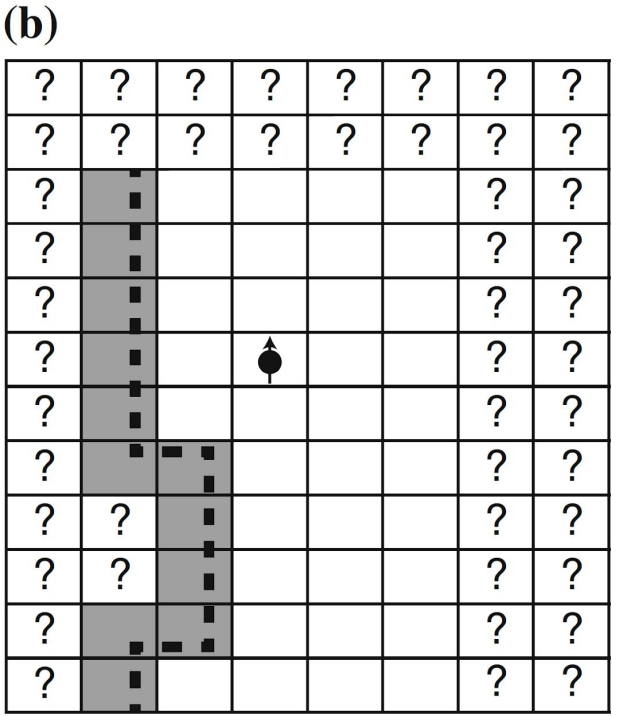
Each feature has varying degrees of probability that it is indeed part of the environment.

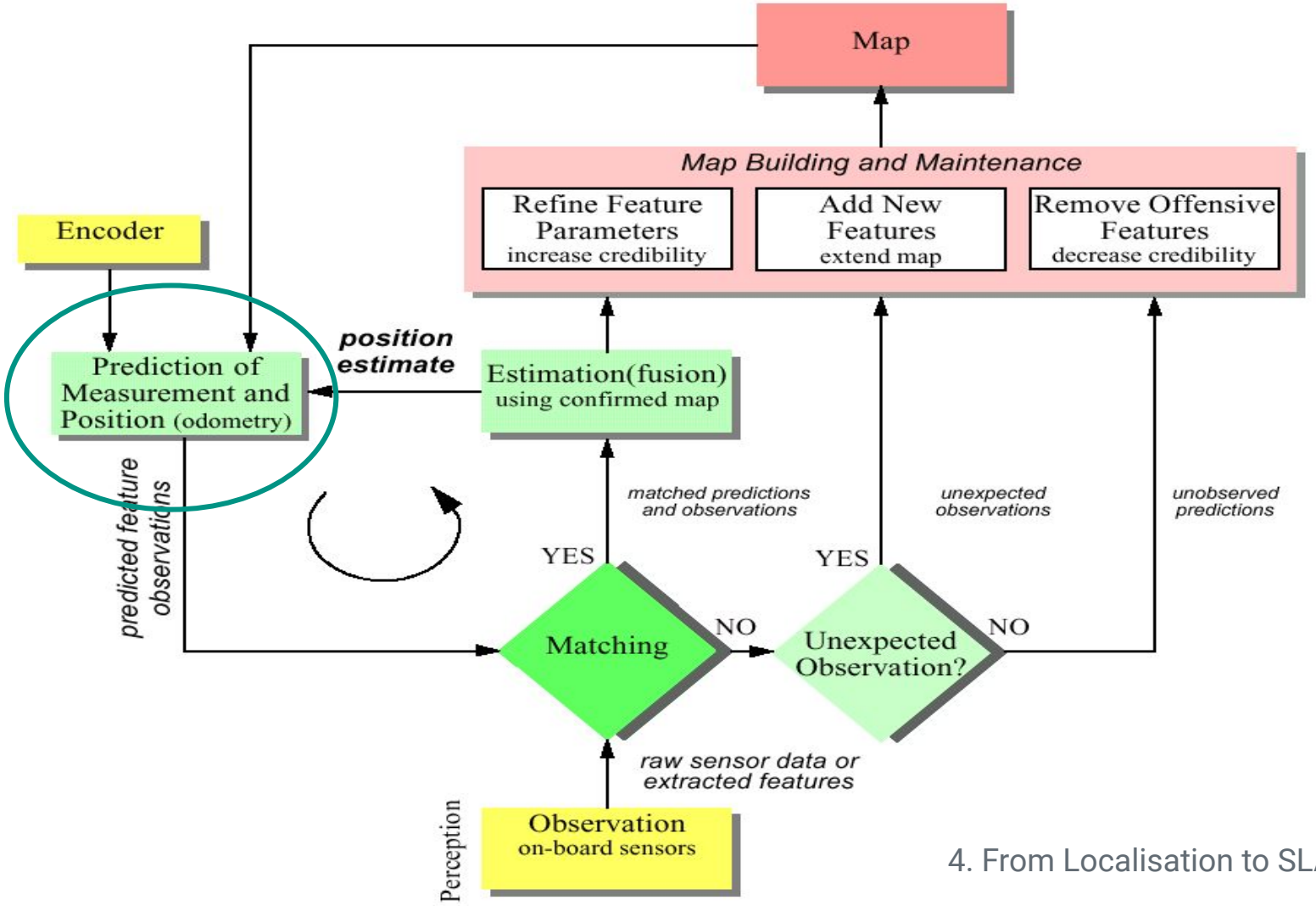
This forms a stochastic map represented by the state vector and a **covariance matrix** containing all cross-correlations, which must be updated at each cycle.



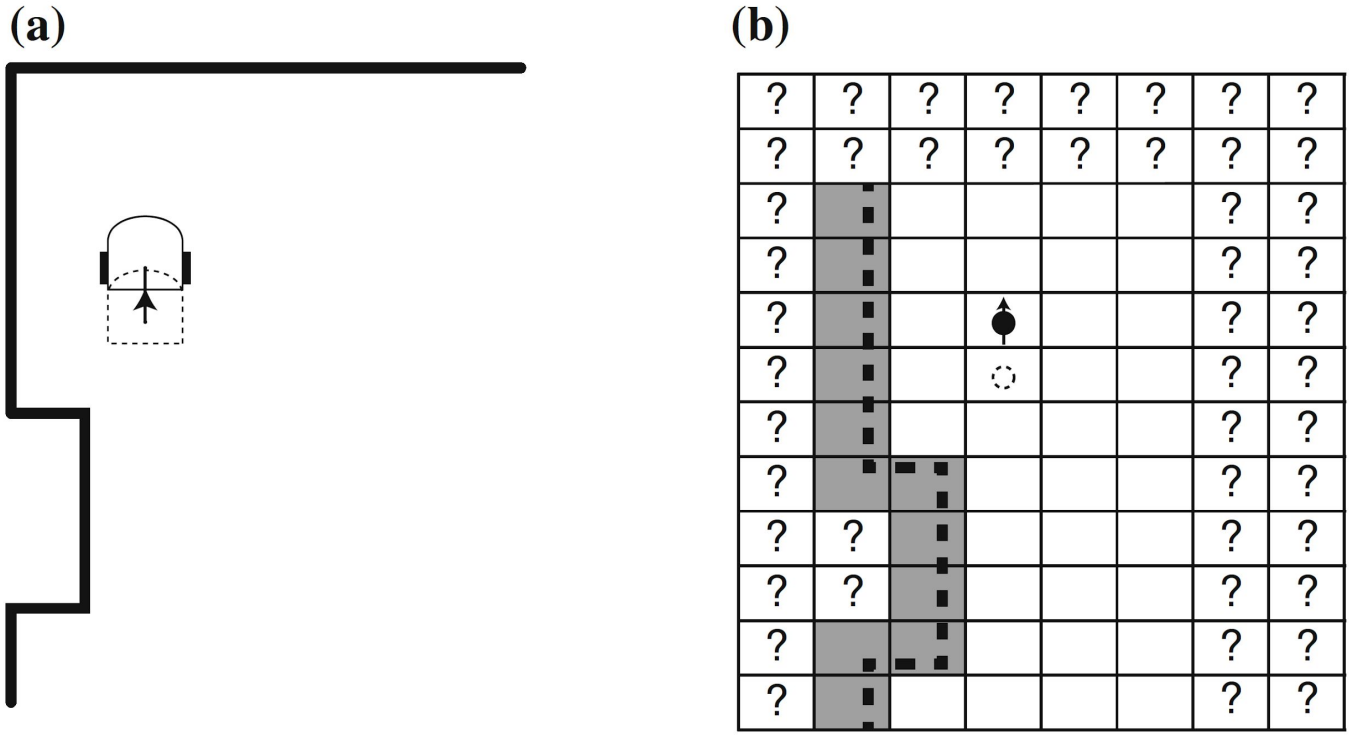
Example : Situation



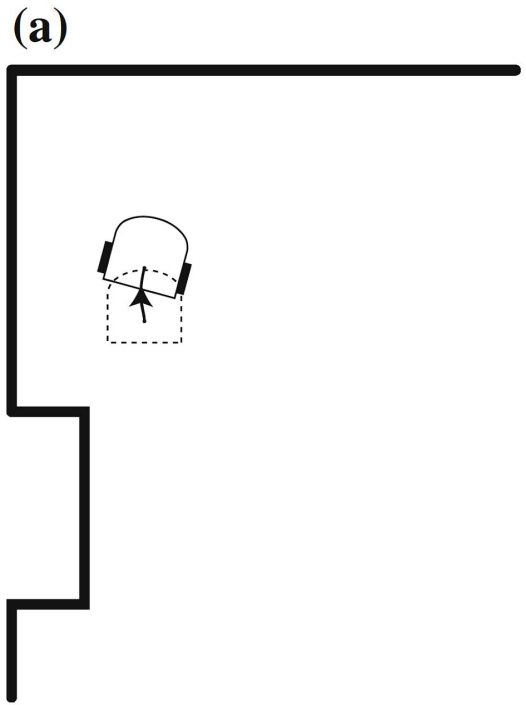


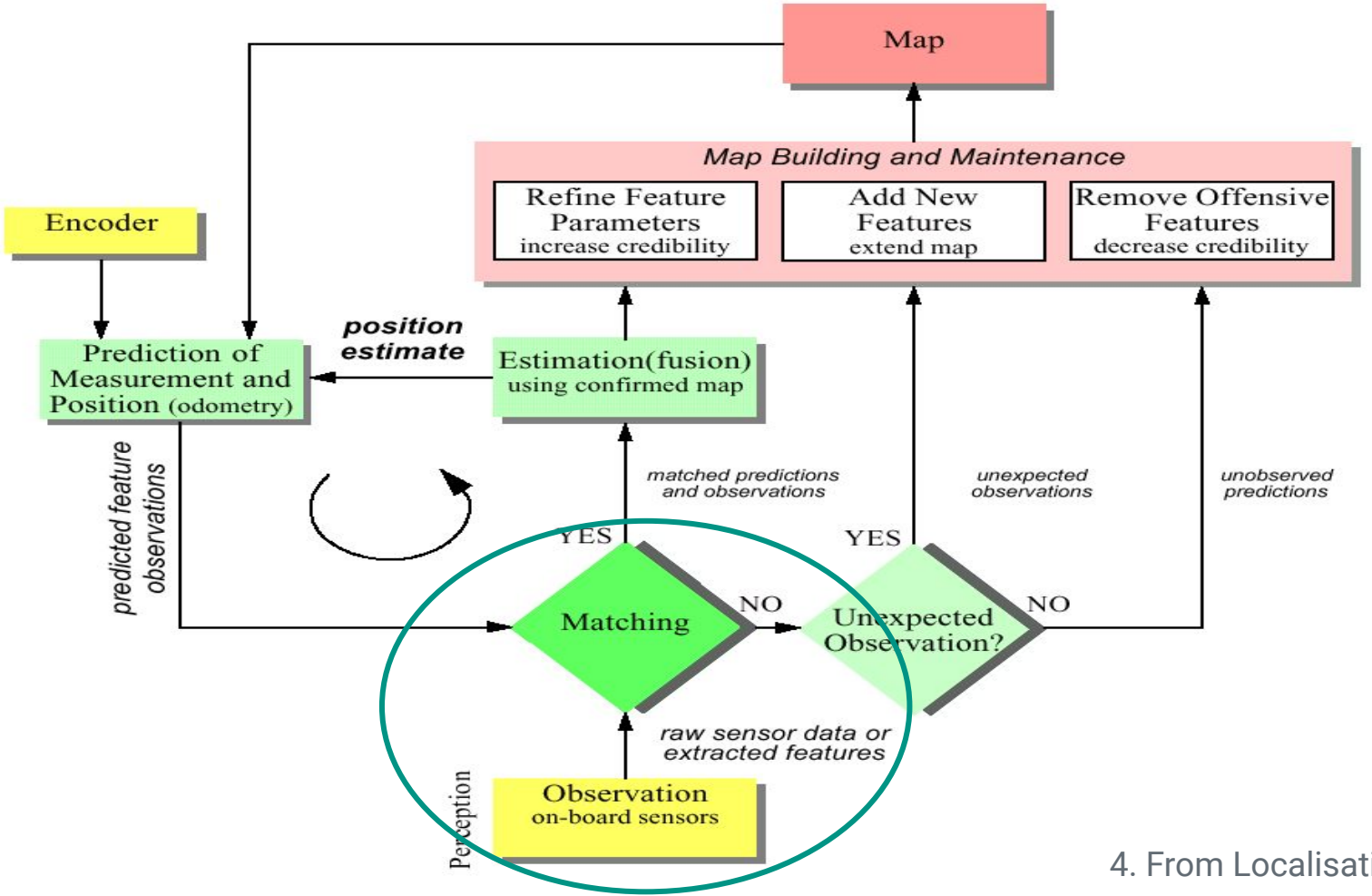

Map
Map Building and Maintenance
Refine Feature Parameters
increase credibility
Add New Features
extend map
Remove Offensive Features
decrease credibility
Encoder
position estimate
Prediction of Measurement and Position (odometry)
Estimation(fusion)
using confirmed map
predicted feature observations
matched predictions and observations
unexpected observations
unobserved predictions
YES
YES
Matching
NO
Unexpected Observation?
NO
raw sensor data or extracted features
Perception
Observation
on-board sensors

Example : Move and Prediction

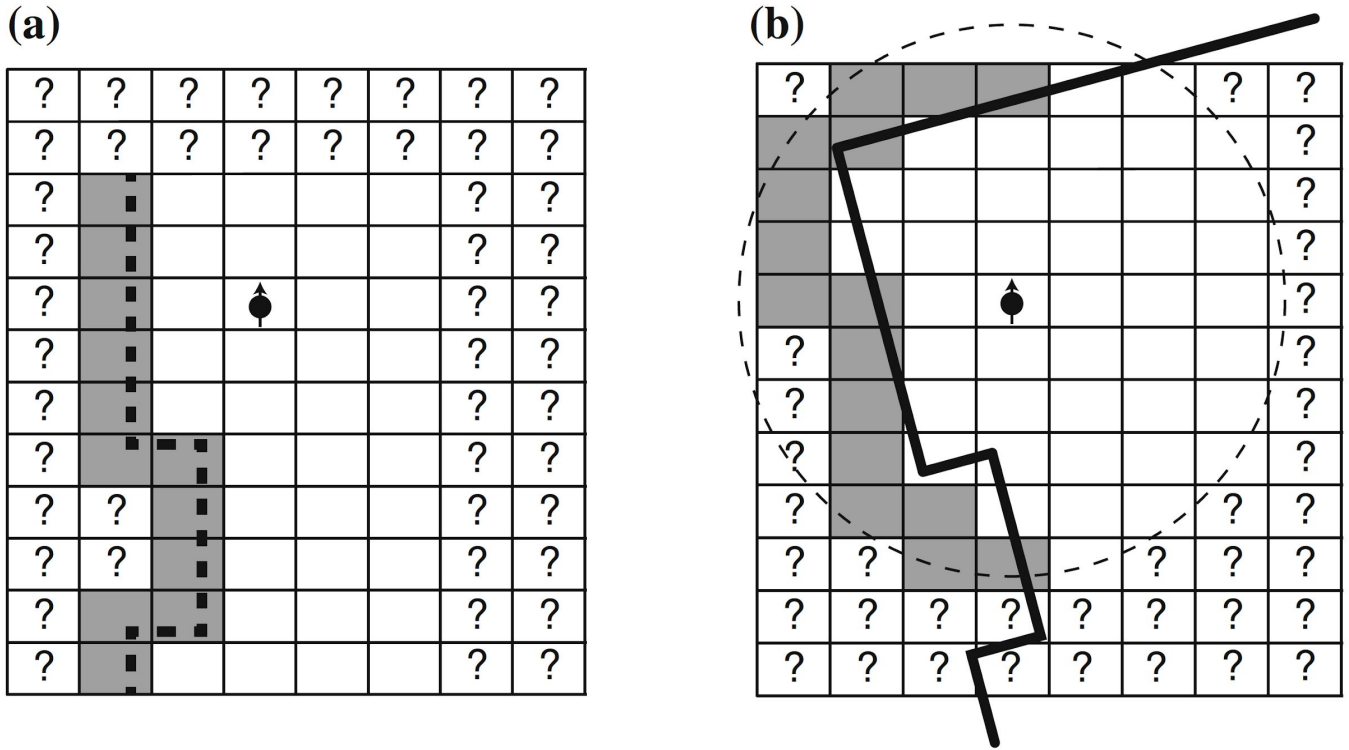


Example : Real Situation

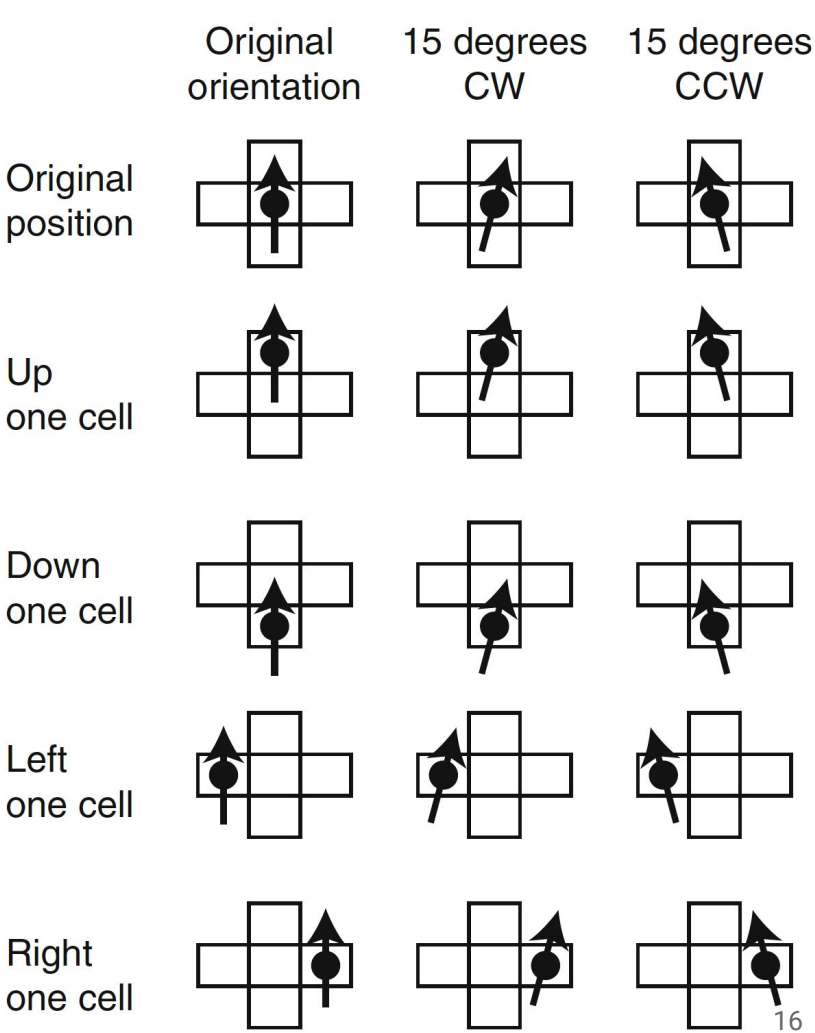



Map
Map Building and Maintenance
Refine Feature Parameters
increase credibility
Add New Features
extend map
Remove Offensive Features
decrease credibility
Encoder
position estimate
Prediction of Measurement and Position (odometry)
Estimation(fusion)
using confirmed map
predicted feature observations
matched predictions and observations
unexpected observations
unobserved predictions
YES
YES
Matching
NO
Unexpected Observation?
NO
raw sensor data or extracted features
Observation
on-board sensors
Perception

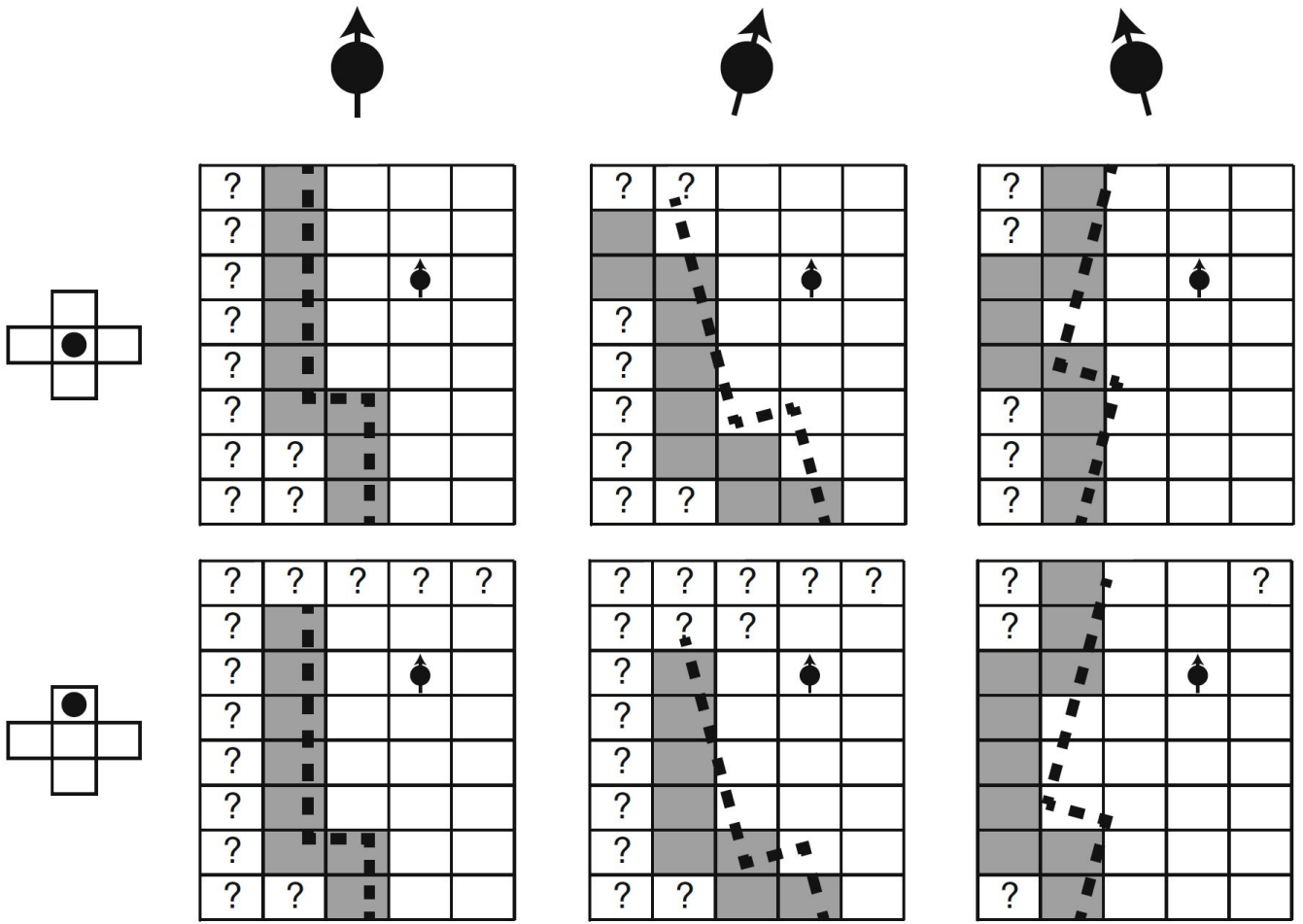
Example : Prediction vs. Observation



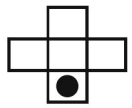
Example: Close Poses

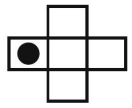


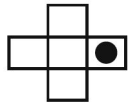
Example : Predictions

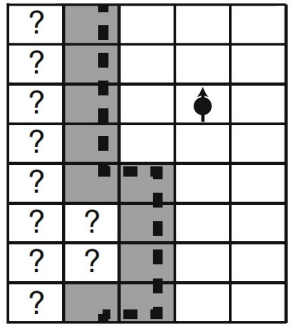


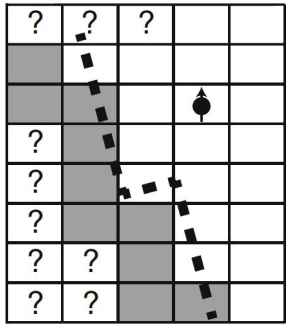
Example : Predictions

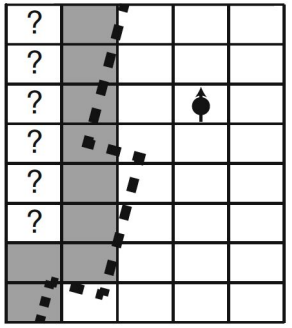


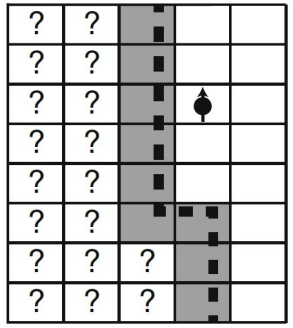


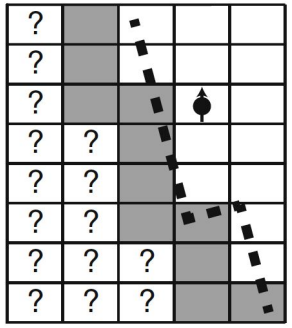


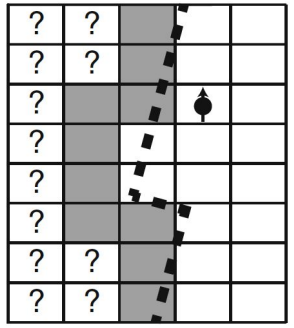


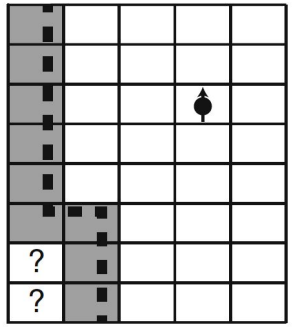


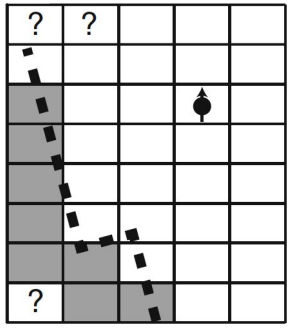


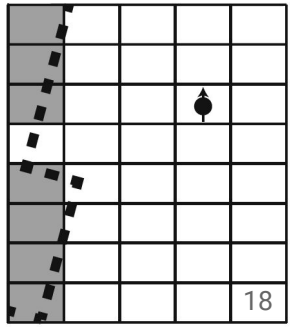




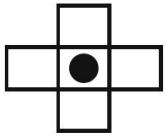








Example: Fit Prediction – Observation



0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	1	-1	-1
0	0	1	-1	-1
0	0	1	-1	-1

x

1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	1	-1	-1
0	0	1	1	-1

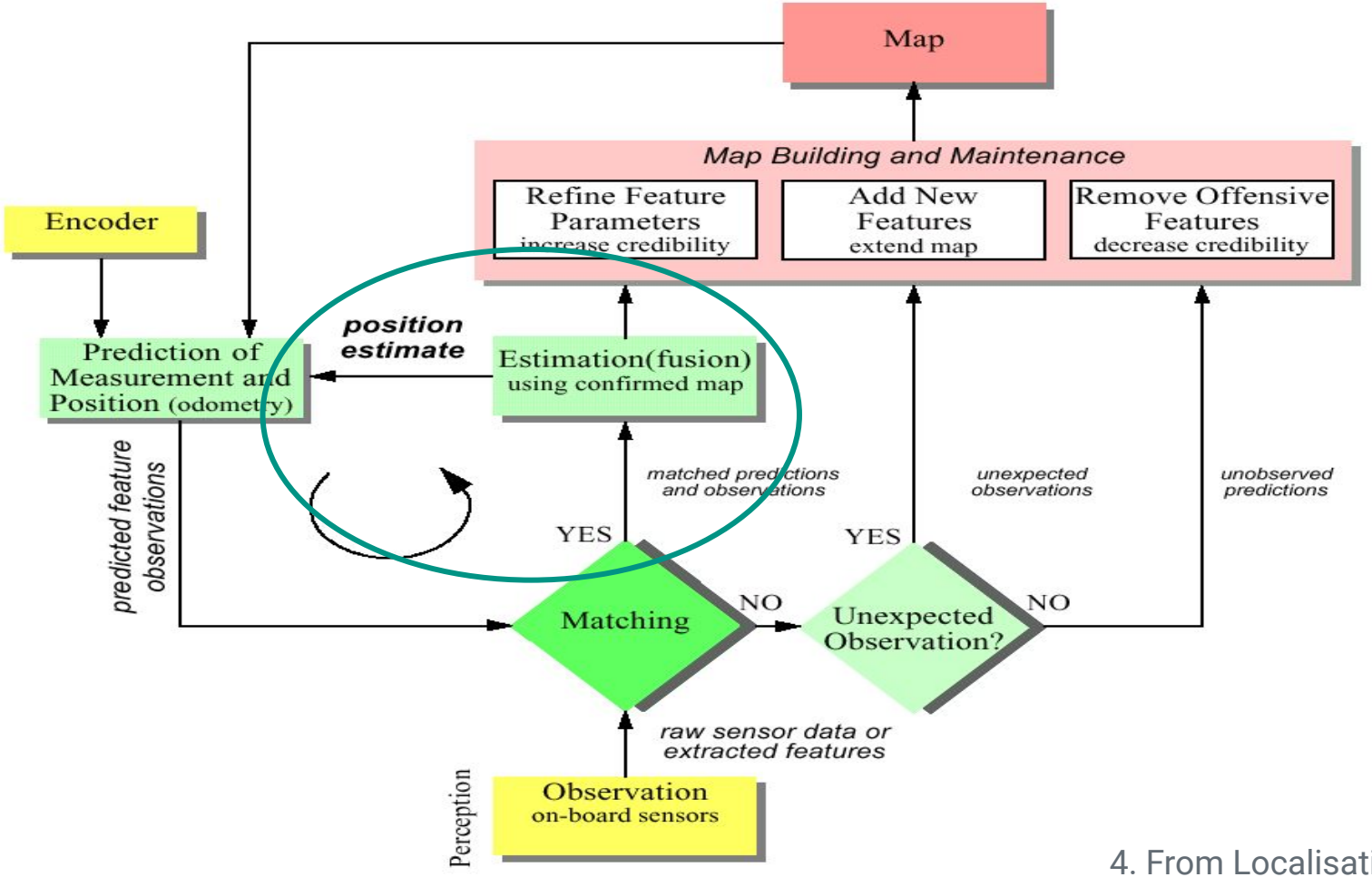
=

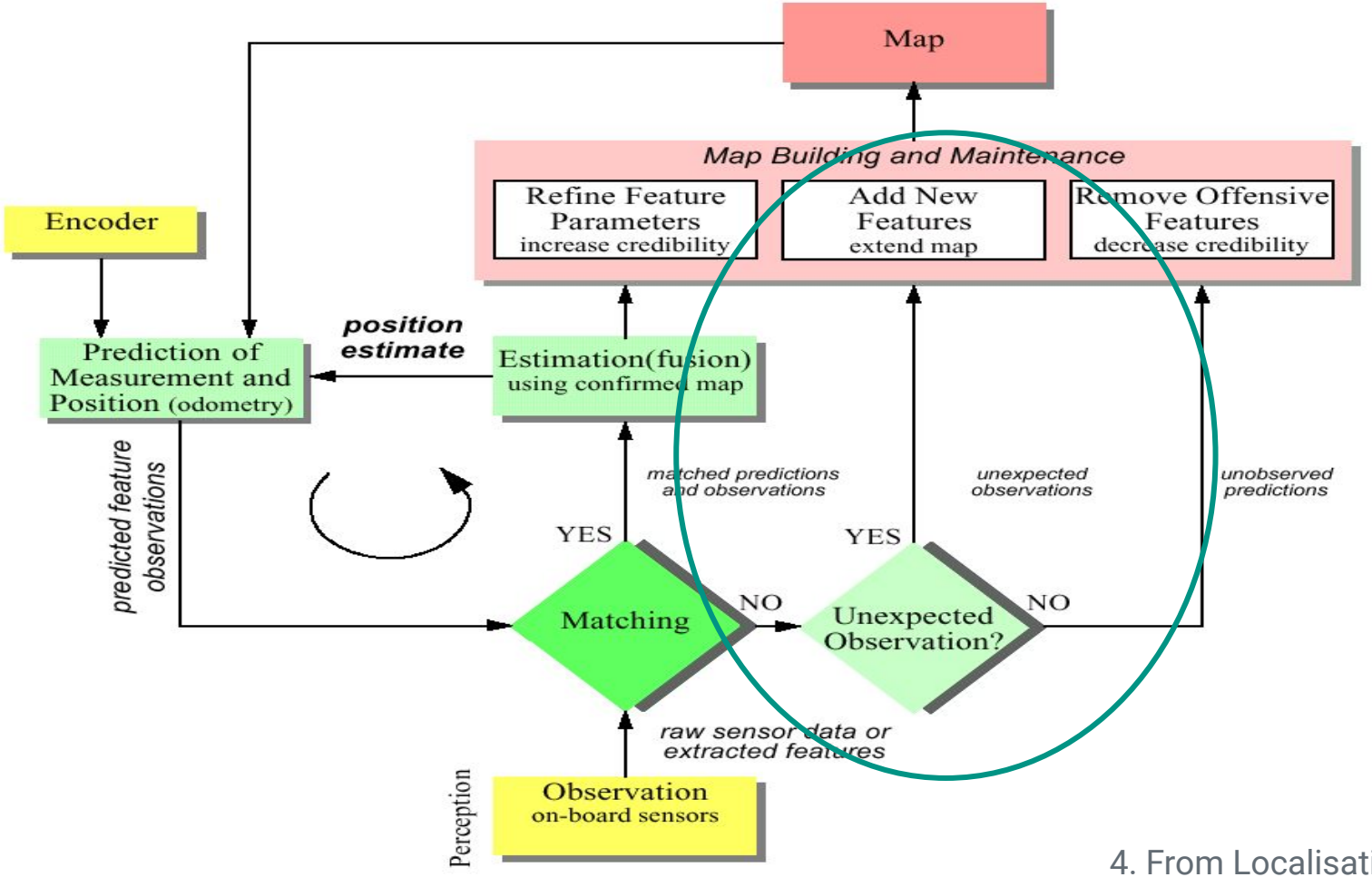
0	-1	1	1	1
0	-1	1	1	1
0	1	1	1	1
0	1	1	1	1
0	1	1	1	1
0	1	-1	1	1
0	0	1	1	1
0	0	1	-1	1

Example: Fit Prediction – Observation

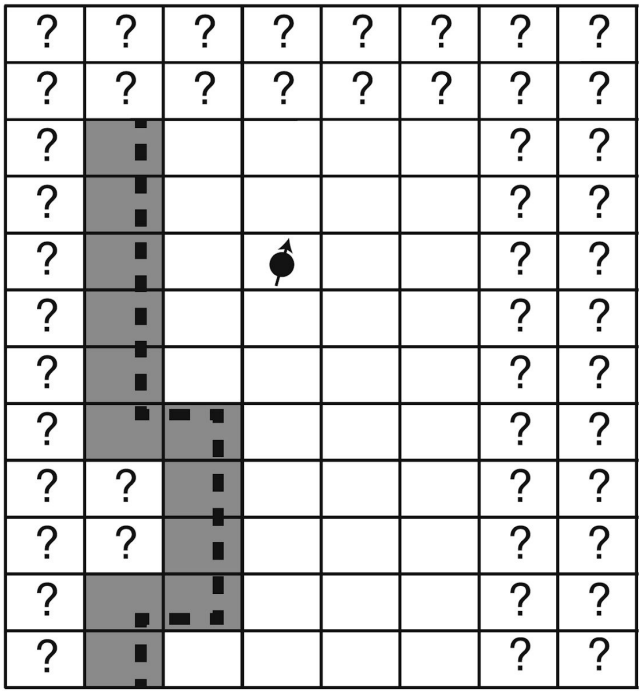
Table 9.1 Similarity $\mathcal{S}$ of the sensor-based map with the current map

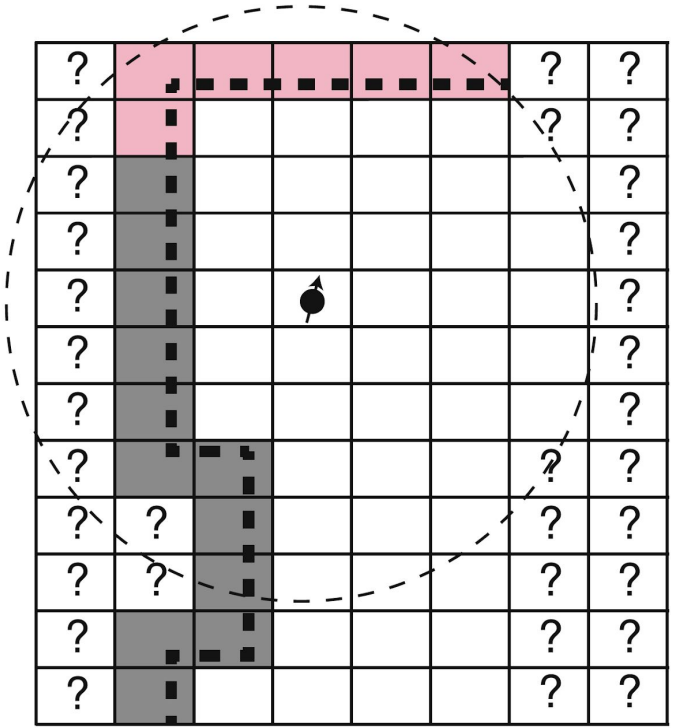
	Intended orientation	15° CW	15° CCW
Intended position	22	**32**	20
Up one cell	23	25	16
Down one cell	19	28	21
Left one cell	6	7	18
Right one cell	22	18	18


Map
Map Building and Maintenance
Refine Feature Parameters
increase credibility
Add New Features
extend map
Remove Offensive Features
decrease credibility
Encoder
position estimate
Prediction of Measurement and Position (odometry)
Estimation(fusion)
using confirmed map
predicted feature observations
matched predictions and observations
unexpected observations
unobserved predictions
YES
YES
Matching
NO
Unexpected Observation?
NO
raw sensor data or extracted features
Perception
Observation
on-board sensors


Map
Map Building and Maintenance
Refine Feature Parameters
increase credibility
Add New Features
extend map
Remove Offensive Features
decrease credibility
Encoder
position estimate
Prediction of Measurement and Position (odometry)
Estimation(fusion)
using confirmed map
predicted feature observations
matched predictions and observations
unexpected observations
unobserved predictions
YES
YES
Matching
NO
Unexpected Observation?
NO
raw sensor data or extracted features
Perception
Observation
on-board sensors

Features Matching





APPLICATION: 3D MODELING

SLAM – Resources

Some interesting open-source SLAM resources:

- http://www.openslam.org contains a comprehensive lists of SLAM software currently available
- https://github.com/tzutalin/awesome-visual-slam contains a list of vision-based SLAM / Visual Odometry open source projects, libraries, dataset, tools, and studies
- Google “SLAM summer school” for educational material

SLAM – Different Approaches

- Volumetric Versus Feature-Based
  - Volumetric has sufficient information to allows rendering of the environment
  - Feature-Based use only some features, no rendering possible
- Topological Versus Metric
  - Topological only considers the relation between positions
  - Metric adds metric information
- Static Versus Dynamic
  - Most of the literature assumes static environments, without changes over time.

SLAM – Different Approaches

- Active Versus Passive
  - In active approach the robot moves to optimize the map construction. Most approaches are passive, based on pure observation.

- Single-Robot Versus Multi-Robot
  - Multi-robot approaches can be based on local communication for better scalability, for instance (distributed system)

Active Map Building : Exploration

- Free center of the map: low obstacles occupancy probabilities (0. 1 or 0. 2).
- Three known obstacles, characterized by high occupancy probabilities (0. 9 or 1. 0).
- Unknown cells elsewhere.
- Frontier cells are free cell that is adjacent (left, right, up, down) to one or more unknown cells. The set of frontier cells is called the frontier (red lines).

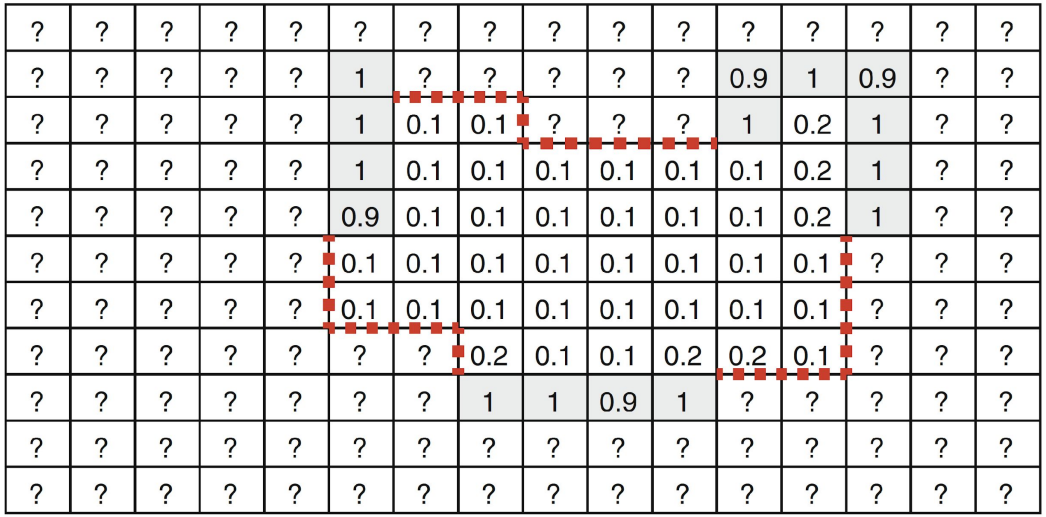
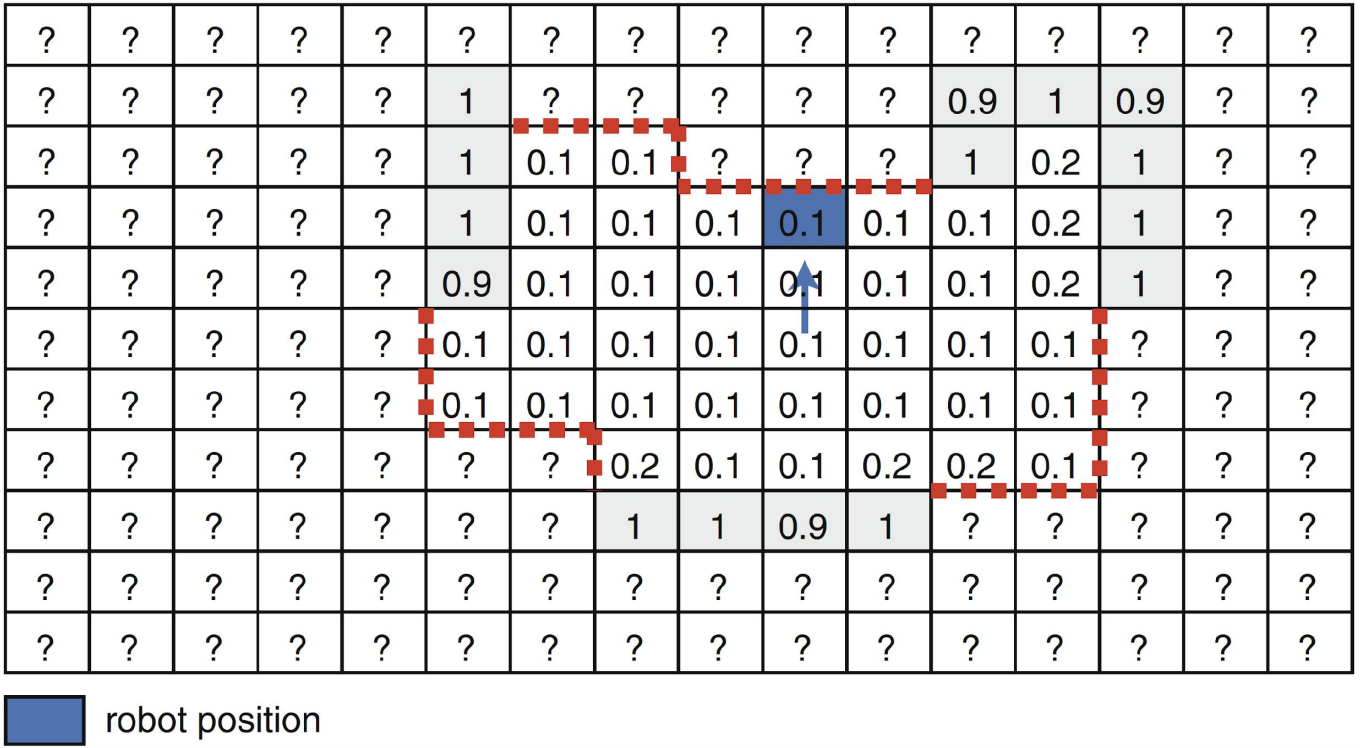
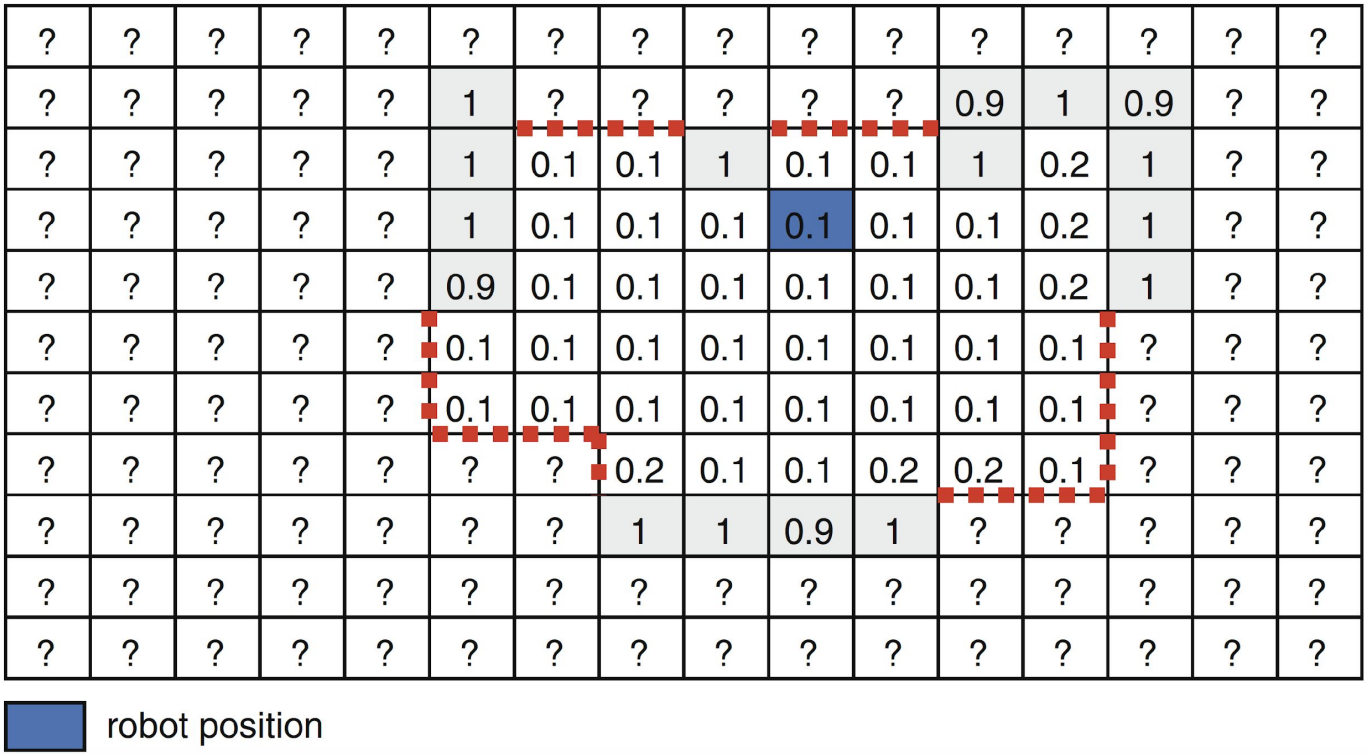


Fig. 9.4 Grid map of an environment with occupancy probabilities

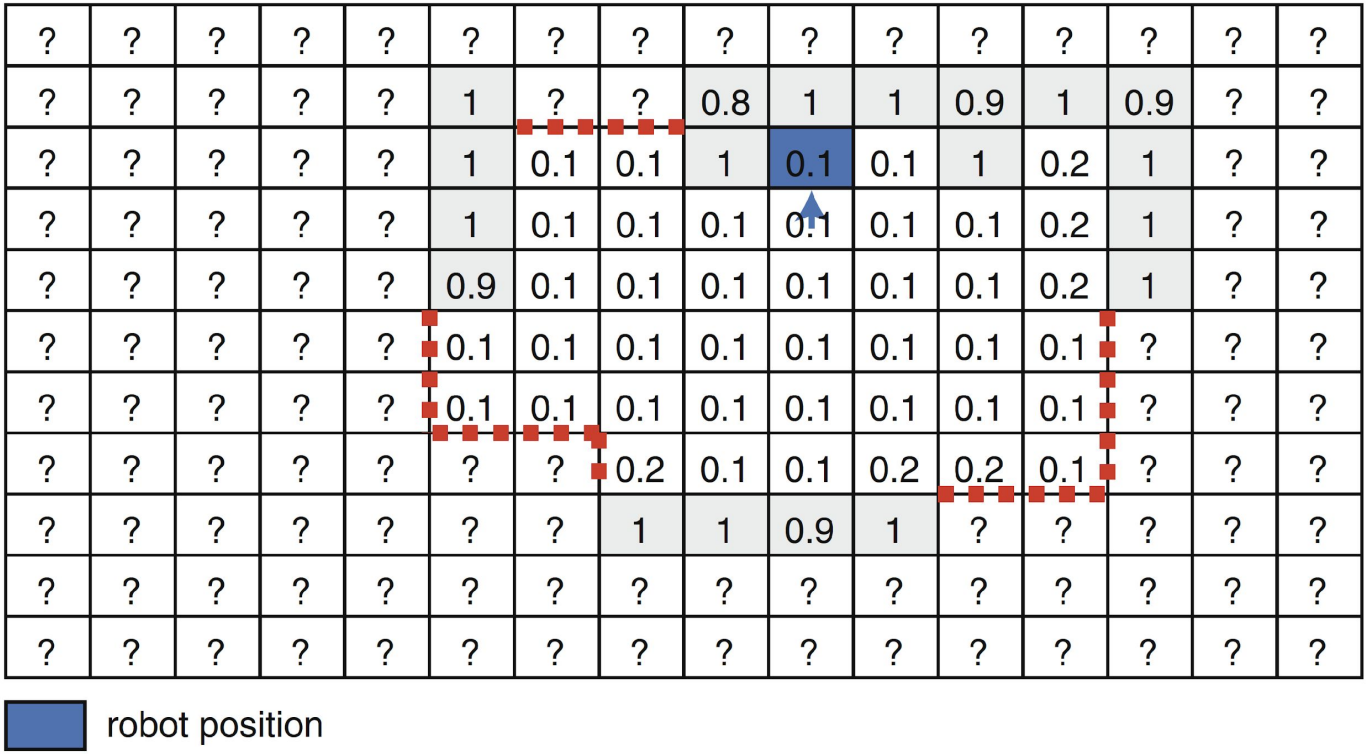
Active Map Building : Exploration



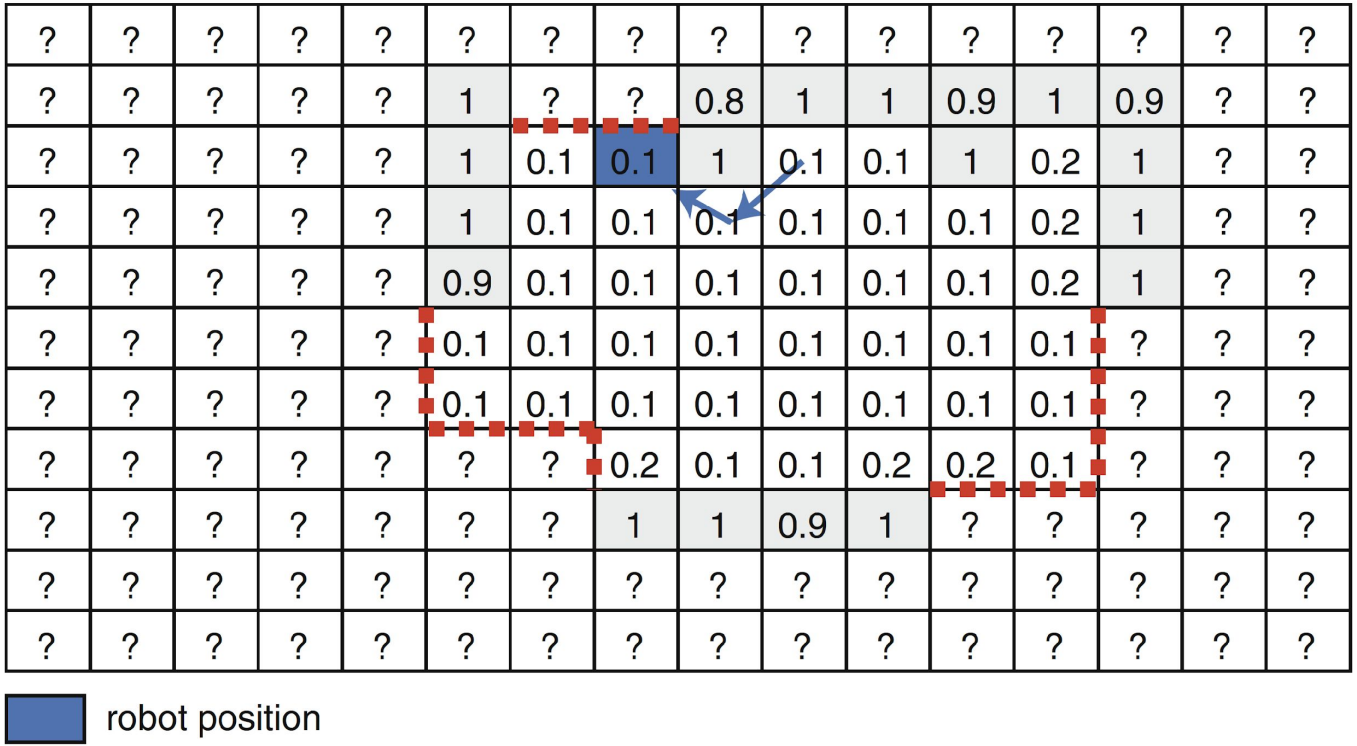
Active Map Building : Exploration



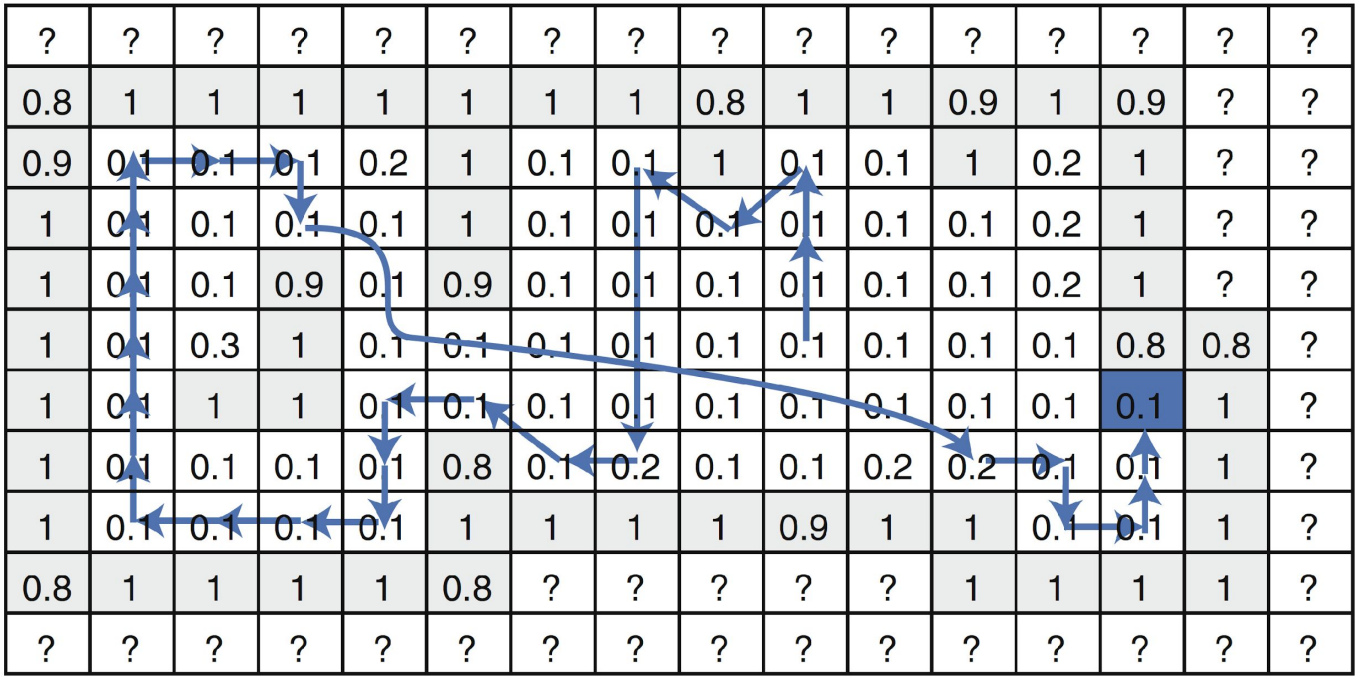
Active Map Building : Exploration



Active Map Building : Exploration



Active Map Building : Exploration



Exploration Criteria

You are here

Frontier cells

The criteria can be changed:

- the closest cell
- the higher number of unknown adjacent cells