COGNITIVE ROBOT SYSTEMS - Contents

Universität Duisburg-Essen
Fakultät für Informatik
Gruppe Intelligente Systeme

WS 23/24

Josef Pauli

crs.is@uni-due.de

Contents at a glance

Outline in details

Outline in details

Outline in details

Outline in details

Outline in details

Outline in details

Outline in details

Outline in details

Outline in details

Outline in details

Literature

- Slides for the course,
  `https://www.uni-due.de/is/wintersemester_2324#CRS`
- B. Siciliano, O. Khatib (Hrsg.): Handbook of Robotics, Springer, 2008 Available free with UNI DUE access
- R. Arkin: Behavior-Based Robotics; The MIT Press, Massachusetts, USA, 1998
- J. Latombe: Robot Motion Planning; Kluwer Academic Publishers, 1991

Literature

- S. Thrun, W. Burgard, D. Fox: Probabilistic Robotics, The MIT Press, 2005
- R. Sutton, A. Barto: Reinforcement Learning - An Introduction, The MIT Press, 1998 Available free on `https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf`
- Selected journal articles
- A. Koubaa (ed.): Robot Operating System; Springer, 2016
- Link to Robot Operating System (ROS), `https://www.ros.org/`

COGNITIVE ROBOT SYSTEMS - Applications

- Space Robots
- Underwater Robots
- Defense, Rescue & Security Robots
- Field Robots

COGNITIVE ROBOT SYSTEMS - Applications

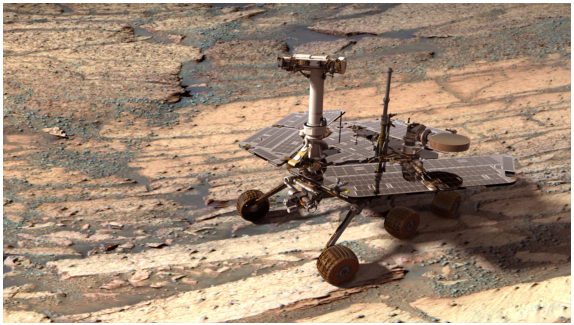
- Inspection Robots
- Industrial Cleaning Robots
- Construction/Production Robots
- Demolition Robots
- Logistics Robots
- Laboratory Robots
- Medical Robots

COGNITIVE ROBOT SYSTEMS - Applications

- Household Robots
- Handicap Assistance and Relief Robots
- Humanoid Robots
- Entertainment & Leisure Robots
- Public Relation Robots

Space Robots

Exploring (URL1), Handling, Repairing, in Outer space.
E.g. Mars rover Opportunity and Curiosity.

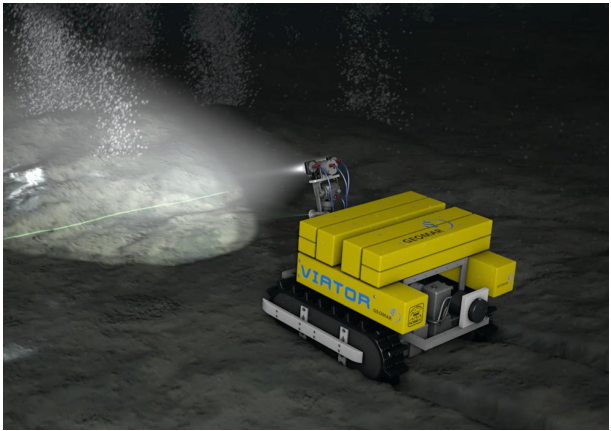


URL2

URL3

Underwater Robots

Manipulation/inspection in the deep sea, etc.
E.g. Crawler VIATOR in deep sea (URL1).



URL2

Defense, Rescue & Security Robots

Reconnaissance robot, Combat drones, Mine clearing, Deactivation (e.g. unexploded bombs), Search and rescue, Fire control, etc.

URL

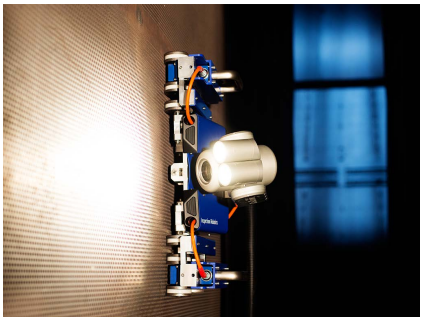
Field Robots

Agriculture, Forestry, Milking robot, Lawn mowing, etc.
E.g. Smart farming (syn. Precision agriculture, URL1).

URL2

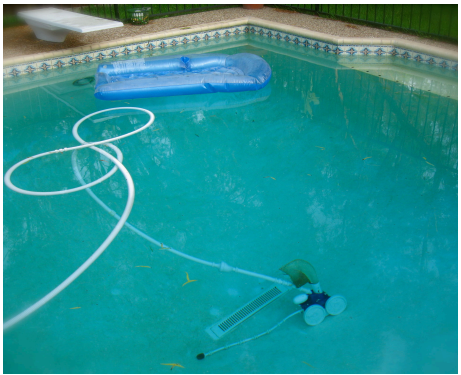
Inspection Robots

Inspection of Sewer, Container, Pipeline, etc.
E.g. Camera-equipped mobile robots for visual inspection (URL).



Industrial Cleaning Robots

Cleaning of floor, window, wall, roof, container, pipeline, etc.
E.g. Swimming pool cleaning (URL).



Construction/Production Robots

Flexible, varied production with few restrictions on surroundings, etc.
E.g., by cooperation of robot arm and human worker.

URL

Demolition Robots

Demolition (e.g. asbestos affected buildings), Disassembly (e.g. electronic waste), etc.

URL

Logistics Robots

Courier/postal services, Warehousing, Mail-order selling, etc.

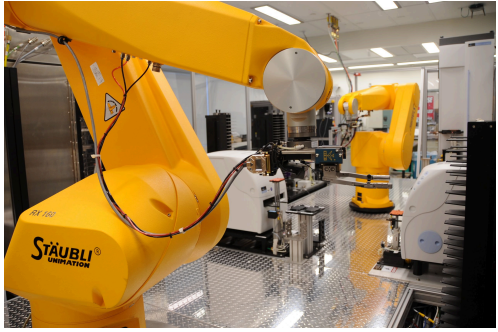
URL1

URL2

Laboratory Robots

Cleanroom robots, Robots for biological lab, etc.
E.g. Robots for High-Throughput Screening.

URL1



URL2

Medical Robots

Robot-assisted diagnostic systems, Robot-assisted surgery/therapy (e.g. treatment of urological cancer), etc.

URL1

URL2

Household Robots

Vacuum cleaning, Kitchen assistant, Ironing clothes, etc. (URL1).
E.g. Vacuum cleaner Roomba of iRobot (URL2, URL3).

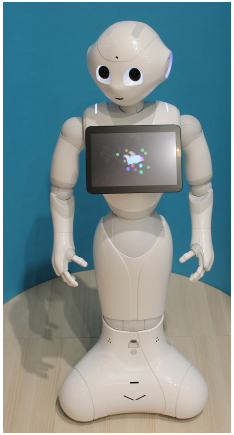
Handicap Assistance and Relief Robots

Wheelchair, Exoskeleton, etc.
E.g. Mate Exoskeleton (URL).

Humanoid Robots

Anthropomorphic robots to imitate human behaviour, etc.
E.g. Robots NAO and Pepper from Aldebaran/SoftBank.

URL1



URL2

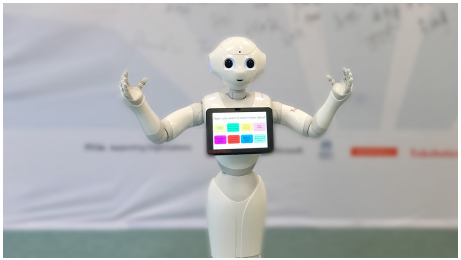
Entertainment & Leisure Robots

Playmate, Hobbies, Entertainment, Training, Education, etc.
E.g. Robotic dog Zoomer.

URL

Public Relation Robots

Hotel robots, Visitor guidance, Marketing, etc.



URL

Further hints

- Aufstieg der Roboter - Intelligente Helfer
  `https://www.youtube.com/watch?v=A6ireYAKX1U`
- Aufstieg der Roboter - Humanoide
  `https://www.youtube.com/watch?v=i8u3Z4Voafo`
- Aufstieg der Roboter - Menschen 2.0
  `https://www.youtube.com/watch?v=vC8LvWhFlgg`

1. Cognitive Perception-Action Systems

Overview:

1.1 Perception-action cycle

Overview:

- States, actions, state transitions, behaviours
- Inclusion of perception
- Inclusion of system knowledge
- Realisation of the functions

States

$\mathcal{S} := \{s_0, s_1, s_2, ...\}$ set of environmental states, e.g.
s_i a 6 dimensional vector, which describes the spatial relationship between gripper and target object (or between camera and target object).

$\mathcal{S}^{seq}$: set of sequences of the elements from $\mathcal{S}$

$\mathcal{S}^{pow}$: power set of $\mathcal{S}$

Actions

The component which acts in the environment is called Actuator (or Effector)

$\mathcal{A} := \{a_0, a_1, a_2, ...\}$ set of possible actions

Decision function

Robot system includes decision function $f_D : \mathcal{S} \rightarrow \mathcal{A}$, or $f_D : \mathcal{S}^{seq} \rightarrow \mathcal{A}$.

In the first case, pure reactive robots decide on the next action using only the current state.

In the latter case, decision on the next action is based on a sequence of previous states, e.g. to drive around without revisiting "former regions".

State transitions

The transition between states through an action is represented as function $f_T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. This deterministic transition, however, applies only in a virtual world or refers to a certain point of time.

In real environments, generally, the effects of actions are not deterministic, i.e. a certain combination of state and action can result in different successive states, therefore $f_T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}^{pow}$.

Behaviours

Sequence of interactions between robot and environment is called Behaviour:

$$\mathcal{B} : s_0 \overset{a_0}{\rightarrow} s_1 \overset{a_1}{\rightarrow} s_2 \overset{a_2}{\rightarrow} s_3 \quad \ldots \quad s_{t-1} \overset{a_{t-1}}{\rightarrow} s_t \rightarrow$$

s_0 is initial state, $\mathcal{B}$ is possible behaviour.

The following must hold:
$a_t = f_D(s_{t-k}, ..., s_t)$; $s_{t+1} \in f_T(s_t, a_t)$

Behaviours

If a certain set of requirements (properties) is met through a robot behaviour, we say this behaviour is *acceptable*.

Usually, there are several acceptable behaviours. If an optimality criterion is met, we say this behaviour is *optimal*.

Inclusion of perception

States s_i must be perceived (Perception). This leads to measurements, e.g. direct distance measurement with LASER ($f_P : \mathcal{S} \rightarrow \mathcal{M}$), or reconstruction of object distance via "Motion Stereo" using a single standard video camera ($f_P : \mathcal{S}^{seq} \rightarrow \mathcal{M}$).

Inclusion of perception

Perception function:
$f_P : \mathcal{S} \to \mathcal{M}$

Decision function redefined:
$f_D : \mathcal{M} \to \mathcal{A}$

State transition function, same as formerly:
$f_T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$

Inclusion of perception

Problem of perception:

Two different states, e.g. $s_i \in \mathcal{S}, s_j \in \mathcal{S}$, might be perceived as identical, i.e.

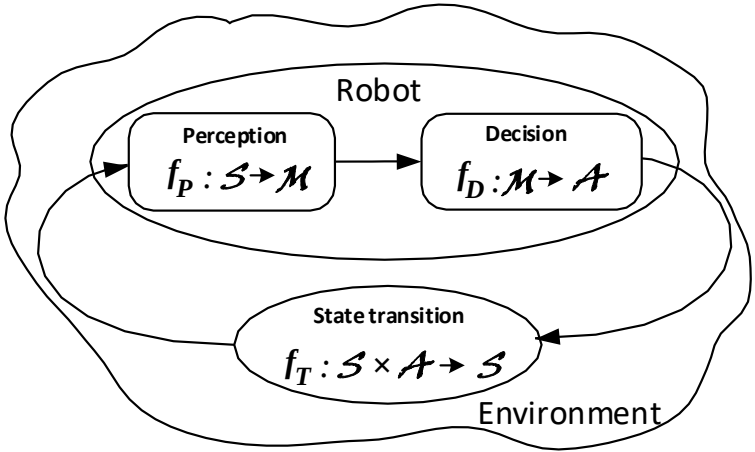
$$s_i \neq s_j, f_P(s_i) = f_P(s_j)$$

Then, possibly a wrong decision will be made.

Idea for solution: Unique perception could be reached by “Active Vision”.

Inclusion of perception

Basic perception-action cycle:



Inclusion of system knowledge

- Inherent robot features, e.g. arm length of manipulators
- Beliefs (assumptions), Desires (goals), Intentions (options, plan)
- Sequence of measurements

$\rightarrow$ System knowledge $\mathcal{I}$

Inclusion of system knowledge

Refreshment function for system knowledge:
$f_R : \mathcal{I} \times \mathcal{M} \to \mathcal{I}$

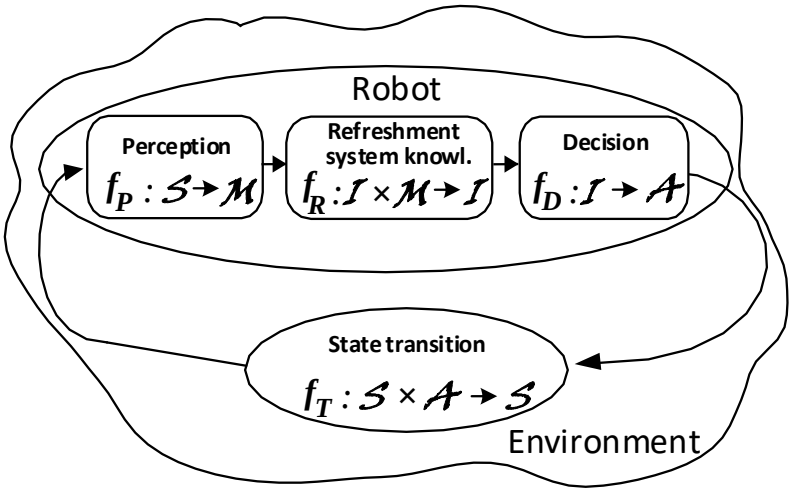
Decision function redefined:
$f_D : \mathcal{I} \to \mathcal{A}$

Perception function, same as formerly:
$f_P : \mathcal{S} \to \mathcal{M}$

State transition function, same as formerly:
$f_T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$

Inclusion of system knowledge

Advanced perception-action cycle:



Inclusion of system knowledge

$$\underbrace{f_D(f_R(i_{t-1}, \underbrace{f_P(s_t)}_{m_t}))}_{i_t} =: a_t \quad (\leftarrow \text{new action})$$

$$f_T(s_t, a_t) =: s_{t+1} \quad (\leftarrow \text{new state})$$

Realisation of the functions

Perception function f_P:

- CRS_3_SEN, Sensor System as Basis for Autonomy
- Course Grundlagen der Bildverarbeitung, SS
- Course Computer/Robot Vision, WS
- Course Neuroinformatik und Organic Computing, SS

Decision function f_D:

- CRS_4_VCR, Vision-based Control of a Robot Arm
- CRS_6_NAV, Path Planning for Robot Navigation
- CRS_8_ORL, Online Learning for Robot Navigation

Realisation of the functions

State transition function f_T:

- CRS_5_RSE, Representation of Scene/Environment
- CRS_6_NAV, Path Planning for Robot Navigation
- CRS_7_PRL, Probabilistic Robot Localisation

Refreshment function f_R:

- CRS_7_PRL, Probabilistic Robot Localisation
- CRS_8_ORL, Online Learning for Robot Navigation

1.2 Systems layered in competences

Overview:

- Brooks' subsumption architecture
- Müller's hybrid architecture

Brooks' subsumption architecture

Also called **Behaviour-based Architecture** or **Competence-based Architecture**.

See Brooks (1986).

Brooks' subsumption architecture

Example of a layered architecture of a mobile robot system

Sensors

	...
5	Move to target object
4	Classify object
3	Build environment map
2	Drive around randomly
1	Avoid contact with obstacle

Effectors

Brooks' subsumption architecture

In early years of AI research, an autonomous robot system was proposed according to the following functionally modularised architecture:

Research/Development in/of the main areas, namely Perception, Planning and Control, was assumed to be achievable separately by scientists of different desciplines.

Brooks' subsumption architecture

However:

- To realise a component, one needs the results of other components.
- E.g. a planning component needs the results of the perception component.
- However, e.g. Computer Vision in general is not solved, i.e. work in progress.
- Accordingly, it was proposed to use simulated representations of the environment (simulations, models), in order to go ahead with implementing the planning component.
- However, there is insufficient similarity between model and reality.
- Therefore, according to the concept of classical AI research, an autonomous robot system could not be realised.

Brooks' subsumption architecture

Brooks proposed:

Perception, Planning and Control are to be realised in direct relation to reality, and only for a certain application in a certain environment.

This takes place on several competence layers (CL), on which the behaviours of a robot are realised.

Each CL has direct access to sensors and effectors of the robot system.

Each CL is realised as a network of elementary processing units (EPUs).

Brooks' subsumption architecture

Each EPU is running on an asynchronous processor.

An EPU receives data from input channels, processes it, and sends the result to output channels.

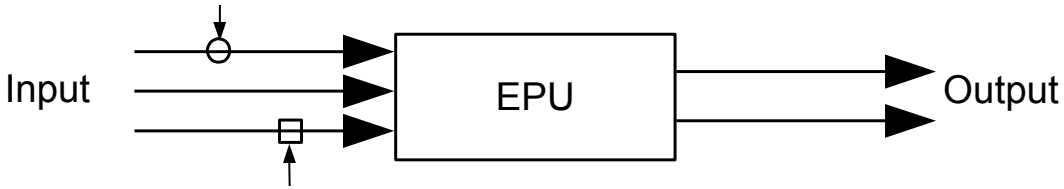
Brooks' subsumption architecture

There are 2 types of suppression regarding the interconnection of EPUs from different competence layers (or within the same competence layer):

- □ from lower competence layers, to ensure the survival of the robot. The lower the layer of competence, the higher the priority.
- ○ from upper competence layers, to allow the execution of overriding tasks. The upper the layer of competence, the higher the priority.

First type (□) has priority over the second (○).

Brooks' subsumption architecture



Several EPUs are simultaneously ready/active.

Which EPU "fires", i.e. effects the robot behaviour, depends on the interconnections and current suppressions.

If at an upper competence layer an EPU fails, the lower competence layers continue to work, i.e. functioning of a rudimentary system.

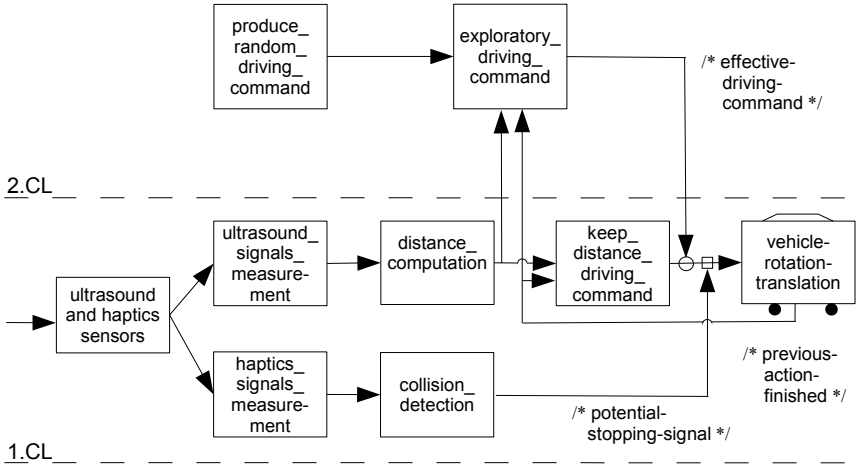
Brooks' subsumption architecture

According to the example of a layered architecture of a mobile robot system, the following strategy is realised in the 1st and 2nd competence layer:

"Vehicle continually drives around randomly, for the purpose of exploring the environment, and meanwhile avoids obstacles."

Brooks' subsumption architecture

Realisation of the two lower competence levels:



Brooks' subsumption architecture

Role of EPUs:

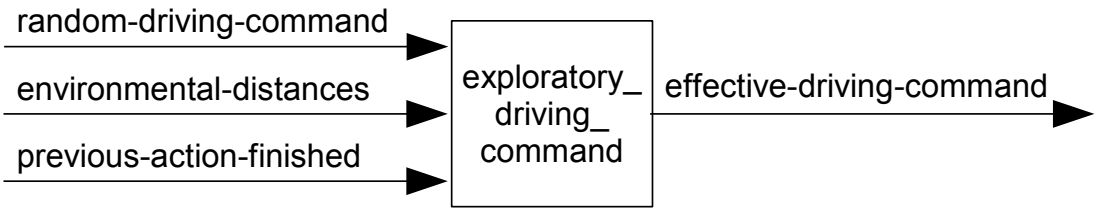
- ultrasound_signals_measurement: Continually measure sound propagation time to the nearest objects.
- distance_computation: Determine distances to the nearest objects based on sound propagation time.
- haptics_signals_measurement: Detect possible contact pointwise on the robot.
- collision_detection: Determine possible collision areas on the robot.

Brooks' subsumption architecture

- keep_distance_driving_command: Tries to achieve a maximum average distance to all the nearest objects.
- produce_random_driving_command: Send out random driving commands in the course of time.
- exploratory_driving_command: Check whether a random driving command is executable based on environmental data. If not, compromise, and send out the effective driving command. Purpose is to explore the environment.

Brooks' subsumption architecture

Exemplary implementation of an EPU:



Brooks' subsumption architecture

```
procedure exploratory_driving_command:

/* Input */
    random-driving-command,
    environmental-distances,
    previous-action-finished;

/* Output */
    effective-driving-command;

/* Internal variables */
    h;
```

Brooks' subsumption architecture

```
/* Event dispatch */
    wait:
      if (random-driving-command and
          environmental-distance and
          previous-action-finished)
      then go to fusion;
      else go to wait;

/* Set internal variables */
    fusion:
      h := compute_effective_driving_command (
                random-driving-command,
                environmental-distances);
      go to decision;
```

Brooks' subsumption architecture

```
/* Conditional dispatch */
    decision:
      if (relevant_action(h))
      then go to start;
      else go to wait;

/* Generate output */
    start:
      effective-driving-command := h;
      go to wait;
```

Brooks' subsumption architecture

Structure (components) of an EPU:

- Event dispatch: There is a sequence of pairs, each of which consists of a condition and a consequence. A condition is met by the occurrence of an event. Check through the conditions until one is matched, then go to the corresponding consequence branch.
- Set internal variables: Calculate values for certain internal variables, based on input and possibly other internal variables.

Brooks' subsumption architecture

- Conditional dispatch: Evaluate a predicate based on the calculation of internal variables, then go to the corresponding branch.
- Generate output: Generate output as a function of inputs and internal variables.

Brooks' subsumption architecture

Comments: Advantages

- Behaviour-based, horizontally layered in competences.
- All layers connected with relevant sensors and effectors.
- Potential for dependability (robustness, availability, reliability, maintainability, durability, safety).

Brooks' subsumption architecture

Comments: Drawbacks

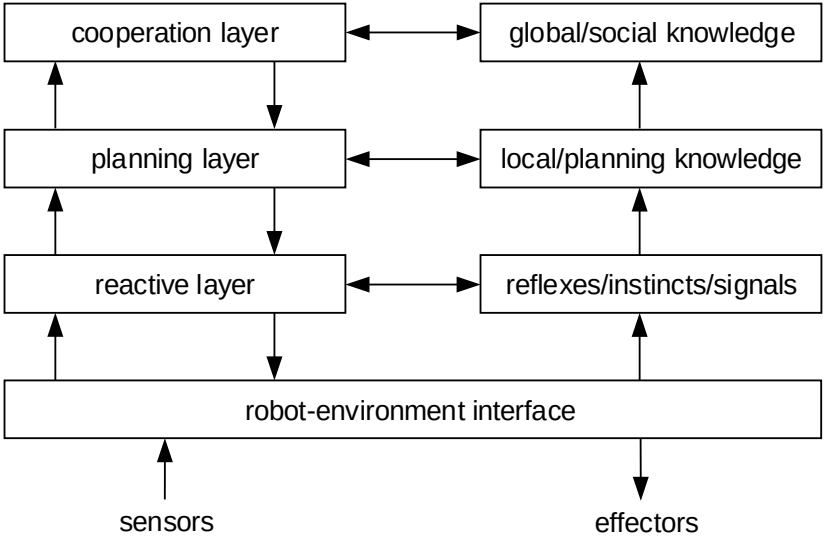
- Approach is not sufficient for challenging tasks, since knowledge representation is hardly provided.
- No systematics regarding the design of such systems.
- No proposals/hints for problem of perception, e.g. uncertainty, time consumption.

Müller's hybrid architecture

Layered organisation of competences in combination with **knowledge representation**.

See Müller (1997).

Müller's hybrid architecture



Müller's hybrid architecture

Grouping of the set of competences into three (macro-)layers:

- Lower ($reactive$) layer, responsible for reactive, survival behaviours.
- Medium ($local\ planning$) layer, creates local plan for the actions to be performed.
- Upper ($cooperation$) layer, takes care of global plan and "social interactions".

Information flows layer-by-layer up and control takes place top-down.

Müller's hybrid architecture

Each layer is associated with a knowledge base, which represents layer-relevant environmental knowledge:

- Lower-level knowledge base represents reflexes, instincts, and elementary measurements from environment.
- Medium-level knowledge base represents plans of local agents.
- Upper-level knowledge base represents the overall plan, including also rough plans of other agents in the system.

Müller's hybrid architecture

Strategy to generate an action:

- Bottom-up activation of upper layers only if the lower layers are not competent for the situation.
- When measurements reach the reactive layer, they could possibly be dealt within this layer, otherwise, control will be forwarded to planning layer.
- Top-down control, e.g. local plans may be suppressed when there are overriding ("social") goals.
- Medium and upper layers may be realised based on the so-called BDI concept (see later), but layered.

1.3 Realisation of deliberation

Overview:

- **Belief-Desire-Intention (BDI)** Architecture (Rao, 1995)

Belief-Desire-Intention Architecture

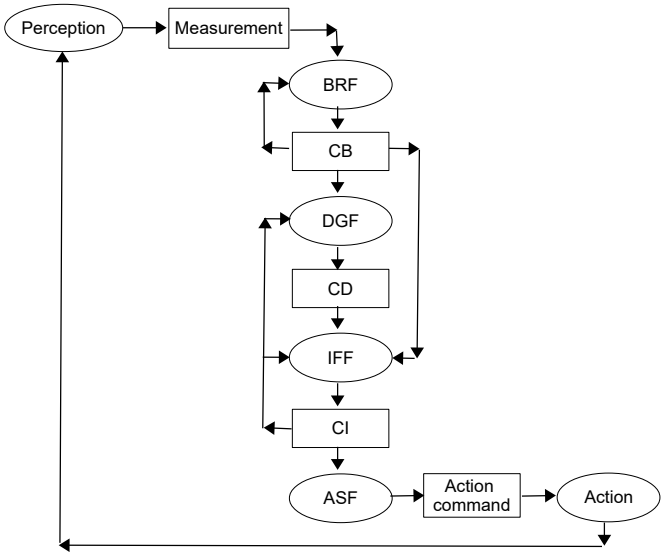
A "BDI-Agent" consists of 3 categories of data and 4 functions.

- Current Beliefs (CB): Information/Assumptions about environment.
- Current Desires (CD): Goal of agents/robots.
- Current Intentions (CI): Options/plans to reach the goal.

Belief-Desire-Intention Architecture

- Belief Revision Function (BRF): Update of belief according to measurements in environment.
- Desire Generation Function (DGF): Determines the goal for agents to follow.
- Intention Filter Function (IFF): Determines which plan (which option) can be applied (is available) to reach the goal.
- Action Selection Function (ASF): Determines executable action.

Belief-Desire-Intention Architecture



Belief-Desire-Intention Architecture

Interplay between DGF and IFF:

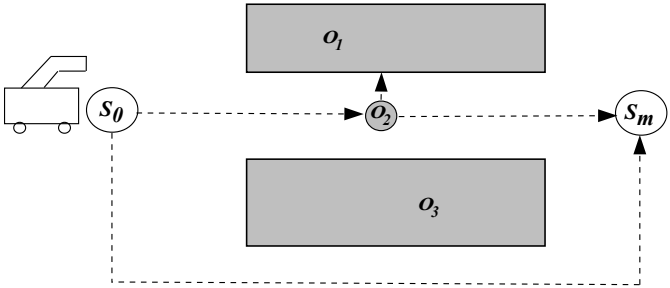
- In order to solve a goal, a rough plan is needed. This rough plan includes sub-goals, for which concrete sub-plans are made, etc. Iteratively refining a rough plan, until finally direct executable actions arise.
- Erase options which are no longer realisable, or the benefit/cost ratio is not favorable to reach the sub-goal.

Belief-Desire-Intention Architecture

- Keep options, which are not yet considered, but could be advantageous, e.g. bypassing an obstacle.
- Adopt new sub-goals, which enable/facilitate the achievement of other sub-goals, e.g. first push away an obstacle, in order to continue driving.

Belief-Desire-Intention Architecture

Example: Robot navigation



Start position s_0 of robot, target (goal) position s_m of robot, several obstacles o_1, o_2, o_3, maybe obstacle o_2 is light weight and slidable.

1.4 Desired characteristics

Overview:

- Cognitive Robot Systems in use
- Competence
- Situativity/Adaptivity
- Emergence
- Corporality

Cognitive Robot Systems in use

Future devices / systems / objects are “smart”, i.e. they are perception-action systems, which:

- include cognitive skills,
- have (possibly distributed) sensors and/or actuators,
- perceive the environment and its own corporality,
- collect, organise, and use knowledge autonomously,
- based on above, make meaningful decisions leading to a sophisticated behaviour,
- react/act flexibly in real time.

Cognitive Robot Systems in use

With the cognitive skills, systems will achieve a higher degree of semi-autonomy, and will become more flexible in a wider spectrum of applications, and will also be better accessible, usable, and maintainable for persons.

Competence

Layered organisation of competences; elementary competences responsible to survive (reactive behaviours); and upon them there are more layers with increasingly sophisticated competences (deliberate behaviours). In case of failure of a competence in a certain layer, the "lower" competences should still be in function.

Situativity/Adaptivity

Adapt generic behaviour models to specific environmental situations, and achieve competent behaviours in these situations.

Emergence

Autonomous emergence of sophisticated competences on the basis of simple, reactive, instinct-like competences.

Corporality

Through the selected sensors, illumination, and effectors, a robot system can perceive and modify the environment in a specific fashion. These have major influence on the emergence of behaviours.

Literature

- R. Brooks: A robust layered control system for a mobile robot, IEEE Journal of Robotics and Automation, 2:14–23, 1986.
- J. Müller: A cooperation model for autonomous agents. In J. Müller et al.: Intelligent Agents. III, Springer Verlag, LNAI 1193, pp. 245-260, 1997.
- A. Rao, et al.: BDI-Agents – From Theory to Practice; in Proc. of the 1. Int. Cont. on Multi-Agent Systems (ICMAS), 1995.

2. Components of Robot Systems

Overview:

2.1 Joints, degrees-of-freedom

Overview:

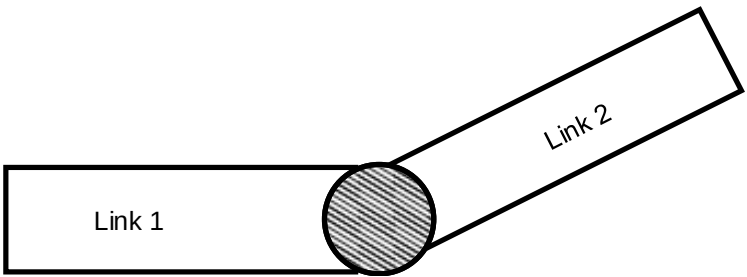
- Revolute joint
- Linear joint
- Screw joint
- Spherical joint

Revolute joint

Synonyms: rotary joint, hinge joint, pin joint, single-axis rotation joint

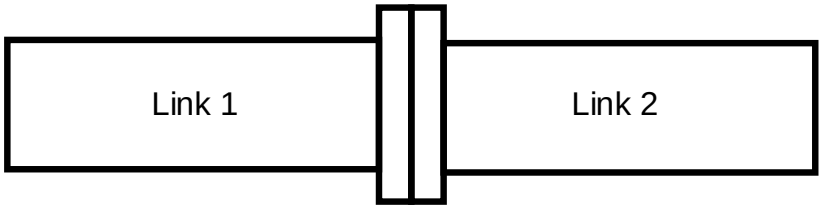
Definition: Rotational movement around a rotation axis

Example 1: Rotation axis is at right angle to the principal axes of both linking components (links).



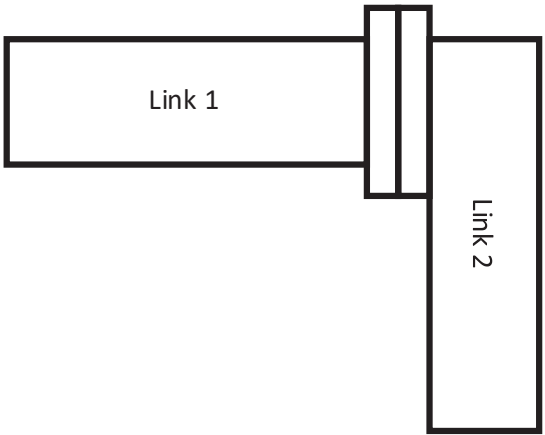
Revolute joint

Example 2: Rotation axis is parallel with the principal axes of both links (i.e. torsion joint).



Revolute joint

Example 3: One link is parallel with the rotation axis, and the other link is at right angle to it (i.e. revolver joint).

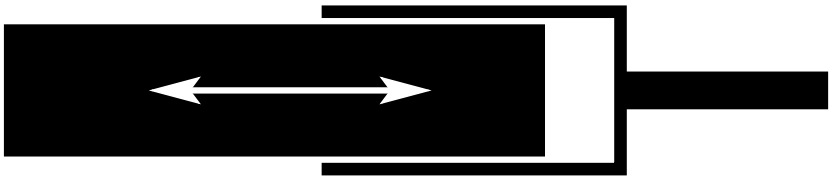


Linear joint

Synonym: prismatic joint

Definition: Linear sliding movement

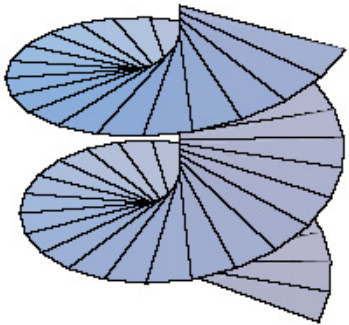
Example: Movement of a link along its principal axis.



Screw joint

Definition: Combined revolute and linear joint

Example: Screw



Spherical joint

Definition: Rotation around a pivot point.

Example: single wheel knuckle

2.2 Configuration of joints

Overview:

- Positioning and orientation of effector/camera
- Articulated robot arm
- Cartesian coordinate robot
- Selectively Compliant Assembly Robot Arm
- Telemanipulator
- Hexapod robots
- Robot head with one camera
- Robot head with two cameras

Positioning and orientation of effector/camera

Two successive joints are connected by a linking component (link).

A configuration of joints and links may be organised in two groups.

The first group (series) of links and joints enables the positioning of an effector or a camera.

The second group of links and joints enables the alignment/orientation of the effector or the camera.

Articulated robot arm

Definition: The effector/camera can be positioned and orientated arbitrarily in a certain sub-space.

Example: Robot arm RX-90 (Stäubli), kinematic chain with six revolute joints, with the first 3 for positioning and the second 3 for orientating of an effector.

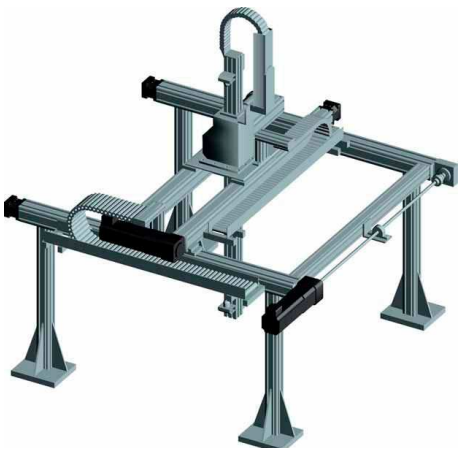
- 1 revolute joint for rotating around vertical axis
- 2 revolute joints for turning out of the horizontal plane
- 3 revolute joints for tool orientation
- Possibly 1 additional linear joint for fingers of a parallel-jaw gripper

Articulated robot arm

Cartesian coordinate robot

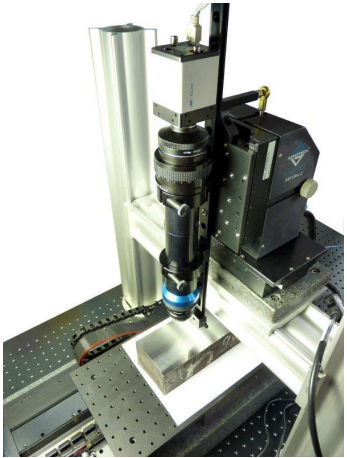
Definition: Linear orthogonal joints, only for positioning.

Example 1: Gantry robot for handling objects.



Cartesian coordinate robot

Example 2: Gantry robot for surface inspection (CHRISS)



Selectively Compliant Assembly Robot Arm (SCARA)

Definition: Series of 2 - 3 revolute joints, only for positioning the tool.

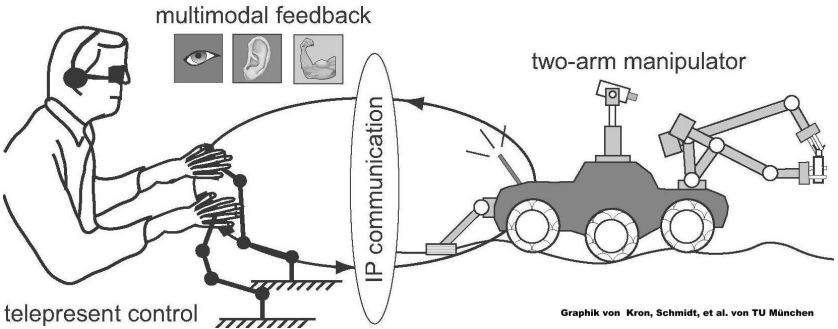
Task: Fast and exact positioning within a small area.

Example: SCARA XK from Yamaha.

Telemanipulator

"Master" part is handled by operator, while the remote "slave" part performs the actions.

Task: Robots for handling hazardous objects/substances; Robots for supporting surgery.



Telemanipulator

Hexapod robots

Definition: Six legs, each with one or more joints

Example 1: Hexapod robot for high-precision instrument handling, each leg with two spherical joints and one linear joint, legs are attached to two platforms (Stewart platform)

Hexapod robots

Example 2: Hexapod crawling robot, with each leg composed of three revolute joints; enables shape adaptation for inspecting narrow spaces

Robot head with one camera

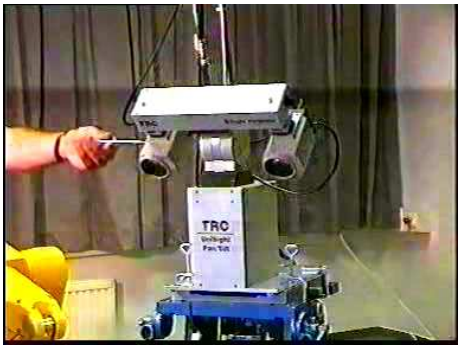
Configuration of joints: Yaw (turning the head), Pitch (tilting the head)

Example: from Sony

Robot head with two cameras

Configuration of joints: Yaw (turning the head), Pitch (tilting the head), Vergence (turning of two cameras for binocular vision)

Example: from TRC

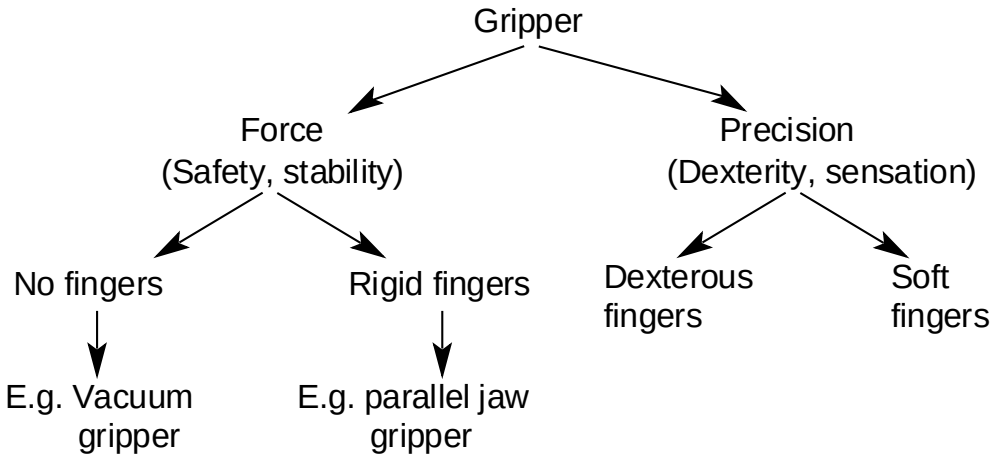


2.3 Effectors for handling and moving

Overview:

- Effectors for handling
  - Two-finger gripper
  - Three-finger gripper
  - Articulated five-finger hand
- Effectors for moving
  - Wheels
  - Robot vehicle
  - Chains
  - Legs

Effectors for handling



Two-finger gripper

Example: Parallel-jaw gripper, mounted on robot arm

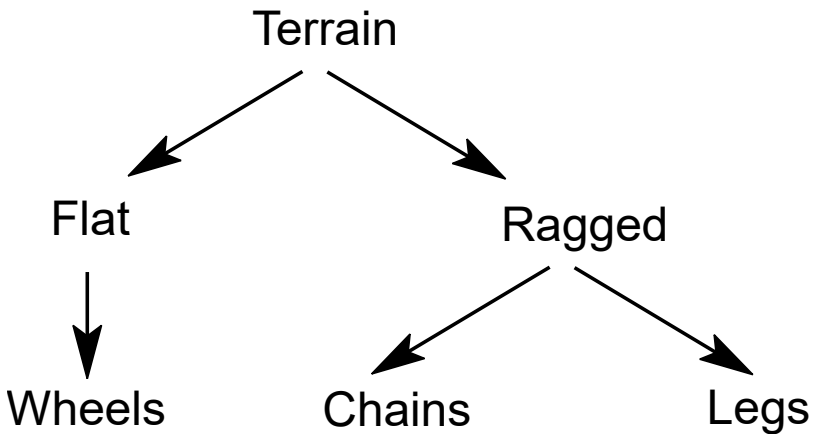
Three-finger gripper

Examples: from Schunk

Articulated five-finger hand

Examples: from DLR

Effectors for moving



Robot vehicle

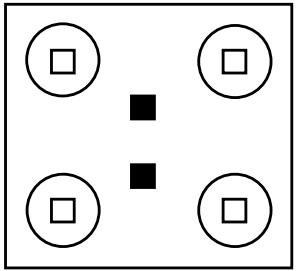
Example: Labmate from TRC.

Robot vehicle

Example: B21 from RWI.

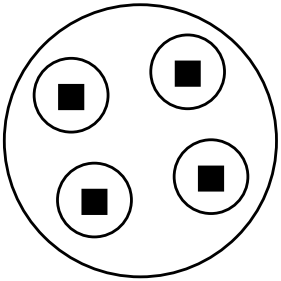
Wheels

Labmate from TRC



2 powered wheels, independently
4 swiveling wheels for stabilisation

B21 from RWI



4 powered and
swiveling wheels

 : non-powered, swiveling

□ : non-powered, non-swiveling

 : powered, swiveling

■ : powered, non-swiveling

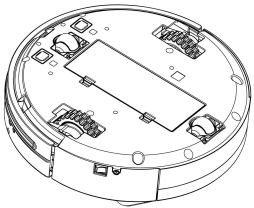
Robot vehicle

Example: Scorpion from Evolution Robotics.

Two powered, non-swiveling wheels; and one non-powered, swiveling wheel.

Robot vehicle

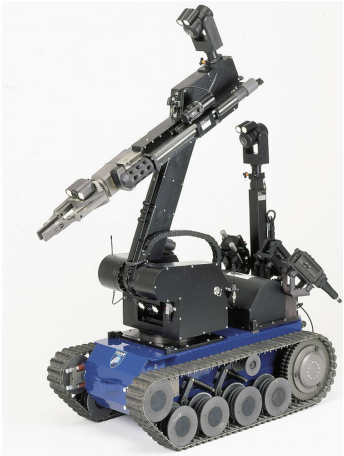
Example: LoCoBot from TrossenRobotics.



Two powered, non-swiveling wheels; two non-powered, non-swiveling wheels.

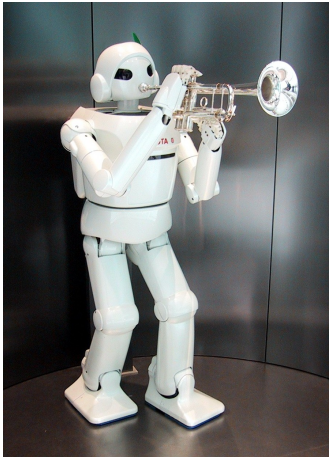
Chains

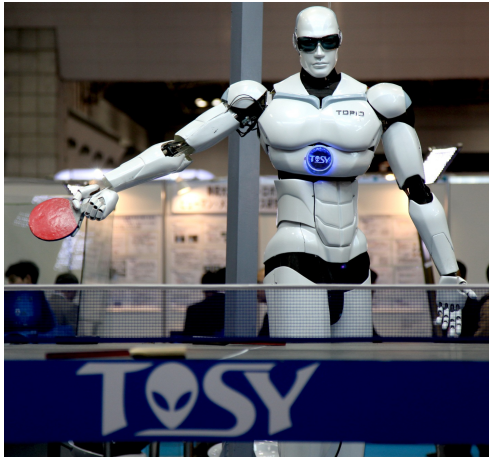
Example: All-terrain vehicles from TELEROB.



Legs

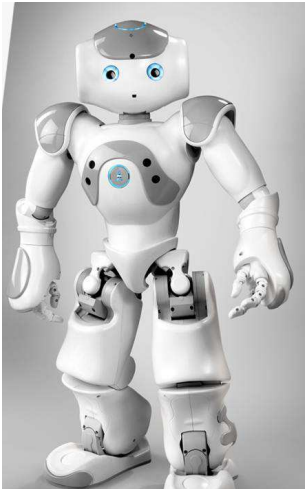
Example: Humanoid robots from Toyota and Tosy.





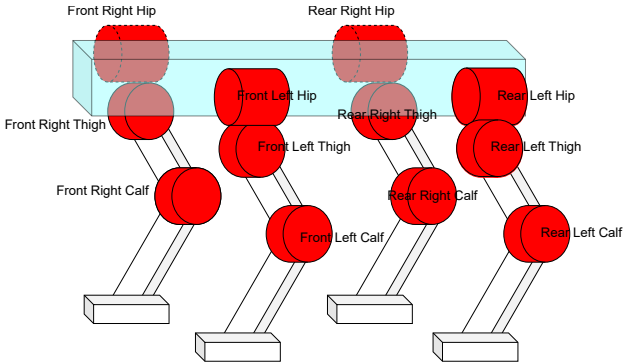
Legs

Example: Humanoid robot Nao from Aldebaran



Legs

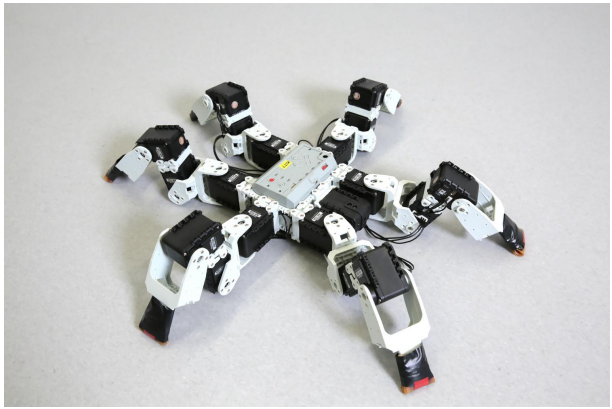
Example: Dog robot Quadruped from Unitree Robotics



Legs

Example: Hexapod crawling robot.

Six legs, each with three revolute joints.



2.4 Equipment

- Robot vehicle
- Robot arm and gripper
- Robot head
- Sensors, cameras
- Motors and drives
- Power engine and supply
- Computers
- Device drivers
- Software platform
- Application programs

3. Sensor System as Basis for Autonomy

Overview:

3.1 Internal sensors, external sensors

Overview:

- Internal sensors
- Clinometer
- Odometry
- External sensors
- Types of media

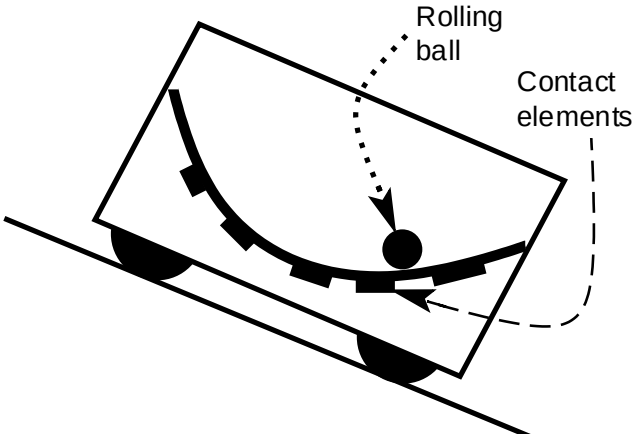
Internal sensors

Deliver internal (proprioceptive) data, e.g.

- error states
- inclination of robot body
- revolute joint angles
- direction (swiveling angle) of the wheels
- speed of the wheels

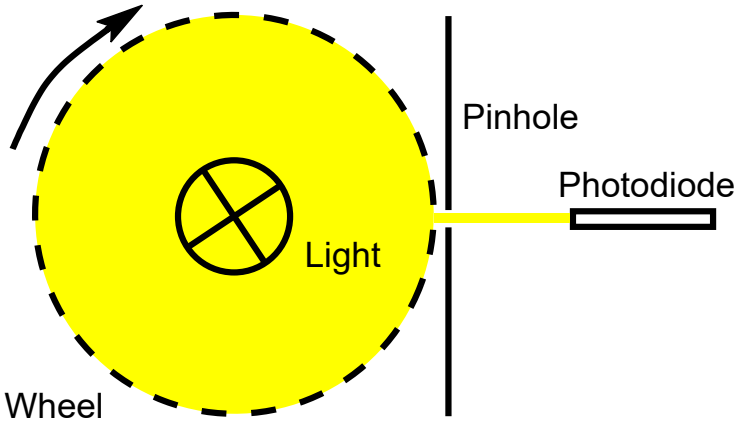
Clinometer

Internal sensors may also be useful to obtain limited information about the environment, indirectly, e.g. slope measurement through clinometer.



Odometry

Determine traveled distance through rounds of wheel circulation.



Odometry

$\boldsymbol{N}$: Number of transparent sections on the rotating wheel
$=$ Number of light impulses per round.

$\boldsymbol{n}$: Total number of impulses in a certain interval of time.

$\boldsymbol{R}$: Radius of the wheel.

$\boldsymbol{s}$: Distance traveled in a certain interval of time.

$$\boldsymbol{s = \frac{n}{N} \cdot 2 \cdot \pi \cdot R}$$

Remark: Inaccuracies due to frictional loss.

External sensors

Deliver environmental (exteroceptive) data, e.g.

- relative position to external landmarks
- structure of the environment (object)
- commands from the environment (e.g. user is pointing to a direction, and the robot should be controlled to move accordingly)

Types of media

Obtain environmental data, dependent on the type of medium between robot and environment.

- Contact
- Air
- Radiation

3.2 Perception via tactile sensors

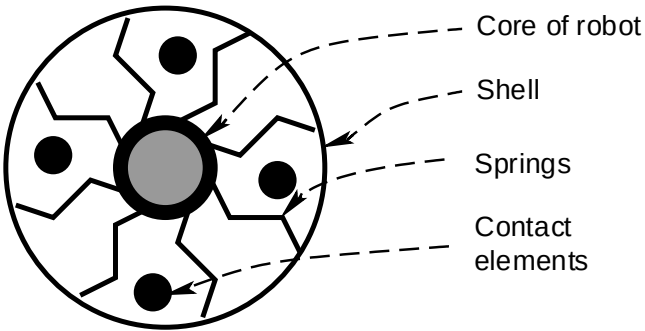
Synonym: Haptic sensors.

Example applications of tactile sensors:

- Collision detection during navigation.
- Stable grasping with robot hand.
- Recognition of shapes or measurement of surface structures, by means of torque sensors and haptic sensors.

Tactile obstacle detection

Collision of the shell of a robot with an environmental object: springs will be compressed, contact points will be touched, stop signal will be triggered.



3.3 Perception via air pressure measurement

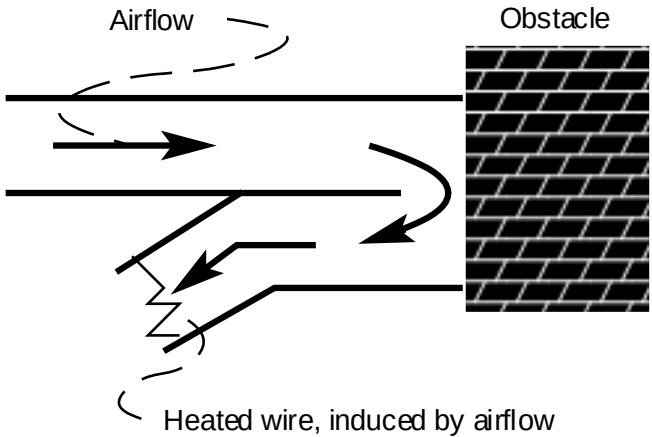
Sending and perceiving air of specific pressure within an interval of time, i.e. waves of airflow.

Overview:

- Pneumatic sensors
- Ultrasound sensors

Pneumatic sensors

Perceives the obstruction of flow of air, caused by an obstacle, e.g. by measuring the heating-up or vibration of a thin wire.



Ultrasound sensors

Ultrasound sensors of robot B21.

Ultrasound sensors

Application in robotics: Ultrasound (US) propagation to navigate, i.e. SONAR (SOund Navigation And Ranging).

- Sound generation through a body's mechanical vibration (e.g. membrane).
- US frequency is over $\mathbf{20\ kHz}$, whereas audible sound is in $\mathbf{16\ Hz - 20\ kHz}$.
- Velocity of propagated US: $v_{sound} \approx \mathbf{3.3 \cdot 10^2\ m/s}$
- Send out an US signal and measure the time of arrival of return signal (time-of-flight).

Ultrasound sensors

Example of calculating distances: Send out an US signal with frequency $\mathbf{20\ kHz}$ for time interval of $\mathbf{1\ ms}$.

- The echo returns after $\boldsymbol{t = 40\ ms}$, i.e. the distance is $\mathbf{6.6\ m}$:
$$\boldsymbol{s = 2 \cdot d = v_{sound} \cdot t = 13.2\ m \Rightarrow d = 6.6\ m}$$
- Per second: $\mathbf{1/41\ ms} \approx$ 24 times/sec measurements.
- The shortest measurable distance is $\mathbf{0.165\ m}$:
$$\boldsymbol{t_{min} = 1\ ms, s = 2 \cdot d = v_{sound} \cdot t_{min} \approx 0.33\ m}$$
$$\boldsymbol{\Rightarrow d \approx 0.165\ m}$$

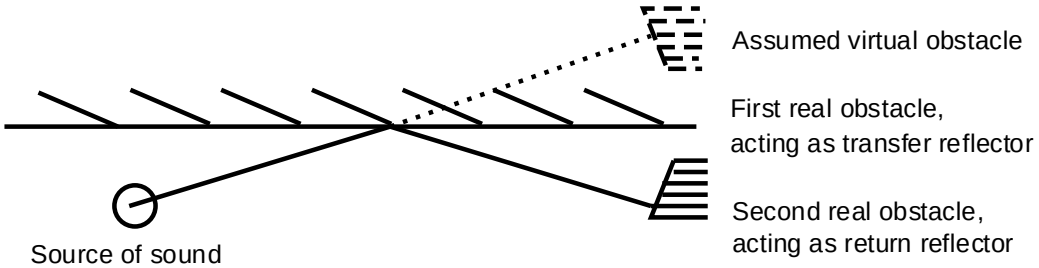
Ultrasound sensors

Problems/Disadvantages of inexpensive US sensors in a lab environment:

- Poor focusing possibility, because the opening angle may be approximately 15 degree.
  Problem of deciding, which fragment of return signal to take.
- Maximum measurable distance, e.g. up to $\mathbf{10}\ \boldsymbol{m}$.
- On rough object surfaces, the returning sound waves are scattered, therefore cause diffusion.

Ultrasound sensors

- On hard and smooth object surfaces, sound waves are reflected to unexpected angles.
  The sound returns after multiple reflections, and an obstacle is assumed at a longer distance.

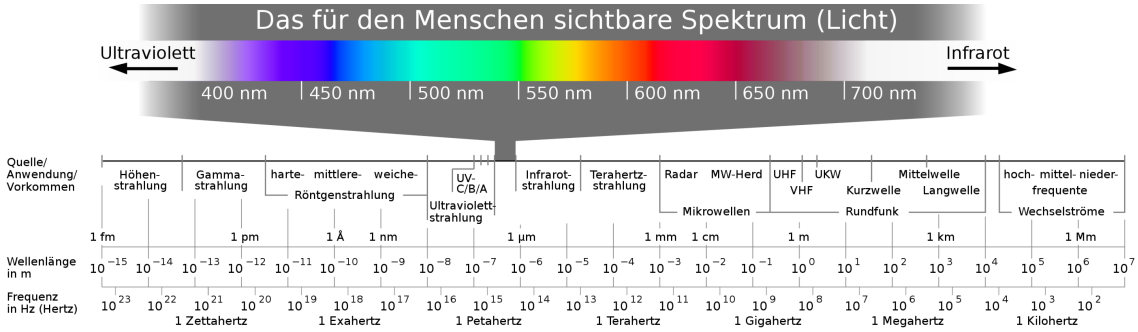


3.4 Perception via light radiation

Overview:

- Electromagnetic spectrum
- Light as part of electromagnetic spectrum
- Light, object, lens and sensor matrix
- Lens systems
- Convex lenses

Electromagnetic spectrum



Wavelength $\boldsymbol{\lambda}$ is between $\mathbf{10^{-7}}\ \boldsymbol{nm}$ and $\mathbf{10^{16}}\ \boldsymbol{nm}$.
Speed of propagation (in vacuum) is $\boldsymbol{v_{elmag} = 3 \cdot 10^{8}\ m/s}$.

Light as part of electromagnetic spectrum

Further features: period $\boldsymbol{T}$ (time interval to propagate over the distance $\boldsymbol{\lambda}$); frequency $\boldsymbol{f_r}$ (in Hertz, $\boldsymbol{Hz}$).

$$\boldsymbol{v_{elmag}/\lambda = 1/T = f_r}$$

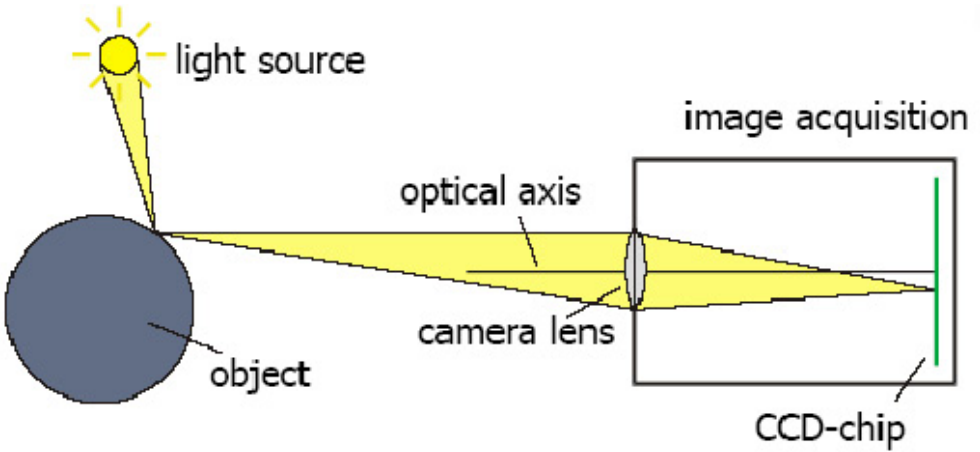
$\Rightarrow$ The range of frequency extends from a few $\boldsymbol{Hz}$ to about $\boldsymbol{10^{24}\ Hz}$.

Light (visible):

Range of wavelengths: $\boldsymbol{390, \cdots, 790\ nm}$

Range of frequencies: $\boldsymbol{3 \cdot 10^{14}, \cdots, 3 \cdot 10^{15}\ Hz}$

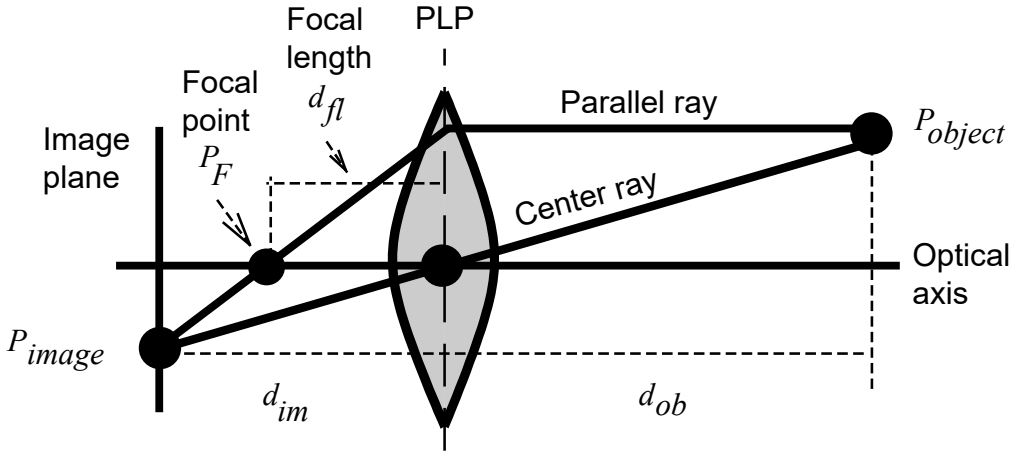
Light, object, lens and sensor matrix



Lens systems

- The light rays pass through a configuration of lenses and will be deflected (refracted) several times, i.e. at every entry and exit of a lens.
- For a simplified mathematical modeling of the optical path, only one lens is assumed.
- Furthermore, the entry and exit refractive planes are approximated as one so-called principal lens plane (PLP), i.e. assuming light will be deflected once.
- The lens is convex to collect light (as opposed to scatter light by a concave lens).

Convex lenses



Lens is positioned, such that all incoming light rays from an object point will be mapped to ONE image point.

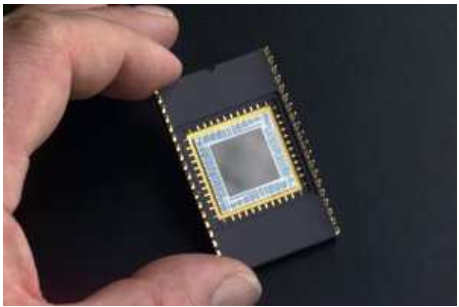
3.5 Industrial digital cameras

Overview:

- CCD cameras
- Functional principle of CCD cameras
- CMOS cameras
- Examples of CMOS chips in consumer cameras

CCD cameras

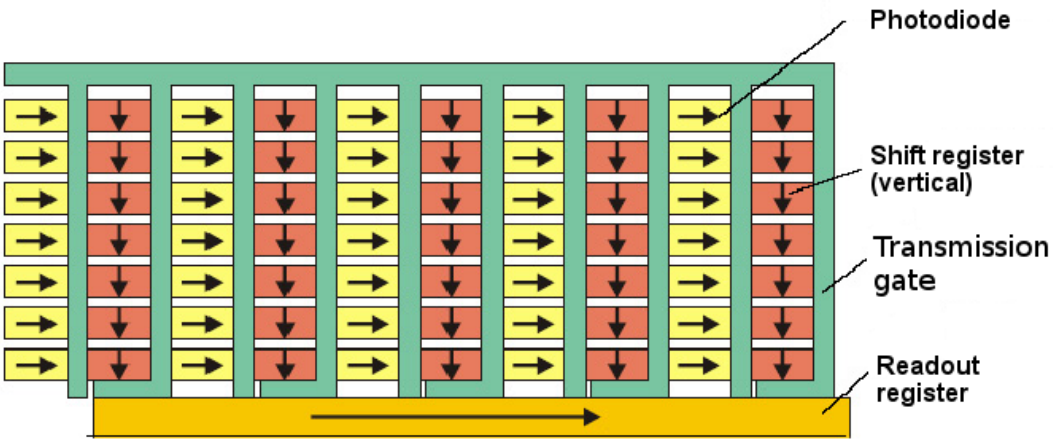
Acronym: **C**harge **C**oupled **D**evice



Functional principle of CCD cameras

- Matrix of light sensitive sensors.
- Recording the incoming light on each sensor in a unit of time.
- Convert light to an electrical signal (photoelectric effect).
- According to the intensity of light, an analog voltage between 0 and 1 Volt (charges) is obtained.
- Sequentially read out the sensors on read and shift registers.
- A/D conversion, i.e. quantisation of the analog voltage value into a discrete value, e.g. natural number between $\mathbf{0}$ and $\mathbf{255}$.

Functional principle of CCD cameras



CMOS cameras

Acronym: **C**omplementary **M**etal-**O**xide-**S**emiconductor

- Sensors are made in CMOS technology to measure light.
- Each sensor has its own integrated circuit for amplification and further processing of the signals.
- Exposure control, contrast correction, A/D conversion are also integrated.
- Advantages compared to CCD cameras: more flexible, lower energy consumption, cheaper.

Examples of CMOS chips in consumer cameras

- Canon EOS 500D:
  chip size $\mathbf{22.3}\ \boldsymbol{mm} \times \mathbf{14.9}\ \boldsymbol{mm}$, number of sensor elements $\mathbf{4752} \times \mathbf{3168} \approx \mathbf{15}$ million, sides ratio $\mathbf{3:2}$, side length of one sensor element $\mathbf{4.7}\ \boldsymbol{\mu m}$.

- Camera of Samsung Galaxy S4:
  chip size $\mathbf{4.6}\ \boldsymbol{mm} \times \mathbf{3.4}\ \boldsymbol{mm}$, number of sensor elements $\mathbf{4128} \times \mathbf{3096} \approx \mathbf{13}$ million, sides ratio $\mathbf{4:3}$, side length of one sensor element $\mathbf{1.12}\ \boldsymbol{\mu m}$.

3.6 Perception via intensive light radiation

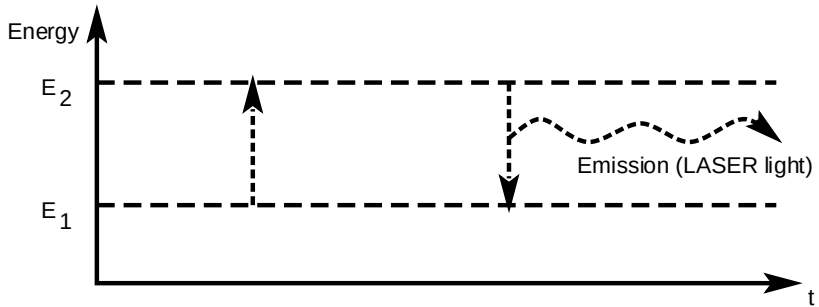
Overview:

- Principle of the LASER
- Characteristics of the LASER
- Approaches for distance measurement
- Distance measurement via time-of-flight
- Distance measurement by triangulation
- Distance measurement from defocus

Principle of the LASER

Acronym: **L**ight **A**mplification by **S**timulated **E**mission of **R**adiation

- Electron is lifted to a higher energy level by energy supply (excitation).
- Through subsequent re-descent to the lower energy level, a light ray is emitted.



Characteristics of the LASER

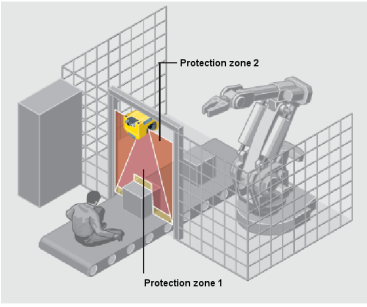
- The emitted light ray is intensive, monochromatic, narrow, polarized.
- Accurate distance measurement is possible.
- Many measurements per time unit are possible due to short-term light impulses.
- Made of e.g. ruby or helium-neon mixture.
- Wavelength e.g. $\boldsymbol{900\ nm}$.

Approaches for distance measurement

- Time-of-flight approach (similar to ultrasound).
- Triangulation approach.
- Defocus approach.
- etc.

Distance measurement via time-of-flight

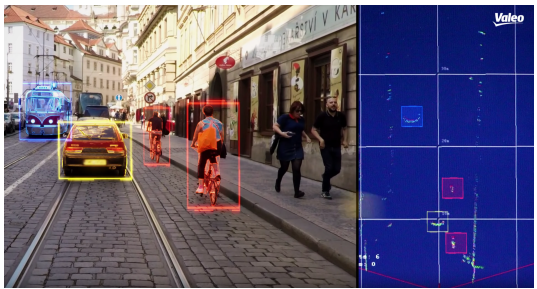
E.g. 2D LASER scanner from company SICK, for safety systems in industrial context



See also LIDAR (LIght Detection And Ranging) method:
`https://en.wikipedia.org/wiki/Lidar`

Distance measurement via time-of-flight

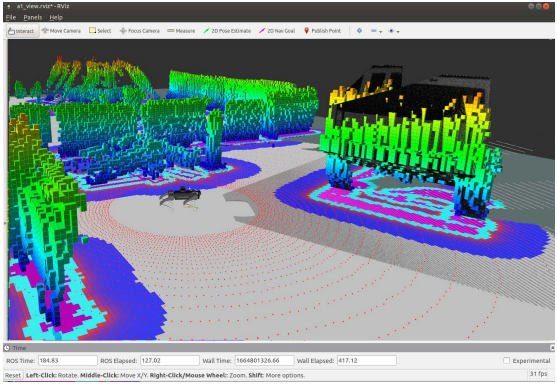
E.g. 2D LASER scanner Scala from company Valeo, for driver assistance systems
`https://www.youtube.com/watch?v=7CWNh_6MBzE`



Distance measurement via time-of-flight

E.g. 3D Multilayer LASER scanner from company Ouster, e.g. for research labs

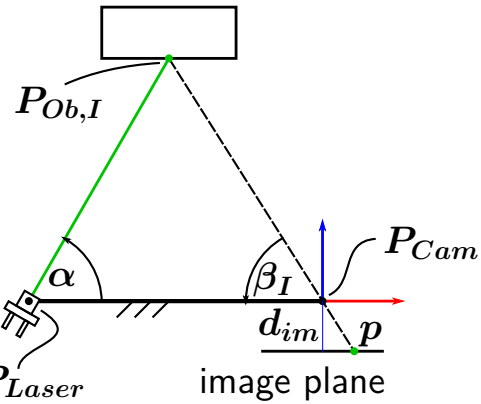
`https://www.mybotshop.de/Ouster-Multi-Layer-3D-LiDAR`

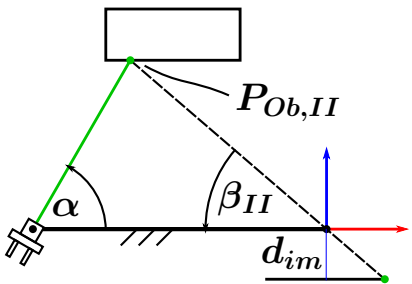


Distance measurement by triangulation

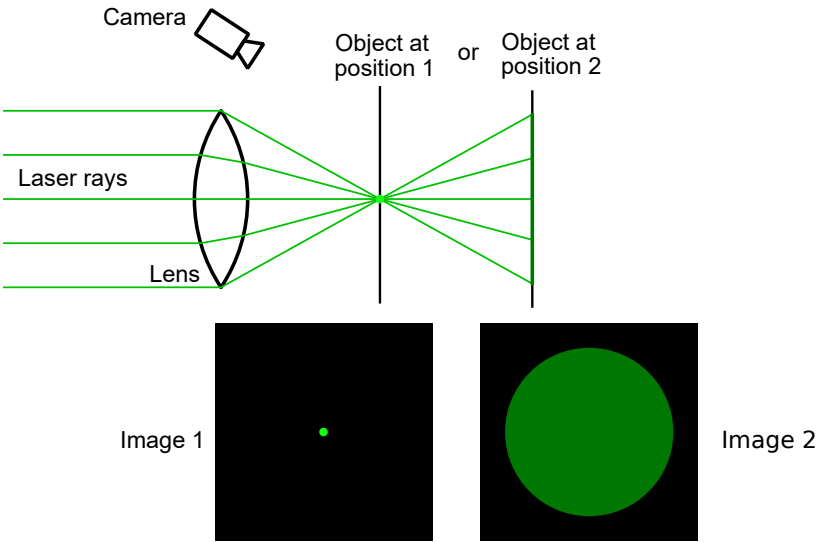
Given: $\boldsymbol{P_{Laser}}$, $\boldsymbol{P_{Cam}}$, $\boldsymbol{\alpha}$, $\boldsymbol{d_{im}}$, image point $\boldsymbol{p}$ of reflected LASER ray.

Quested: $\boldsymbol{P_{Ob}}$





Distance measurement from defocus



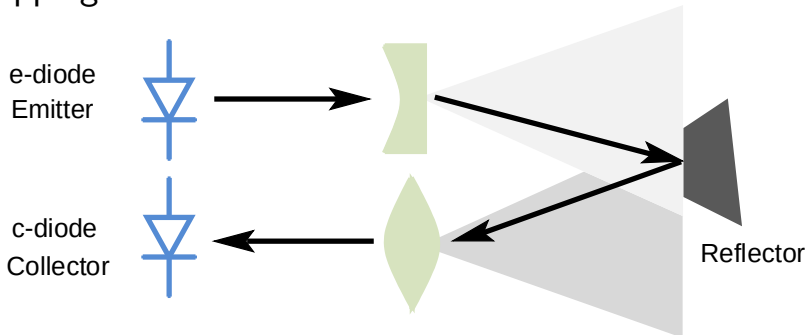
3.7 Perception via infrared light

Overview:

- Infrared diodes
- Application of IR-diodes in robotics
- Relationship between temperature and radiation
- Infrared cameras
- Applications of infrared cameras

Infrared diodes

- An emitting diode sends out infrared radiation through a concave lens.
- A collecting diode receives infrared radiation, reflected by an obstacle, through a convex lens.
- Overlapping area of the IR-cones from e- and c-diode.

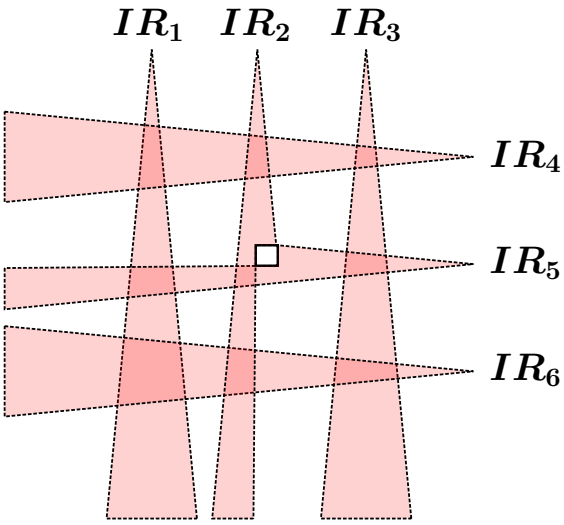


Application of IR-diodes in robotics

Determine position of target objects for grasping:

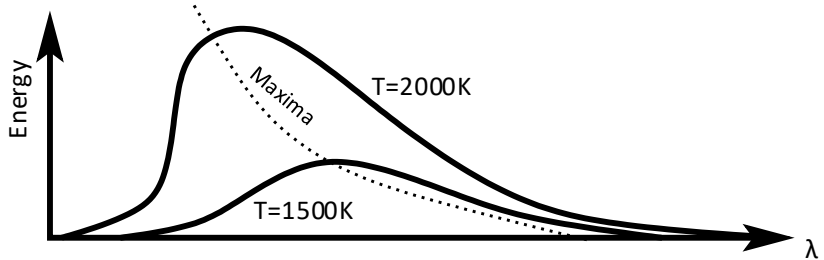
- Several IR systems (pairs of e- and c-diode) are mounted on the gripper.
- Measurement of reflected radiation, to determine if there is an obstacle inside overlapping area of IR-cones of e- and c-diode.
- Through intersection of several overlapping areas, provided by two or more IR-systems, one can obtain more detailed knowledge concerning position and shape of the target object.

Application of IR-diodes in robotics



Relationship between temperature and radiation

- Visible light radiation and infrared light radiation may result from thermal energy.
- With sufficient high temperature, a significant portion of radiation can be emitted in the relevant range of wavelengths.



Infrared cameras

With infrared diodes, artificial IR radiation is sent/received. However by using infrared cameras, natural IR radiation is recorded.

- Perception of an object based on its IR radiation.
- Object is imaged through an IR sensitive sensor matrix.
- The values obtained from the sensor are mapped into temperature values, leading to a color-coded representation.
- Typical range of wavelengths $\mathbf{1 - 12}\ \boldsymbol{\mu m}$, sensor matrix with e.g. $\mathbf{480}$ lines and $\mathbf{640}$ columns, possible range of distances up to a few kilometers.

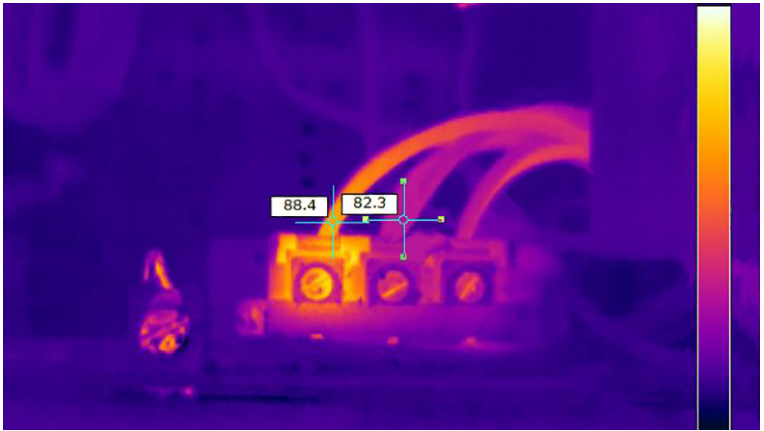
Applications of infrared cameras

Home inspection (heat loss):

URL

Applications of infrared cameras

Material testing:



URL

Applications of infrared cameras

Surveillance:

3.8 Combined color and distance perception

Overview:

- Kinect 1
  Overview, Characteristics, Distance measurement, Exemplary perception
- Kinect 2
  Overview, Exemplary perception
- Realsense
  Overview, Exemplary perception
- ZED 2
  Overview, Exemplary perception

Kinect 1 - Overview

- Developed by PrimeSense and Microsoft.
- Kinect type 1 available since 2009.
- Actually intended to control a video game console (XBOX 360), based on human motion tracking.

Kinect 1 - Characteristics

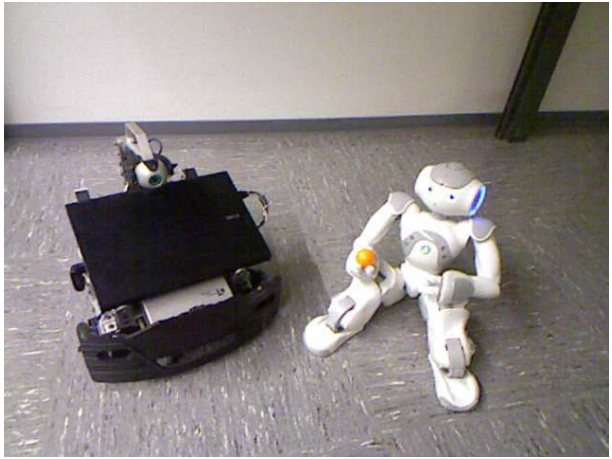
- Color perception through CMOS chip with Bayer color filter (in fig: middle), spatial resolution $\mathbf{640 \times 480}$ pixels at repetition rate $\mathbf{30\ Hz}$, up to $\mathbf{1280 \times 1024}$ pixels at lower repetition rate, intensity resolution $\mathbf{8}$ bit.
- Distance measurement through infrared laser projector (in fig: left), combined with monochrome CMOS sensor chip (in fig: right), resolution $\mathbf{640 \times 480}$ pixels at repetition rate $\mathbf{30\ Hz}$, distance resolution $\mathbf{11}$ bit (i.e. $\mathbf{2048}$ levels), millimeter-accuracy for distances up to $\mathbf{4\ m}$.

Kinect 1 - Distance measurement

- The principle for distance measurement is active triangulation.
- Projection of an IR dot pattern into the room.
- For an expected reference object, its 3D distance (in mm) and the position of back-projected dot pattern in the CMOS image (in pixel) are known.
- Localisation of the back-projected dot pattern from an actual (farther or closer) object in CMOS image.
- From the difference of expected and actual position in the image, 3D object distance can be determined by triangulation.

Kinect 1 - Exemplary perception

Color image:

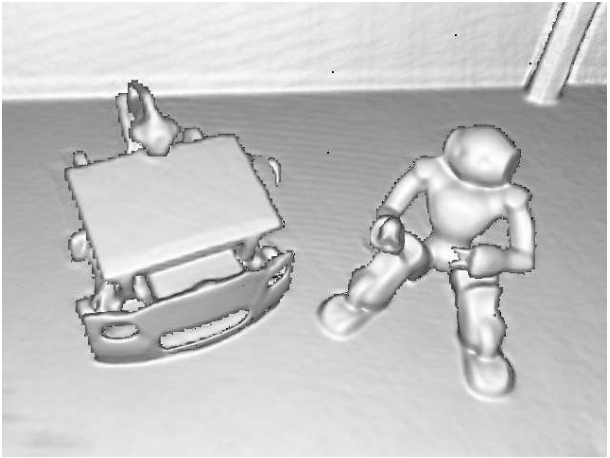


Kinect 1 - Exemplary perception

Distance image:

Kinect 1 - Exemplary perception

Distance image, rendered with Software “Kinect Fusion“:

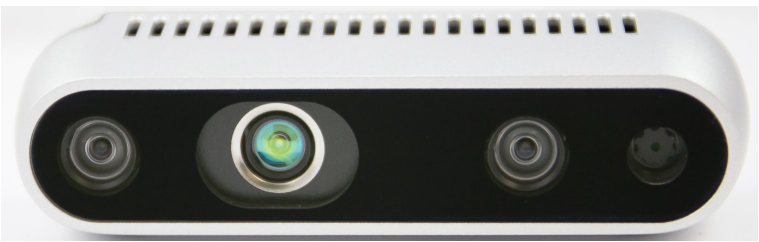


Kinect 2 - Overview

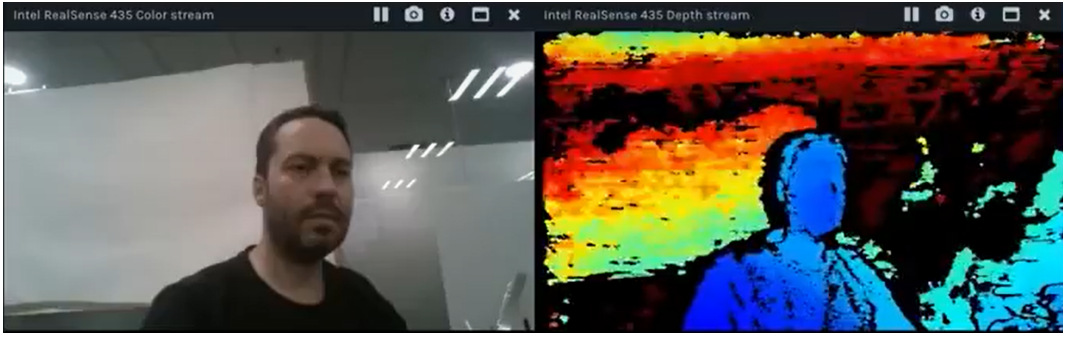
- Kinect type 2 (for XBOX ONE) is available since 2014.
- Distance measurement with time-of-flight approach.
- Greater accuracy at lower distance between sensor and player.

Realsense - Overview

- Developed by Intel.
  `https://www.intelrealsense.com/depth-camera-d435i/`
- Two infrared stereo cameras plus projector, RGB camera.
- SDK and wrappers (e.g. for ROS).
- Depth field of view: $\boldsymbol{H87°, V58°}$, Depth range $\boldsymbol{0.2m - 10m}$.
- HD; Integrated IMU



Realsense - Exemplary perception



(taken from `https://www.youtube.com/watch?v=BVM1wxE9x0E`)

ZED 2 - Overview

- Developed by StereoLabs.
  `https://www.stereolabs.com/zed-2`
- Two stereo cameras, and SDK (camera fusion, depth computation).
- Depth field of view: $\boldsymbol{H110°, V70°}$, Depth range $\boldsymbol{0.2m - 20m}$
- FullHD; Integrated IMU, barometer, magnetometer.

ZED 2 - Exemplary perception

4. Vision-based Control of a Robot Arm

Overview:

4.1 Task and approach at a glance

Overview:

- Hints to literature
- Task description and applications
- Industrial robot and stereo camera
- Approach and advantages

Hints to literature

See Craig (2005) for basics in coordinate systems and pose of objects.

See Hager (1995) for robot arm control in this chapter.

See Hutchinson (1996) and Hashimoto (1993) for „Visual Servoing“ in general.

See Föllinger (1990) for further details in control theory.

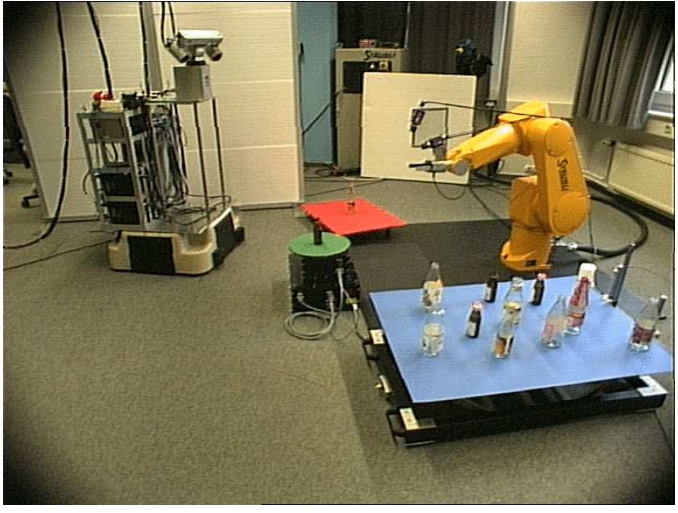
Task description and applications

Task description:

- Robot hand holds an object (the so-called gripper object, e.g. a working tool).
- This gripper object should be moved towards a second object (the so-called target or goal object), in order to take on a certain geometric relation.
- If the target object is mobile, then it should be tracked by the gripper object, in order to keep the desired relation.
- Image pairs taken by a stereo camera are used for vision-based control.

Applications: Assembly or tracking tasks.

Industrial robot and stereo camera



Approach and advantages

Approach: The gripper movement is step-by-step, i.e. continuous control, instead of one-step movement.

Advantages:

- Goal-directed, reliable 3D movement through continuous iterative feedback (servoing).
- Avoidance of precise calibrations, i.e. only approximate camera and camera-robot modeling.
- Possibility of timely reactions to unexpected obstacles.
- Avoidance of expensive 3D reconstructions.

4.2 Coordinate systems, pose of objects

Overview:

- Cartesian coordinate system
- Coordinate systems for robot arm
- Position of a point
- Pose of an object
- Position of an object
- Orientation of an object
- Properties of a rotation matrix
- Position in alternative coordinate systems

Cartesian coordinate system

$$CS\{A\} = \{P_A, (\vec{X}_A, \vec{Y}_A, \vec{Z}_A)\}$$

Example:

$$P_A = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad \vec{X}_A = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \vec{Y}_A = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \vec{Z}_A = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

P_A represents the origin of the coordinate system.

$\vec{X}_A$, $\vec{Y}_A$, $\vec{Z}_A$ are three pairwise orthogonal unit vectors.

$\Rightarrow$ **Cartesian coordinate system.**

Coordinate systems for robot arm

$\{St\}$: **Station (World) coordinate system**,
attached at a certain point in the workspace.

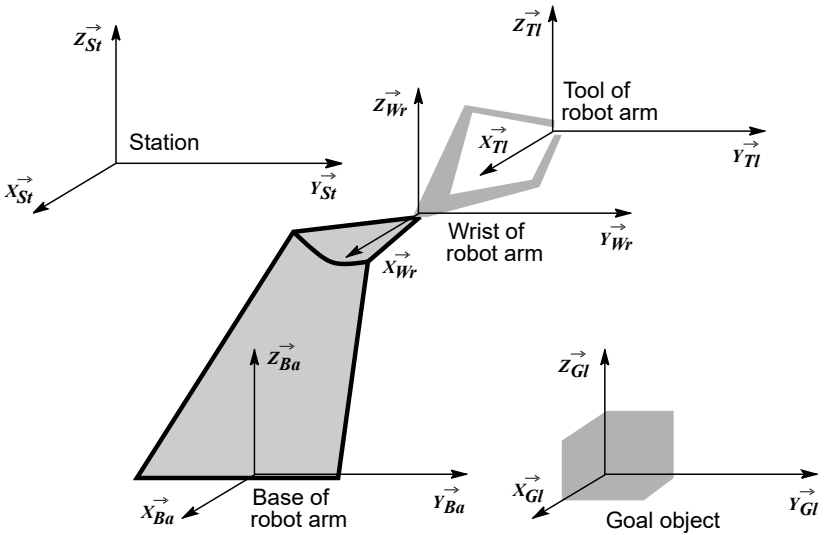
$\{Ba\}$: **Base coordinate system** of the robot arm,
is described relative to $\{St\}$.

$\{Wr\}$: **Wrist coordinate system**,
attached on the last joint of the robot arm,
may be described relative to $\{Ba\}$.

$\{Tl\}$: **Tool coordinate system**,
attached at a certain point of the tool.

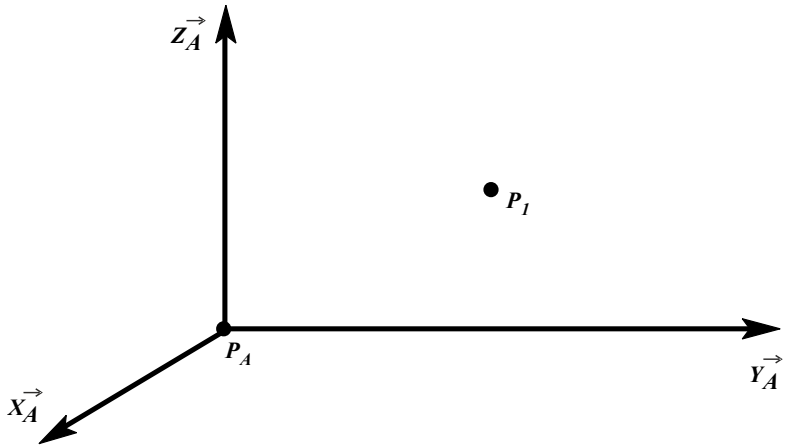
$\{Gl\}$: **Goal coordinate system**,
attached at a certain point of the goal object.

Coordinate systems for robot arm



Position of a point

${}^{A}\boldsymbol{P}_1 = (\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})^{T}$, position vector of a point $\boldsymbol{P}_1$ in coordinate system $\{\boldsymbol{A}\}$.



Pose of an object

Pose subsumes the position and orientation of an object.

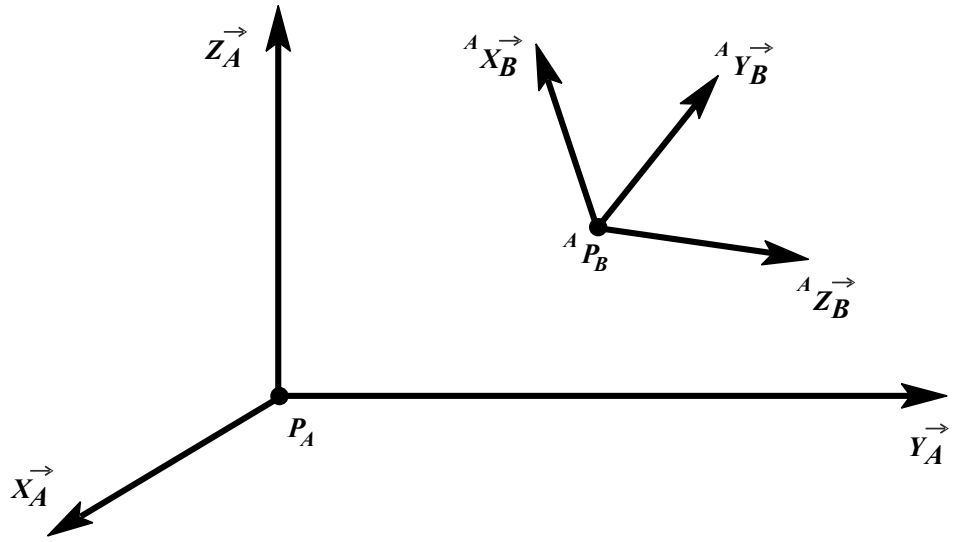
- Coordinate system $\{B\}$ is attached to the object, e.g. Goal coordinate system $\{Gl\}$ to the goal object.
- Representation of the object pose relative to another coordinate system $\{A\}$, e.g. to Station coordinate system $\{St\}$.

Position of an object

A certain point, belonging to the object, defines the origin of the axes of coordinate system $\{B\}$, i.e. ${}^{A}P_{B}$.

For example, the center point of gravity of the object, or a corner point of the boundary of the object.

Orientation of an object



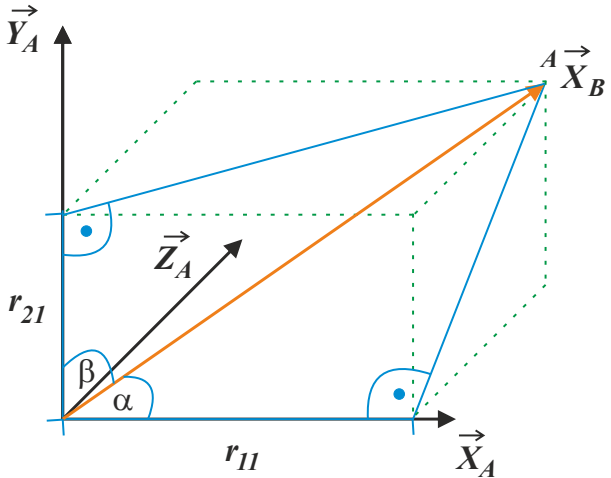
Orientation of an object

$$ {}^{A}_{B}\boldsymbol{R} = ({}^{A}\vec{\boldsymbol{X}}_{B}, {}^{A}\vec{\boldsymbol{Y}}_{B}, {}^{A}\vec{\boldsymbol{Z}}_{B}) = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} $$

Matrix ${}^{A}_{B}\boldsymbol{R}$ describes the rotation of coordinate system $\{B\}$ relative to coordinate system $\{A\}$.
Each column vector has length 1 and describes the direction of an axis of the coordinate system $\{B\}$ relative to the axes of $\{A\}$.
The entries r_{ij} are called directional cosines.

Orientation of an object

$$cos(\alpha) = r_{11}/|{}^{A}\vec{X}_B| = r_{11}\ , \quad cos(\beta) = r_{21}/|{}^{A}\vec{X}_B| = r_{21}$$



Properties of a rotation matrix

Every rotation matrix $\boldsymbol{R}$ has three orthonormal column vectors and has determinant 1.

Theorem concerning rotation matrix:

A matrix $\boldsymbol{R}$ with above properties has the form:

$$\boldsymbol{R} = (\boldsymbol{I_3} - \boldsymbol{S_3})^{-1}(\boldsymbol{I_3} + \boldsymbol{S_3}) \ ,$$

with $\boldsymbol{I_3}$ a $\mathbf{3} \times \mathbf{3}$ unit matrix,

and $\boldsymbol{S_3}$ a $\mathbf{3} \times \mathbf{3}$ skew-symmetric matrix, i.e. $\boldsymbol{S_3} = -\boldsymbol{S_3^T}$,

$$\boldsymbol{S_3} = \begin{pmatrix} \mathbf{0} & -\boldsymbol{s_1} & \boldsymbol{s_2} \\ \boldsymbol{s_1} & \mathbf{0} & -\boldsymbol{s_3} \\ -\boldsymbol{s_2} & \boldsymbol{s_3} & \mathbf{0} \end{pmatrix} \quad \Rightarrow \text{only three parameters.}$$

Properties of a rotation matrix

The nine parameters $r_{11}, ..., r_{33}$ from $\boldsymbol{R}$ include redundancies. Avoidance through following six constraints:

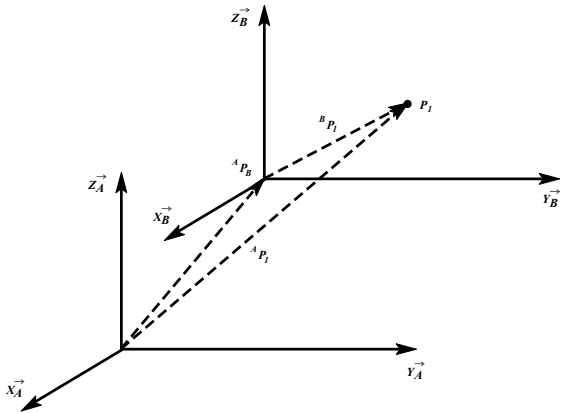
- $\|\vec{\boldsymbol{X}}\| = 1$, $\|\vec{\boldsymbol{Y}}\| = 1$, $\|\vec{\boldsymbol{Z}}\| = 1$;
  $\Rightarrow$ All column vectors have length 1.
- $\vec{\boldsymbol{X}} \circ \vec{\boldsymbol{Y}} = 0$, $\vec{\boldsymbol{Y}} \circ \vec{\boldsymbol{Z}} = 0$, $\vec{\boldsymbol{X}} \circ \vec{\boldsymbol{Z}} = 0$;
  $\Rightarrow$ Dot (scalar) products are 0, i.e. vectors are pairwise orthogonal to each other.

Rotation defined by three independent parameters, e.g. (a) two for rotation axis and one for rotation angle, or (b) three Euler angles Yaw, Pitch, Roll.

Position in alternative coordinate systems

Translation of a coordinate system:

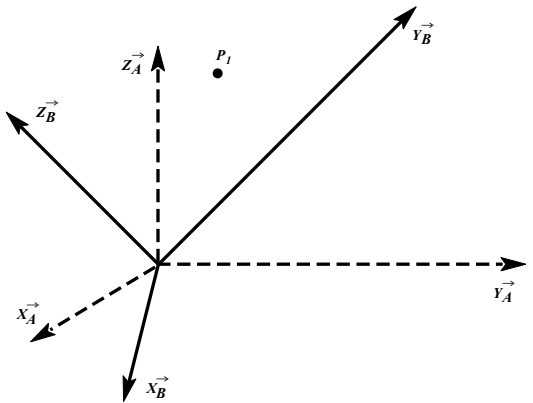
Vector addition: ${}^{A}\boldsymbol{P_1} = {}^{A}\boldsymbol{P_B} + {}^{B}\boldsymbol{P_1}$



Position in alternative coordinate systems

Rotation of a coordinate system:

Multiplication of matrix with vector: ${}^{A}\boldsymbol{P_1} = {}^{A}_{B}\boldsymbol{R} \cdot {}^{B}\boldsymbol{P_1}$



Position in alternative coordinate systems

Translation and rotation of a coordinate system:

Multiplication of matrix with vector and vector addition:

$$ {}^{A}\boldsymbol{P}_1 = {}^{A}_{B}\boldsymbol{R} \cdot {}^{B}\boldsymbol{P}_1 + {}^{A}\boldsymbol{P}_B $$

Alternative matrix form:

$$ \begin{pmatrix} {}^{A}\boldsymbol{P}_1 \\ 1 \end{pmatrix} = \begin{pmatrix} {}^{A}_{B}\boldsymbol{R} & {}^{A}\boldsymbol{P}_B \\ 000 & 1 \end{pmatrix} \cdot \begin{pmatrix} {}^{B}\boldsymbol{P}_1 \\ 1 \end{pmatrix} $$

Here the vector $\begin{pmatrix} {}^{A}\boldsymbol{P}_1 \\ 1 \end{pmatrix}$ represents the homogeneous coordinates with an appended 1.

Position in alternative coordinate systems

Matrix ${}^{A}_{B}\boldsymbol{R}$, vector ${}^{A}\boldsymbol{P}_{B}$, and the components $\{0001\}$ together form the transformation matrix ${}^{A}_{B}\boldsymbol{T}$.

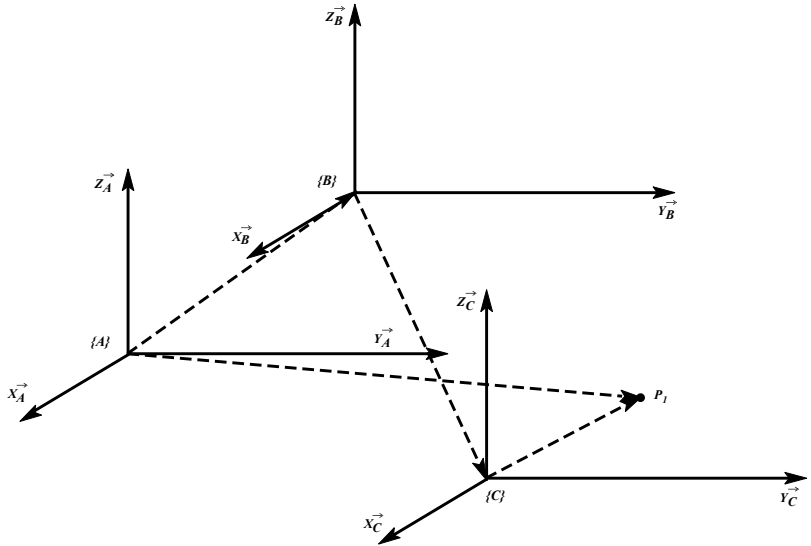
In the following, homogenous point representation is assumed when describing transformation in matrix form:

$$ {}^{A}\boldsymbol{P}_{1} = {}^{A}_{B}\boldsymbol{T} \cdot {}^{B}\boldsymbol{P}_{1} $$

Compact form for combination of transformations:

$$ {}^{A}\boldsymbol{P}_{1} = {}^{A}_{B}\boldsymbol{T} \cdot {}^{B}_{C}\boldsymbol{T} \cdot {}^{C}\boldsymbol{P}_{1} $$

Position in alternative coordinate systems



4.3 Equations for perspective projection

Overview:

- Basic symbols for perspective projection
- Figure of perspective projection
- Equations of perspective projection
- Collinear stereo cameras as special case

Basic symbols for perspective projection

In the following notation, index St is omitted when assuming points or vectors in the Station coordinate system $\{St\}$.

$\boldsymbol{P}_{C_1}^{3D}, \boldsymbol{P}_{C_2}^{3D}$: positions of the two cameras relative to $\{St\}$

$\vec{\boldsymbol{X}}_{C_i}, \vec{\boldsymbol{Y}}_{C_i}, \vec{\boldsymbol{Z}}_{C_i}$: orientation of Camera coordinate systems $\{C_i\}$, $i \in \{1, 2\}$, relative to $\{St\}$

$\boldsymbol{d_{im}}$: distance between lens plane and sensor plane of both cameras (uniform), respectively

$\boldsymbol{s_{xy}}$: Side length of a quadratic pixel (in mm per pixel)

Basic symbols for perspective projection

$\boldsymbol{P}_{*}^{\boldsymbol{3D}}$: point (position) of target object

$\boldsymbol{P}^{\boldsymbol{3D}}$: point (position) of gripper object

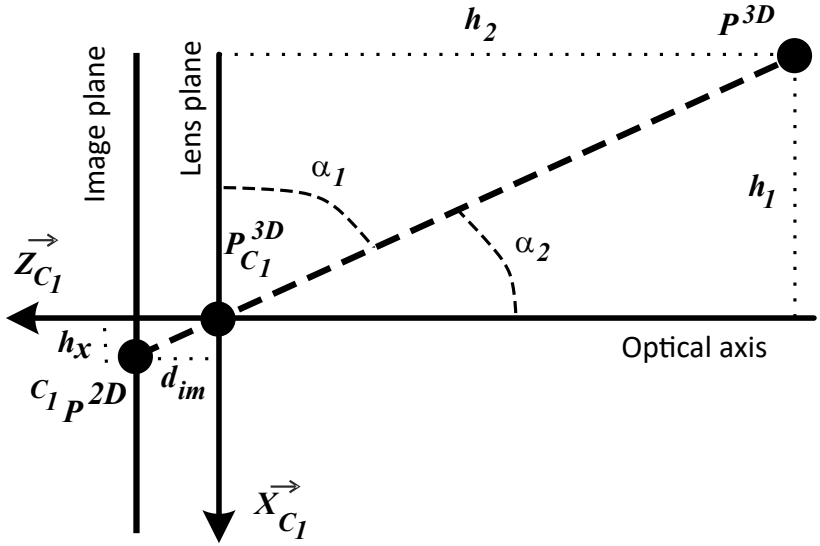
${}^{C_i}\boldsymbol{P}_{*}^{\boldsymbol{2D}}$: image point (mm) of target object in camera $\boldsymbol{i}$

${}^{C_i}\boldsymbol{P}^{\boldsymbol{2D}}$: image point (mm) of gripper object in camera $\boldsymbol{i}$

${}^{I_i}\boldsymbol{P}_{*}^{\boldsymbol{2D}}$: image point (pixel) of target object in camera $\boldsymbol{i}$

${}^{I_i}\boldsymbol{P}^{\boldsymbol{2D}}$: image point (pixel) of gripper object in camera $\boldsymbol{i}$

Figure of perspective projection



Equations of perspective projection

Definitions, to be used:

$$\boldsymbol{P^{3D}} := \begin{pmatrix} \boldsymbol{X_P} \\ \boldsymbol{Y_P} \\ \boldsymbol{Z_P} \end{pmatrix}, \quad \boldsymbol{P^{3D}_{C_1}} := \begin{pmatrix} \boldsymbol{X_{C_1}} \\ \boldsymbol{Y_{C_1}} \\ \boldsymbol{Z_{C_1}} \end{pmatrix}$$

$$\overset{St}{C_1}\boldsymbol{R} := (\vec{\boldsymbol{X}}_{C_1}, \vec{\boldsymbol{Y}}_{C_1}, \vec{\boldsymbol{Z}}_{C_1}) := \begin{pmatrix} \boldsymbol{r_{11}} & \boldsymbol{r_{12}} & \boldsymbol{r_{13}} \\ \boldsymbol{r_{21}} & \boldsymbol{r_{22}} & \boldsymbol{r_{23}} \\ \boldsymbol{r_{31}} & \boldsymbol{r_{32}} & \boldsymbol{r_{33}} \end{pmatrix}$$

$\overset{St}{C_1}\boldsymbol{R}$: orientation of $\{\boldsymbol{C_1}\}$ relative to $\{\boldsymbol{St}\}$.

Coordinates of an image point from 1st camera,

in mm: ${}^{C_1}\boldsymbol{P^{2D}} := \begin{pmatrix} {}^{C_1}\boldsymbol{x_P} \\ {}^{C_1}\boldsymbol{y_P} \end{pmatrix}$, and in pixel: ${}^{I_1}\boldsymbol{P^{2D}} := \begin{pmatrix} {}^{I_1}\boldsymbol{x_P} \\ {}^{I_1}\boldsymbol{y_P} \end{pmatrix}$.

Similar definitions from above for 2nd camera.

Equations of perspective projection

$$(\boldsymbol{P^{3D}} - \boldsymbol{P^{3D}_{C_1}}) \circ \vec{\boldsymbol{X}}_{C_1} =$$

$$||\boldsymbol{P^{3D}} - \boldsymbol{P^{3D}_{C_1}}|| \cdot ||\vec{\boldsymbol{X}}_{C_1}|| \cdot \boldsymbol{cos}(\boldsymbol{\alpha_1}) = \boldsymbol{h_1}$$

$$(\boldsymbol{P^{3D}} - \boldsymbol{P^{3D}_{C_1}}) \circ \vec{\boldsymbol{Z}}_{C_1} =$$

$$||\boldsymbol{P^{3D}} - \boldsymbol{P^{3D}_{C_1}}|| \cdot ||\vec{\boldsymbol{Z}}_{C_1}|| \cdot \boldsymbol{cos}(\boldsymbol{\alpha_2}) = \boldsymbol{h_2}$$

with $||\vec{\boldsymbol{Z}}_{C_1}|| = ||\vec{\boldsymbol{X}}_{C_1}|| = \mathbf{1}$

Hint: ${}^{St}_{C_1}\boldsymbol{R} = \left({}^{C_1}_{St}\boldsymbol{R}\right)^{\boldsymbol{T}}$, and operation $\circ$ is commutative.

Equations of perspective projection

Calculate $\boldsymbol{X}$-coord. of image point (1st camera) in pixel:

$$\boldsymbol{tan(\alpha_2)} = \frac{h_x}{d_{im}} = \frac{h_1}{h_2} = \frac{(P^{3D} - P^{3D}_{C_1}) \circ \vec{X}_{C_1}}{(P^{3D} - P^{3D}_{C_1}) \circ \vec{Z}_{C_1}}$$

$$\Rightarrow h_x = {}^{C_1}x_P = d_{im} \cdot \frac{(P^{3D} - P^{3D}_{C_1}) \circ \vec{X}_{C_1}}{(P^{3D} - P^{3D}_{C_1}) \circ \vec{Z}_{C_1}} \Rightarrow$$

$$g_1(P^{3D}) := {}^{I_1}x_P := \frac{{}^{C_1}x_P}{s_{xy}} := \frac{d_{im}}{s_{xy}} \cdot \frac{(P^{3D} - P^{3D}_{C_1}) \circ \vec{X}_{C_1}}{(P^{3D} - P^{3D}_{C_1}) \circ \vec{Z}_{C_1}}$$

Calculate $\boldsymbol{Y}$-coord. of image point (1st camera) in pixel:

$$g_2(P^{3D}) := {}^{I_1}y_P := \frac{{}^{C_1}y_P}{s_{xy}} := \frac{d_{im}}{s_{xy}} \cdot \frac{(P^{3D} - P^{3D}_{C_1}) \circ \vec{Y}_{C_1}}{(P^{3D} - P^{3D}_{C_1}) \circ \vec{Z}_{C_1}}$$

Equations of perspective projection

Calculate the image point (2nd camera) in pixel:

$$g_3(\boldsymbol{P}^{3D}) := {}^{I_2}\boldsymbol{x}_P := \frac{d_{im}}{s_{xy}} \cdot \frac{(\boldsymbol{P}^{3D} - \boldsymbol{P}^{3D}_{C_2}) \circ \vec{\boldsymbol{X}}_{C_2}}{(\boldsymbol{P}^{3D} - \boldsymbol{P}^{3D}_{C_2}) \circ \vec{\boldsymbol{Z}}_{C_2}}$$

$$g_4(\boldsymbol{P}^{3D}) := {}^{I_2}\boldsymbol{y}_P := \frac{d_{im}}{s_{xy}} \cdot \frac{(\boldsymbol{P}^{3D} - \boldsymbol{P}^{3D}_{C_2}) \circ \vec{\boldsymbol{Y}}_{C_2}}{(\boldsymbol{P}^{3D} - \boldsymbol{P}^{3D}_{C_2}) \circ \vec{\boldsymbol{Z}}_{C_2}}$$

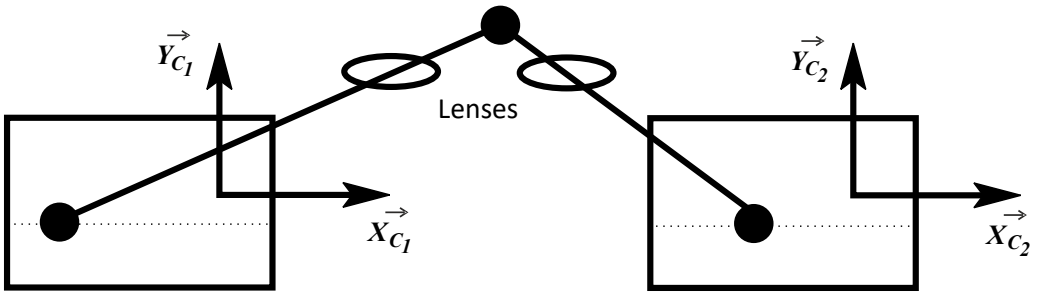
Remark: The quotients of two dot products, involved in all four definitions, have typically the following form,

e.g. within g_1: $\frac{((X_P - X_{C_1})r_{11} + (Y_P - Y_{C_1})r_{21} + (Z_P - Z_{C_1})r_{31})}{((X_P - X_{C_1})r_{13} + (Y_P - Y_{C_1})r_{23} + (Z_P - Z_{C_1})r_{33})}$

They reveal included position and orientation parameters of the two cameras.

Collinear stereo cameras as special case

If $\vec{Y}_{C_1}$ parallel with $\vec{Y}_{C_2}$, and $\vec{X}_{C_1}$ collinear with $\vec{X}_{C_2}$, then ${}^{C_1}\boldsymbol{y}_P = {}^{C_2}\boldsymbol{y}_P$, so the function $\boldsymbol{g_4}$ is redundant.



Collinear stereo cameras as special case

Projection of point $\boldsymbol{P^{3D}}$, using both cameras:

$$\boldsymbol{H^{2D}} := (\boldsymbol{^{I_1}x_P}, \boldsymbol{^{I_1}y_P}, \boldsymbol{^{I_2}x_P}) :=$$

$$\boldsymbol{g}(\boldsymbol{P^{3D}}) := (\boldsymbol{g_1}(\boldsymbol{P^{3D}}), \boldsymbol{g_2}(\boldsymbol{P^{3D}}), \boldsymbol{g_3}(\boldsymbol{P^{3D}}))$$

4.4 Movements in 3D vs. changes in 2D

Overview:

- Basics to Jacobi matrix
- Jacobi matrix for projection function
- Features assumed to be known

Basics to Jacobi matrix

Given n functions $(f_1, ..., f_n) =: f$, each of m variables.

$$y_1 := f_1(x_1, ..., x_m), ..., y_n := f_n(x_1, ..., x_m)$$

For short: $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$

Let $a_1, ..., a_m$ be specific values for $x_1, ..., x_m$, and $\acute{a}_1, ..., \acute{a}_m$ the instantaneous velocities at a certain point in time.

In our case, the velocities are due to the movements of a robot hand in time, i.e. velocity of the moving hand.

Basics to Jacobi matrix

Theorem concerning Jacobi matrix:

Let $b_1, ..., b_n$ be the function values for the input values $a_1, ..., a_m$, and let $\acute{b_1}, ..., \acute{b_n}$ be the velocities, then the Jacobi matrix represents the relation:

$$\begin{pmatrix} \acute{b_1} \\ \cdot \\ \cdot \\ \cdot \\ \acute{b_n} \end{pmatrix} = J_f(a_1, ..., a_m) \cdot \begin{pmatrix} \acute{a_1} \\ \cdot \\ \cdot \\ \cdot \\ \acute{a_m} \end{pmatrix} \quad ; \quad \text{with}$$

Basics to Jacobi matrix

$$J_f(a_1, ..., a_m) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(a_1, ..., a_m) & ..... & \frac{\partial f_1}{\partial x_m}(a_1, ..., a_m) \\ . & ..... & . \\ . & ..... & . \\ . & ..... & . \\ \frac{\partial f_n}{\partial x_1}(a_1, ..., a_m) & ..... & \frac{\partial f_n}{\partial x_m}(a_1, ..., a_m) \end{pmatrix}$$

Jacobi matrix $J_f(a_1, ..., a_m)$ contains all partial derivations of function f, computed at a certain vector $(a_1, ..., a_m)$ of input values.

Basics to Jacobi matrix

Proof (sketch):

First consider the special case $\boldsymbol{f : \mathbb{R} \to \mathbb{R}}$

Can be represented as: $\boldsymbol{f(x) = \underbrace{T_l(x)}_{\text{Taylor polynomial}} + \underbrace{R_l(x)}_{\text{Rest}}}$

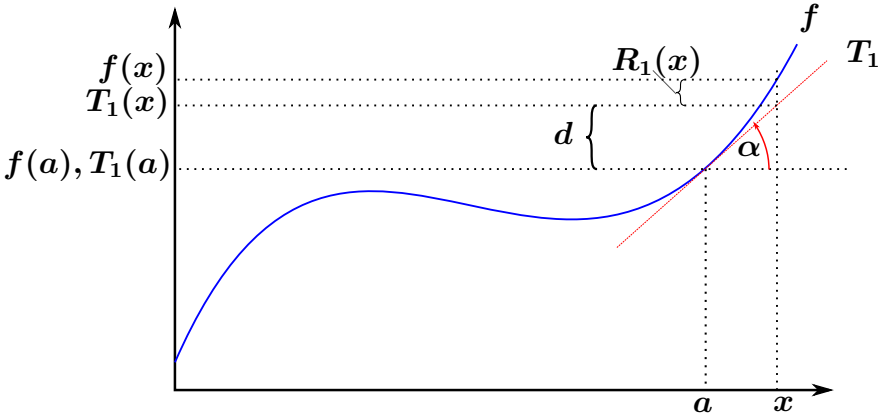
Definition of Taylor polynomial of order $\boldsymbol{l}$:

$$\boldsymbol{T_l(x) = \sum_{k=0}^{l} \frac{f^{(k)}(a)}{k!} \cdot (x-a)^k}$$

Taylor polynomial of order $\boldsymbol{l = 1}$:

$$\boldsymbol{T_1(x) = f(a) + \overbrace{\underbrace{f^{(1)}(a)}_{tan(\alpha)} \cdot (x-a)}^{d}}$$

Basics to Jacobi matrix



$tan(\alpha) = \frac{d}{x-a}$. For x close to a: $T_1(x) \approx f(x)$

Applied: $f(x) - f(a) \approx f^{(1)}(a) \cdot (x - a)$

Basics to Jacobi matrix

Now take the general case $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$

$$\boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) - \boldsymbol{f}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m) \approx$$

$$\boldsymbol{J}_f(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m) \cdot \begin{pmatrix} \boldsymbol{x}_1 - \boldsymbol{a}_1 \\ \ldots \\ \boldsymbol{x}_m - \boldsymbol{a}_m \end{pmatrix},$$

with $\boldsymbol{J}_f \in \mathbb{R}^{n \times m}$.

Jacobi matrix for projection function

Jacobi matrix for projection function $\boldsymbol{g}$:

$$\boldsymbol{J_g} := \begin{pmatrix} \frac{\partial g_1}{\partial X_P} & \frac{\partial g_1}{\partial Y_P} & \frac{\partial g_1}{\partial Z_P} \\ \frac{\partial g_2}{\partial X_P} & \frac{\partial g_2}{\partial Y_P} & \frac{\partial g_2}{\partial Z_P} \\ \frac{\partial g_3}{\partial X_P} & \frac{\partial g_3}{\partial Y_P} & \frac{\partial g_3}{\partial Z_P} \end{pmatrix}$$

As an example, see the partial derivation $\frac{\partial g_1}{\partial X_P}$:

$$g_1(P^{3D}) := \frac{d_{im}}{s_{xy}} \cdot \frac{((X_P - X_{C_1}) r_{11} + (Y_P - Y_{C_1}) r_{21} + (Z_P - Z_{C_1}) r_{31})}{((X_P - X_{C_1}) r_{13} + (Y_P - Y_{C_1}) r_{23} + (Z_P - Z_{C_1}) r_{33})} = \frac{d_{im}}{s_{xy}} \cdot \frac{h_1}{h_2}$$

$$\frac{\partial g_1}{\partial X_P} = \frac{d_{im}}{s_{xy}} \cdot \frac{(r_{11} \cdot h_2 - r_{13} \cdot h_1)}{h_2^2}$$

Jacobi matrix for projection function

Similar for other components.

Finally, all these leads to the state vector differential equation,

$$\boldsymbol{V_{P^{2D}} = J_g(P^{3D}) \cdot V_{P^{3D}}} \ ,$$

with $\boldsymbol{V_{P^{3D}}}$, the velocity vector of tool movement by the robot in 3D, and $\boldsymbol{V_{P^{2D}}}$, the velocity vector of the induced change in the 2D stereo image.

For vision-based control, the Jacobi matrix must be inverted.

Features assumed to be known

- Distance $\boldsymbol{d_{im}}$ between lens and image, and pixel-millimeter relation $\boldsymbol{s_{xy}}$.
- Position and orientation of both cameras, i.e. $\boldsymbol{P^{3D}_{C_1}}$, $\boldsymbol{{}^{St}_{C_1}R}$, $\boldsymbol{P^{3D}_{C_2}}$, $\boldsymbol{{}^{St}_{C_2}R}$.
- Spatial relation between Base coordinate system $\boldsymbol{\{Ba\}}$ of robot arm and Station coordinate system $\boldsymbol{\{St\}}$, i.e. $\boldsymbol{\{{}^{St}P_{Ba}, {}^{St}_{Ba}R\}}$.
- Direct kinematics of the robot arm and all joint angles, for computing the pose of the tool, i.e. $\boldsymbol{{}^{Ba}P^{3D}}$.

4.5 Basic control methods

A controller defines a function to map a time-dependent control difference $\boldsymbol{D}(\boldsymbol{t})$ (i.e. difference between actual system's state/measurement and target system's state/measurement) into a time-dependent correction $\boldsymbol{C}(\boldsymbol{t})$ of control variables (i.e. for movement of an actuator).

There are three basic types of controller:

- Proportional controller (P-controller)
- Integral controller (I-controller)
- Derivative controller (D-controller)

These types of controllers can be combined, e.g. into a PI-controller or a PID-controller.

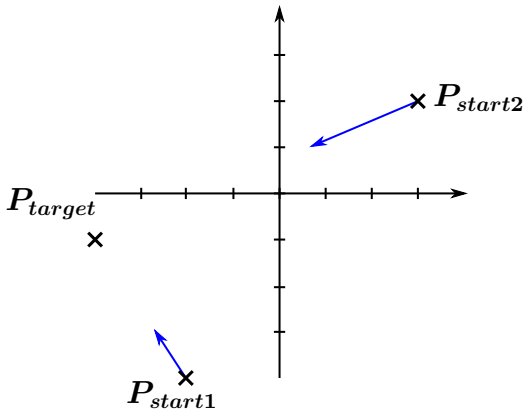
Proportional controller

Control function: $C(t) := \mu_1 \cdot D(t)$

- Reaction with an amount $C(t)$, which is proportional to control difference $D(t)$.
- Parameter μ_1 for values in the unit interval, to be set by the user.
- When μ_1 is big $\Rightarrow$ maybe ignoring unexpected, dangerous situations,
  when μ_1 is small $\Rightarrow$ high resource consumption due to high-frequent situation analysis, and slow behaviour.
- Hypersensitive behaviour in case of erroneous measurements.

Proportional controller

Example: First step of a robot moving towards $\boldsymbol{P_{target}}$, starting at position $\boldsymbol{P_{start1}}$ respectively at $\boldsymbol{P_{start2}}$, assuming $\boldsymbol{\mu_1 = 0.33}$. Step size $\boldsymbol{C(t)}$ is $\boldsymbol{33\%}$ of $\boldsymbol{D(t)}$.



Integral controller

Control function: $C(t) := \mu_2 \cdot \int_{t_a}^{t_b} D(t)dt$

- Takes into account a sequence of control differences from the past, i.e. t_b is the current point of time and t_a is a past point of time in the previous sequence of actions.
- Suppression of hypersensitive reaction in case of erroneous (outlier) measurements.
- Parameter μ_2 for values in the unit interval, to be set by the user.
- Balanced behaviour.

Derivative controller

Control function: $C(t) := \mu_3 \cdot \frac{\partial D(t)}{\partial t}$

- Estimation of the recent change of control differences, e.g. decreasing or increasing control differences.
- E.g. relevant for tracking a moving target object (as opposed to approaching an immobile object).
- Parameter μ_3 for values in the unit interval, to be set by the user.
- Foresighted behaviour.

Combination of different controller types

Control function of PID-controller:

$$C(t) := \mu_1 \cdot D(t) + \mu_2 \cdot \int_{t_a}^{t_b} D(t)dt + \mu_3 \cdot \frac{\partial D(t)}{\partial t}$$

- Parameters μ_1, μ_2, μ_3 for an individual weighting of the three types of controllers, relevant for the actual application, e.g. PI-controller for moving a gripper towards an immobile target object.
- In general, the parameters have to be multiplied by matrices in order to map the space of control differences into the space of actuator changes.

4.6 Vision-based control of a robot gripper

Overview:

- Control function for visual gripper servoing
- Gray-level corners via derivative filters
- Gray-level corners via SUSAN
- Experiments of visual gripper servoing

Control function for visual gripper servoing

- At time $\boldsymbol{t}$, the robot arm is in a state, such that the gripper object is at position $\boldsymbol{P^{3D}(t)}$. This position is available via direct kinematics.
- The projected gripper object in the two stereo images can be extracted via image processing (see later in this section) leading to the vector $\boldsymbol{H^{2D}(t)}$.
- The target object is initially at $\boldsymbol{P^{3D}(t)}$ too, but will then be carried manually to a position $\boldsymbol{P_*^{3D}}$, which is unknown to the robot system.
- However, from the two stereo images the target object can be extracted leading to the vector $\boldsymbol{H_*^{2D}}$.
- Task: The robot arm should move its gripper object to the new position of the target object.

Control function for visual gripper servoing

- Control difference in the 3D coordinate system:

$$\boldsymbol{D^{3D}(t) := P_*^{3D} - P^{3D}(t)}$$

- Control difference in the stereo images:

$$\boldsymbol{D^{2D}(t) := H_*^{2D} - H^{2D}(t)}$$

- $\boldsymbol{P^{3D}(t)}$ must be properly moved, but only $\boldsymbol{D^{2D}(t)}$ is given.
- Task: Determine a sequence of appropriate values for a time-dependent 3D vector $\boldsymbol{C(t)}$ of corrective variables for gradually moving the gripper to the target.

Control function for visual gripper servoing

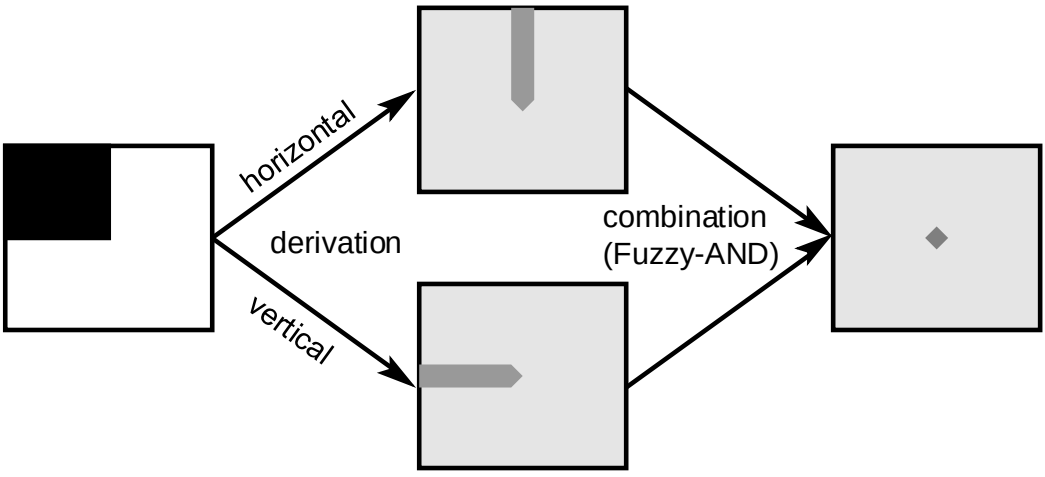
$$\boldsymbol{C(t) := J_g^{-1}(P^{3D}(t)) \cdot \left(\mu_1 \cdot D^{2D}(t) + \mu_2 \cdot \int_{t_a}^{t_b} D^{2D}(t) dt \right)}$$

- The control function is realised as a PI-controller.
- The weighted three-dimensional sum is multiplied by the inverted Jacobi matrix for the projection function $\boldsymbol{g}$, which in turn is applied at the current position $\boldsymbol{P^{3D}(t)}$ of the gripper object.
- Via this multiplication the difference between gripper and target object in the stereo images is mapped into appropriate values for the time-dependent 3D vector $\boldsymbol{C(t)}$ of corrective variables.

Gray-level corners via derivative filters

- Derivation of image function to focus the attention to significant changes of gray-levels.
- Combination of the results of two orthogonally applied derivative filters.
- Evidence for a gray-level edge in both directions indicates a gray-level corner.
- Generalisation is needed when the relevant contour segments have arbitrary orientations.

Gray-level corners via derivative filters



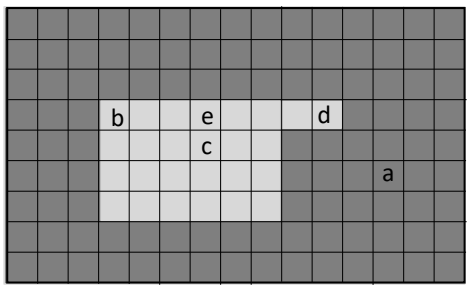
Gray-level corners via SUSAN

Acronym: Smallest Univalue Segment Assimilating Nucleus (SUSAN)

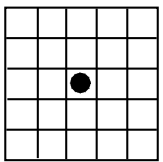
- A „window“ is moved successively over the image.
- At each position in the image, count the number of pixels within the window having the same or similar gray-level as the center pixel.
- The result is a new image with pixel values defined by the determined counter values.
- Binarisation of the resulting image by applying a threshold, i.e. „black“(„white“) pixel if counter value is below (above) the threshold.
- Detect local minima of the „black“ regions respectively by resorting to the counter values.

Gray-level corners via SUSAN

Partial illustration: SUSAN applied to an image with two homogeneous regions, counter values at selected points (a : 24, b : 9, c : 20, d : 5, e : 15), gray-level corners at points b and d.



Image



SUSAN window

Gray-level corners via SUSAN

Remarks:

- To achieve roughly a rotational invariance, the SUSAN window must be approximately circular.
- The criterion „similar gray-level“ must be formalised by a further threshold parameter.
- An also well known, alternative method is the so-called „Harris Corner Detector“.

Experiments of visual gripper servoing

- Implementation on a SUN computer (in year 1995!) in C++.
- Target object at a distance of 30 - 90 cm from the cameras.
- Searching for the target object in the next stereo images within an area of +10 to -10 pixel from the current position (efficiency reasons).
- System runs with 18.5 Hz. During one second, 18.5 stereo images are processed, values for the vector of corrective variables calculated, and the robot gradually moved.

Experiments of visual gripper servoing

- The gripper is allowed to move in 3D, such that a maximum distance of 185 pixel is covered in the stereo images during one second. This is equivalent to approximately 170 mm in the scene (i.e. maximal velocity of the gripper).
- If corner extraction is inaccurate up to one pixel, then there is an inaccuracy in positioning of the gripper object of approximately 1 mm.

4.7 Modeling of camera and camera-robot relation

Overview:

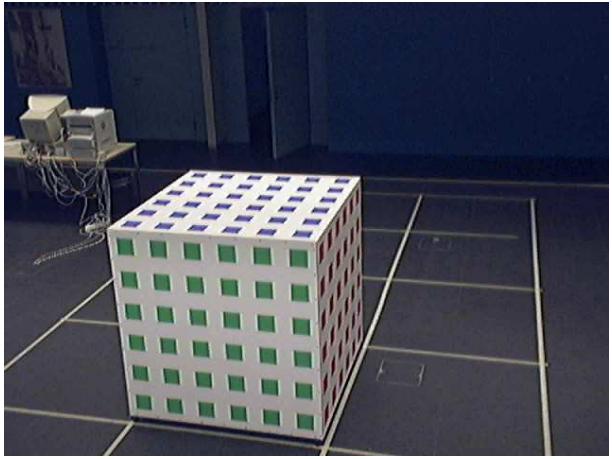
- Explicit estimation of camera parameters
- Direct modeling of camera-robot relation
- Servoing for localisation of optical axis
- Servoing for characterisation of field-of-view

Explicit estimation of camera parameters

- Placing a so-called calibration object in the field of view of a camera.
- A pattern is imprinted on the object surface, with known 3D positions ${}^{St}\boldsymbol{P}_{\boldsymbol{j}}^{\boldsymbol{3D}}$ of salient points in mm (e.g. corner points of squared shapes).
- The corresponding points ${}^{I_i}\boldsymbol{P}_{\boldsymbol{j}}^{\boldsymbol{2D}}$ in the image in pixel are extracted via image processing.

Explicit estimation of camera parameters

Calibration cube with imprinted pattern.



Explicit estimation of camera parameters

- A set $\{({}^{St}\boldsymbol{P}_j^{3D}, {}^{I_i}\boldsymbol{P}_j^{2D})\}$ of pairs of corresponding points is used as input to a standard parameter estimation method (e.g. from Matlab) to determine the intrinsic camera parameters, e.g. $\boldsymbol{d_{im}}$ and $\boldsymbol{s_{xy}}$, and extrinsic camera parameters, i.e. pose of camera relative to the station coordinate system.

Explicit estimation of camera parameters

Set of 8 unknown camera parameters:

$$\underbrace{X_C, Y_C, Z_C}_{3}, \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}}_{9}, \underbrace{d_{im}, s_{xy}}_{2}$$

The 9 entries in the rotation matrix include dependencies, which are formalized by separate equations, resulting in 3 independent rotation parameters.

Explicit estimation of camera parameters

Parameter estimation:

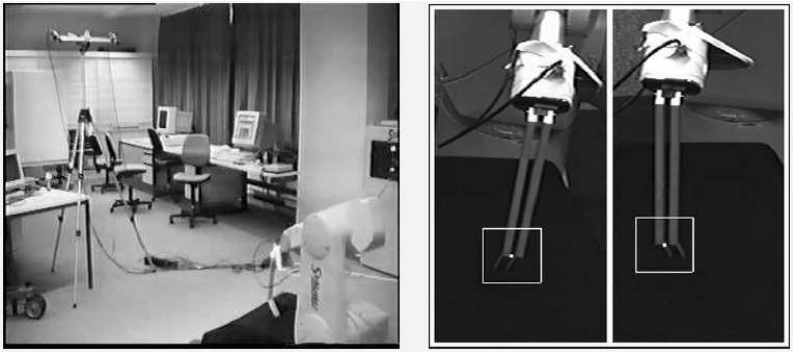
- For each pair (${}^{St}\boldsymbol{P}_j^{3D}$, ${}^{I_i}\boldsymbol{P}_j^{2D}$) of corresponding 3D and 2D point, one obtains 2 equations.
- At least 4 such pairs are needed to estimate the 8 independent parameters.
- In practice, however, hundreds of such pairs (distributed throughout the whole scene) are taken into account, in order to obtain a globally adequate projection function $\boldsymbol{g}$.

Direct modeling of camera-robot relation

- Algorithmic learning of the mapping between stereo image coordinates (in pixel) and robot coordinates (in mm), i.e. the inverse of projection function $\boldsymbol{g}$.
- For different positions of the robot hand, provide 4D vector of extracted gripper object in the coordinate system of the stereo images together with 3D vector of gripper object in the Base coordinate system of the robot.
- Based on a set of such example pairs, apply artificial neural network learning (e.g. using Gaussian Basis Function Networks) to obtain the mapping directly, i.e. omitting the Station coordinate system.

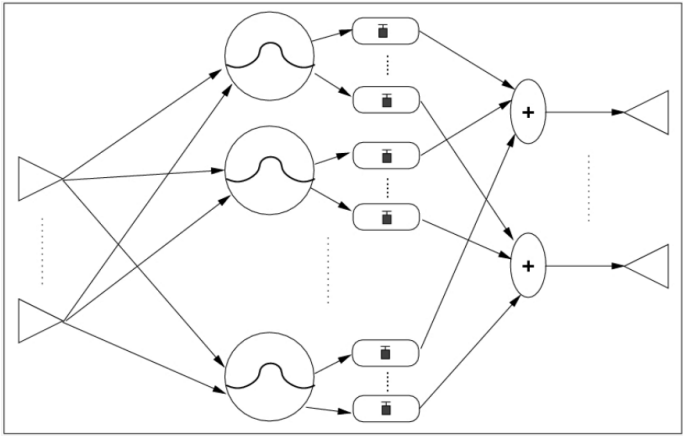
Direct modeling of camera-robot relation

Systematic „stop-and-go“ movement of the gripper to acquire a set of corresponding 4D-3D pairs of points, and thus supersede the physical calibration object.



Direct modeling of camera-robot relation

Two-layered Gaussian Basis Function Network for learning the inverted projection function.

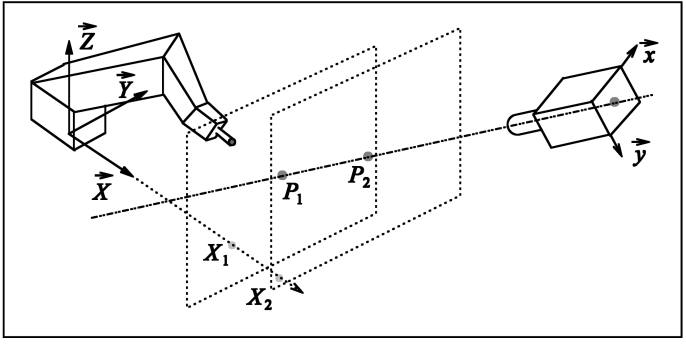


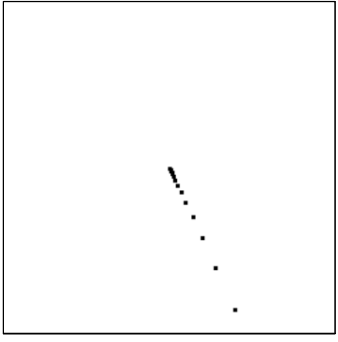
Servoing for localisation of optical axis

- Servoed movement of gripper object within a virtual plane parallel to $\vec{\boldsymbol{Y}} \times \vec{\boldsymbol{Z}}$ plane of the Base coordinate system of the robot (see next figure), such that the projected point is finally located at image center.
- Save this final 3D gripper position.
- Repeat this procedure within another virtual plane parallel to $\vec{\boldsymbol{Y}} \times \vec{\boldsymbol{Z}}$ plane at a certain offset along the $\boldsymbol{X}$ axis.
- The two saved points define the optical axis of the involved camera.
- Possible application: Bring an object closer to the camera along its optical axis, for the purpose of detailed inspection.

Servoing for localisation of optical axis

Left: Servoed movement of gripper object on two parallel planes, in order to determine points $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$ for defining the optical axis.
Right: Behavior of P-controller, as shown in the image.





Servoing for characterisation of field-of-view

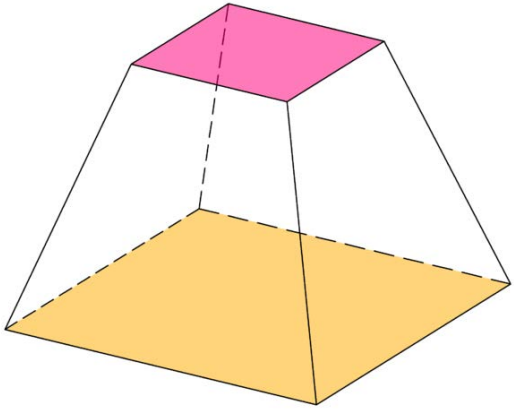
- Assume, the optical axis is now known.
- Servoed movement of gripper object along the optical axis as long as it is visible.
- At the closest point, moving the gripper on a virtual, orthogonal plane such that the projected point becomes finally located at the four image corners.
- Save the four 3D positions to build a close rectangle.

Servoing for characterisation of field-of-view

- Repeat this procedure at the farthest visible point on the optical axis, leading to a remote rectangle.
- Define a rectangular frustum (truncated rectangular pyramid) as the camera's field-of-view.

Servoing for characterisation of field-of-view

Rectangular frustum as approximation for the camera's field-of-view.



4.8 Fundamental role of control methods

Overview:

- Applications and characterising features
- Approach in general

Applications and characterising features

- Sensor-/Camera-based, elementary robot behaviours.
- Direct mapping of sensor measurements to motor signals.
- Successful in spite of unknown, changing environment and/or inaccurate system model.
- Learning and adaption of system model.

Approach in general

- Current measurement $\boldsymbol{M(t)}$ in the environment, e.g. position of gripper object (determined in the image).
- Desired, expected measurement $\boldsymbol{M_*}$ in environment, e.g. gripper at position of the target object.
- Fixed part of the system's state vector $\boldsymbol{S_f}$, e.g. lengths of the links of a robot arm, features/parameters of projection function.
- Variable part of the system's state vector $\boldsymbol{S_v(t)}$, e.g. angles of the joints of a robot arm.

Approach in general

- Vector $\boldsymbol{C(t)}$ of corrective variables, e.g. change of six joint angles of an articulated robot arm.
- Transition function $\boldsymbol{f_T}$, e.g. addition/subtraction:

$$\boldsymbol{S_v(t+1) := f_T(S_v(t), C(t))}$$

- Control (decision) function $\boldsymbol{f_D}$, e.g. PID combination including measurement vector(s) and state vector(s):

$$\boldsymbol{C(t) := f_D(M_*, M(t), S_f, S_v(t))}$$

- Examples for types of controllers, P, PI, PID.

Literature

- J. Craig: Introduction to Robotics; Pearson, 2005.
- O. Föllinger: Regelungstechnik; Huethig Verlag, 1990.
- G. Hager, et al.: Robot hand-eye coordination based on stereo vision; IEEE Control Systems, 15:30-39, 1995.
- K. Hashimoto (ed.): Visual Servoing; World Scientific Publishing, 1993.
- S. Hutchinson, et al.: A tutorial on visual servo control; IEEE Trans. on Robotics and Automation, 12:651-670, 1996.

5. Representation of Scene/Environment

Overview:

Restriction to ground-bounded robots

- Simplified description of a scene/environment through so-called „maps“.
- We define a map as 2D projection of obstacles and free-space onto the ground plane.
- A sophisticated description of 3D scene/environment would be needed for aerial robot vehicles.

5.1 Polygonal representation

Overview:

- Representation of occupied space
- Representation of free-space

Representation of occupied space

- For a mobile robot, an occupied space may be regarded as an obstacle, which must be bypassed during navigation.
- A camera may be mounted on a ceiling (indoor) or on a pole (outdoor).
- Low-level image processing produces a binary image with gray-level edges belonging to the boundaries of obstacle objects.
- Medium-level image processing produces polygonal approximations of the objects boundaries, which are perceptual organisations/ subsets of the set of gray-level edges.

Representation of occupied space

- Each polygon is a sequence of straight line segments, for example ordered clockwise.
- According to this arrangement, at righthand side the space is occupied and at lefthand side the space is free.
- This representation of scene/environment is
  - object-centered, i.e. enables a robotic manipulation of the object,
  - boundary-based, i.e. describing the contours of objects.

Representation of free-space

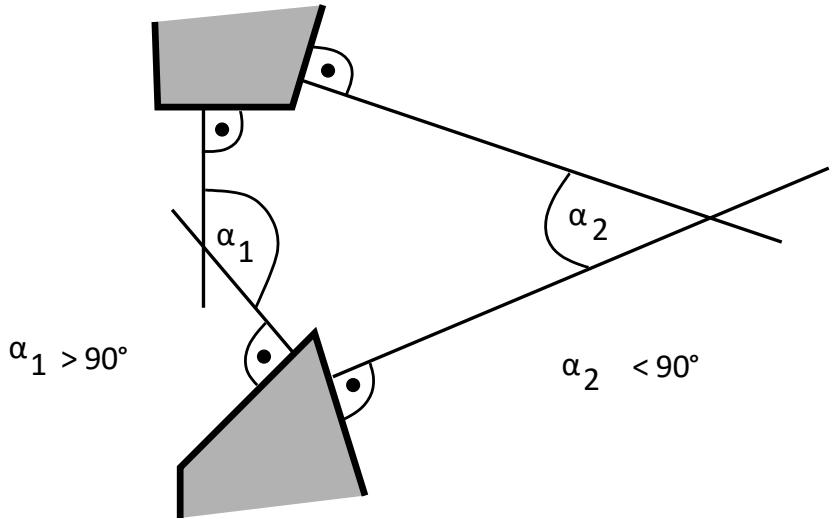
Adapted from Brooks (1990): Approximation of free-space through a superposition of triangles.

- Consider from all obstacles, including also the walls, the overall set of straight line segments from the boundaries.
- These boundary lines are taken from the polygonal representation (see above).
- This representation of scene/environment is
  - free-space-centered, i.e. enables a robot navigation,
  - boundary-based.

Representation of free-space

- For all possible pairs of boundary lines, check the following two conditions, and keep a pair if both conditions become true:
  - The lines orthogonal to the two boundary lines must form an angle between $\mathbf{90}^\circ$ and $\mathbf{270}^\circ$, i.e. the two boundary lines are roughly across from each other.
  - If the two boundary lines are across from each other, then there must not be an object located in between.

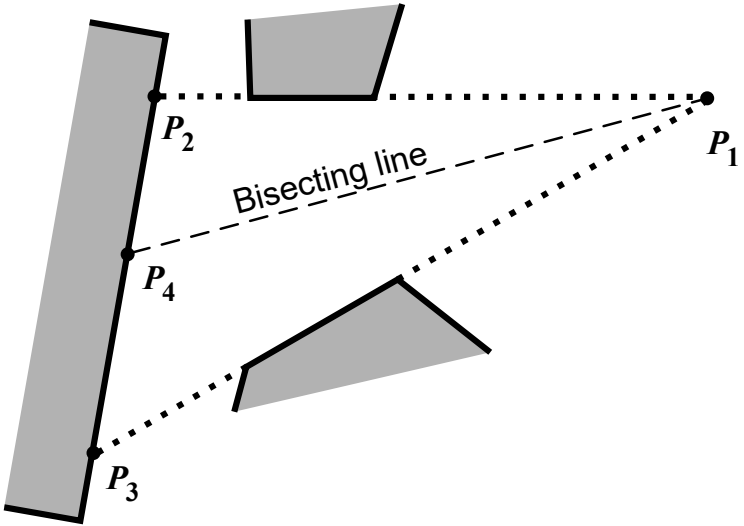
Representation of free-space



Representation of free-space

- For each kept (remaining) pair of boundary lines:
  - Extend the boundary lines until the intersection point $\boldsymbol{P_1}$.
  - Regard these as the principal lines of a triangle.
  - Extend the principal lines on the other end according to their orientations, and until reaching an obstacle.
  - This leads to points $\boldsymbol{P_2}$ and $\boldsymbol{P_3}$, forming the basis line of the triangle.
  - Compute the bisecting line segment between the principal lines, ending at $\boldsymbol{P_1}$ and at the intersection point $\boldsymbol{P_4}$ with the basis line.

Representation of free-space



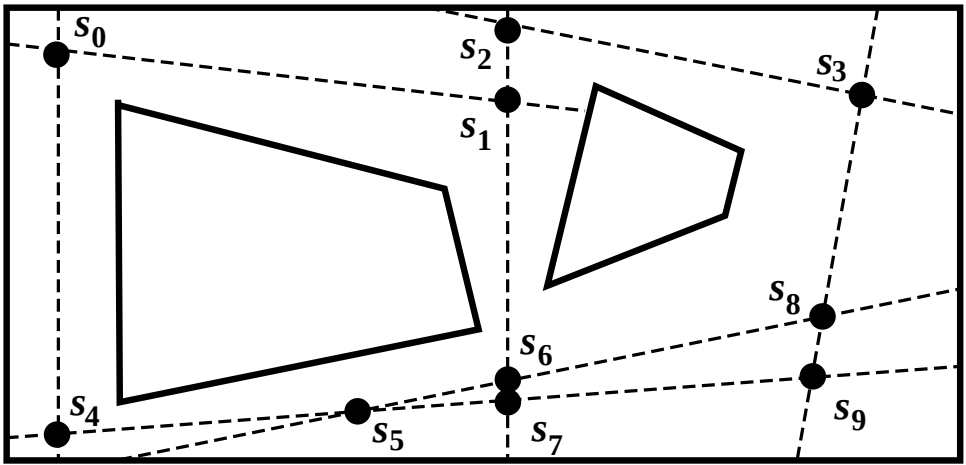
Representation of free-space

Usage for robot navigation:

- The set of bisecting line segments is used as an approximation of the free-space, i.e. a skeleton.
- A robot moving along the bisecting line segments keeps largest distances from obstacles.
- Intersections between bisecting line segments are used to define branching points.
- Robot motion planning may be based on bisecting line segments and branching points.
- I.e. find an optimal sequence of connected bisecting line segments from starting point to target point (see section 6.2, A*-Algorithm).

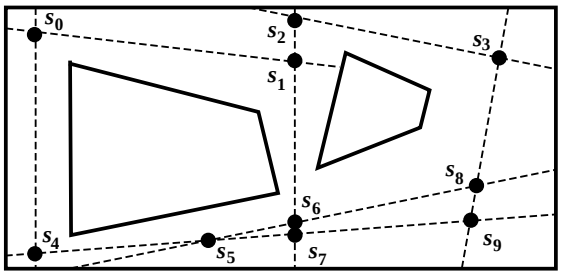
Representation of free-space

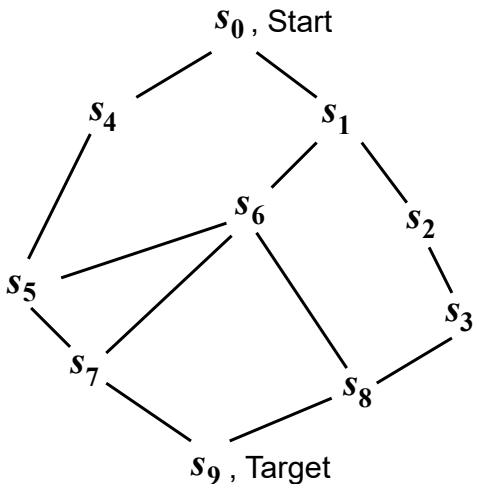
Bisecting lines in the free-space, branching points s_i.



Representation of free-space

Navigation from start position to target position, e.g. from s_0 to s_9, using a state graph.





5.2 Quadtree representation

Overview:

- Construction of a quadtree
- Example of a quadtree
- Remarks concerning quadtree

Construction of a quadtree

- This representation of scene/environment is
  - object-centered,
  - region-based, i.e. describing the areas of objects.
- A binarised image is assumed, with white pixels representing free-space and black pixels representing obstacles.

Construction of a quadtree

- The image has rectangular form, with the size assumed as $(k \cdot 2^i) \times (l \cdot 2^i)$, for natural numbers k, l, i.
- Examples:
  For an image with 480×640 pixel, $k = 15, l = 20, i = 5$;
  for an image with 576×768 pixel, $k = 9, l = 12, i = 6$.
- Decomposition of the image into four uniform rectangles of size $(k \cdot 2^{i-1}) \times (l \cdot 2^{i-1})$.
- For certain rectangles, further decomposition into four smaller, uniform rectangles (sub-rectangles) of size $(k \cdot 2^{i-2}) \times (l \cdot 2^{i-2})$.

Construction of a quadtree

- Continue with decomposition iteratively for certain rectangles, until a stopping criterion is met.
- Each rectangle is represented as a node in the quadtree.
- The original image defines the root node.
- The decomposition of a rectangle into four sub-rectangles is represented as the so-called expansion of a node by four child nodes.

Construction of a quadtree

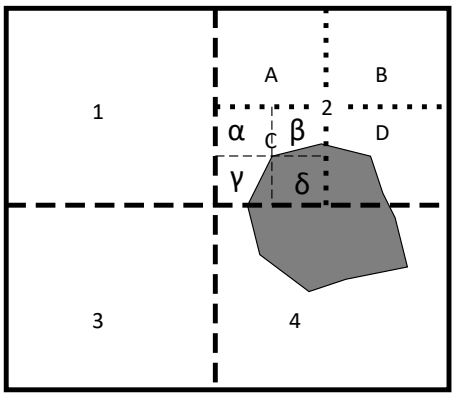
- There are four types of nodes.

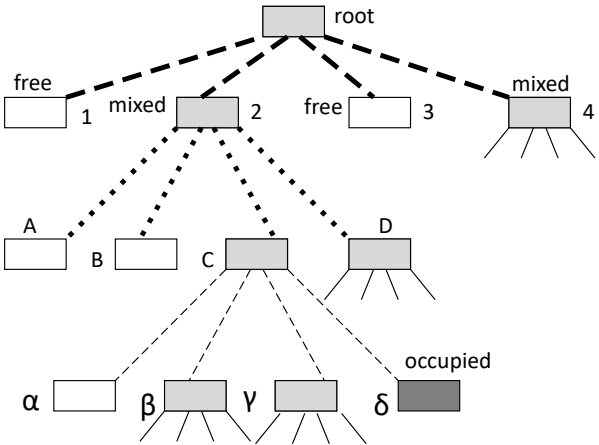
  For a rectangle, represented by a:

  - „free node“, the area is totally free;
  - „occupied node“, the area is totally occupied;
  - „mixed node“, the area is partially free and occupied;
  - „minimal node“ the area should not be decomposed further, because of an application-dependent „minimal-criterion“.

- Only a „mixed node“, for which the „minimal-criterion“ is not fulfilled, must be expanded further.

Example of a quadtree





Remarks concerning quadtree

- Concerning „minimal-criterion“: A „mixed node“ needs no further expansion, if the robot size is larger then the relevant rectangle.
- Based on the constructed quadtree, the navigation task must be solved. However, this causes problems due to missing links between neighbouring areas.
- A natural extension to representing 3D environments can be obtained with so-called octrees (eight child nodes).

5.3 Probabilistic occupancy grid

Overview:

- Construction of an occupancy grid
- Example of an occupancy grid

Construction of an occupancy grid

See Elfes (1989).

- Originally designed for use of ultrasonic sensors, which produce inaccurate distance measurements.
  $\Rightarrow$ Exact positions of obstacles are hardly available.
- Inaccuracy and uncertainty concerning position of obstacles are represented as probabilities.
- Environment is decomposed in a regular grid of cells.
- For each cell, a value in the unit interval $[0, ..., 1]$ represents the probability, that this cell is occupied by an object.

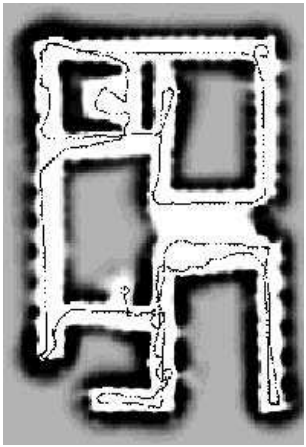
Construction of an occupancy grid

- Value $\mathbf{0}$ means, cell is totally free, with certainty.
- Value $\mathbf{1}$ means, cell is totally occupied, with certainty.
- A value in between means, cell is occupied at a certain part, or may be totally occupied but not with certainty.
- Initially, all cells have value $\mathbf{0.5}$, which means "occupancy state is not known".
- The probability values in the occupancy grid are determined by robotic exploration of the environment, making continual measurements, and fusing all the data.

Construction of an occupancy grid

- An essential precondition is automatic robot self-localisation in order to accumulate all relevant data for the same cell, e.g. see chapter 7 (PRL) in this course Cognitive Robot Systems, and chapter 5 (ODM) in course Computer/Robot Vision.
- This representation of scene/environment is
  - free-space-centered,
  - boundary-based.

Example of an occupancy grid



From Thrun and Bücken, AAAI, 1996.

5.4 Vector field representation

Overview:

- Vision-based detection of target and obstacles
- Physics-based representation of free-space
- Attractor vector field
- Repellor vector field
- Superposition of all force vector fields
- Possible definitions of force vector fields
- Remarks to vector field representation

Vision-based detection of target and obstacles

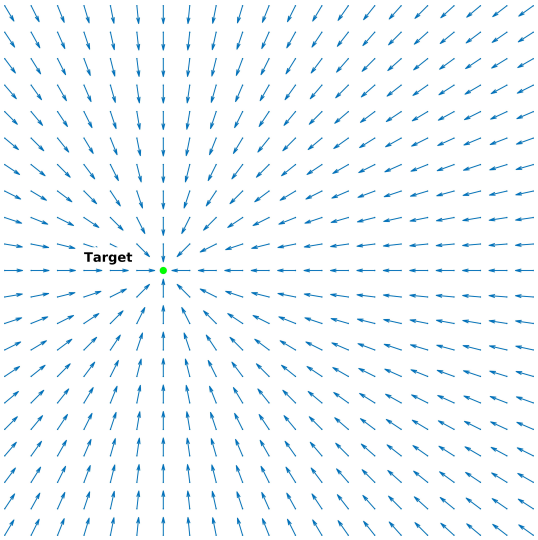
- Assuming a navigation task (as opposed to manipulation task).
- Take an image from a bird's eye view.
- Analyse the image to detect objects.
- Classify objects into target (goal) versus obstacles.
- Extract center points and/or boundary points from the target and obstacles.

Physics-based representation of free-space

- Regard the free-space as containing an overlay of virtual forces, which have a behavioral effect on the movement of the robot.
- Target object should virtually pull (attract) the robot, which is realised by so-called attractor vector fields.
- Obstacle objects should virtually keep away (push, repel) the robot, which is realised by so-called repellor vector fields.
- See Latombe (1991, pp. 295-355).
- This representation of scene/environment is
  - free-space-centered,
  - boundary-based and region-based.

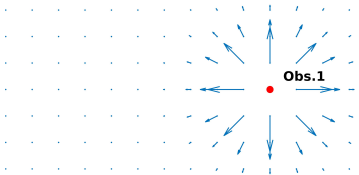
Attractor vector field

Simplification:
Point-like target object.



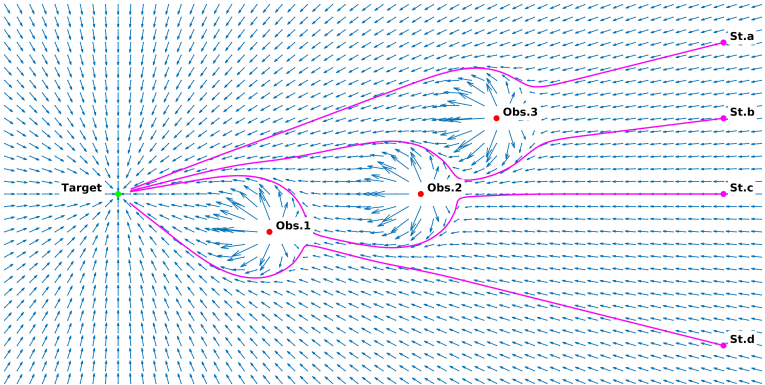
Repellor vector field

Simplification:
Point-like obstacle object.



Superposition of all force vector fields

Example: One target object, three obstacles, four different starting positions for the robot; obstacle-avoiding, goal-directed navigations based on the resulting („force“) vector field.



Possible definitions of force vector fields

Attractor:

$$\boldsymbol{AF}_{p_G}(\boldsymbol{p}) := \boldsymbol{\alpha} \cdot \frac{(p_G - p)}{\|p_G - p\|}$$

Separate definition for special case $\boldsymbol{p_G} = \boldsymbol{p}$

Repellor:

$$\boldsymbol{RF}_{p_O}(\boldsymbol{p}) := \beta \cdot \frac{(p - p_O)}{\|p - p_O\|} \cdot \exp(-\|\boldsymbol{p} - \boldsymbol{p_O}\|^2)$$

Separate definition for special case $\boldsymbol{p} = \boldsymbol{p_O}$

Additive superposition:

$$\boldsymbol{V}(\boldsymbol{p}) := \boldsymbol{AF}_{p_G}(\boldsymbol{p}) + \Sigma_{\forall p_O}\, \boldsymbol{RF}_{p_O}(\boldsymbol{p})$$

Remarks to vector field representation

- Synonym: Potential field representation
- The range of an attractor should cover the whole scene, i.e. imposing an effect on all possible robot positions. The virtual force may be constant or depend proportionally on the distance between target and robot, e.g. controlled by factor α.
- The range of a repellor is practically limited, and may include uncertainty of sensing, i.e. close or far bypassing. The virtual force may be inverse proportional with the distance between robot and obstacle, e.g. controlled by factor β.

Remarks to vector field representation

- If target object is extended (as opposed to point-like or circular), then anisotropic attractor vector fields have to be defined and centered individually at the boundary points of the target, e.g. to specifically home in on the target for subsequent manipulation.
- Also for an extended obstacle, several anisotropic repellor fields have to be centered at the boundary, e.g. for close (far) bypassing at high- (low-) accurate parts of boundary extraction.

Literature

- R. Brooks: Solving the find-path problem; in Cox et al. (eds.): Autonomous Vehicles, pp. 290-297, Springer Verlag, 1990.
- A. Elfes: Occupancy Grids - A Probabilistic Framework for Robot Perception and Navigation; PhD thesis, CMU, 1989.
- J.-C. Latombe: Robot Motion Planning; Kluwer Academic Publishers, Boston, USA, 1991.
- S. Thrun, E. Bücken: Integrating grid-based and topological maps for mobile robot navigation; AAAI, pp. 944-950, 1996.
- Wikipedia (-> Quadtree).

6. Path Planning for Robot Navigation

Overview:

6.1 Planning task at a glance

Overview:

- Optimality criteria for motion planning
- Motion planning based on state graphs
- Heuristics in state graph based search
- Motion planning based on vector fields

Optimality criteria for motion planning

- Assuming a certain Start position of the robot.
- From there, determine a path of movement towards the Target position.
- This so-called robot motion planning is subject to optimality criteria, e.g.:
  - minimal path length, collision free, minimal driving time, minimal number and extents of robot rotations, minimal robot vibrations.
- See combinations of optimality criteria in section 6.4.

Motion planning based on state graphs

- The basis may be state graphs, whose edges represent the sub-paths of movement, and whose nodes represent states (i.e. positions of the robot) which are branching points for alternative sub-paths (see section 5.1).
- Robot motion planning is then tackled by applying standard or sophisticated techniques from graph theory, i.e. determine optimal path from Start to Target node.

Heuristics in state graph based search

- Searching for the optimal path in a state graph may be resource-intensive and time-consuming.
- Including additional, common-sense knowledge (heuristics) of motion planning can significantly accelerate the search.

Motion planning based on vector fields

- Superposition of attractor and repellor vector fields may lead to suboptimal robot behavior.
  Postprocessing of local, vector-based sub-paths should yield a globally optimal path.
- The vector field representation should be generalised to also include the occupied volume of the robot and, if necessary, even its changing volume.
  E.g. in case of navigating the hand of an articulated robot arm, or navigating the head of an articulated pipe robot.

6.2 A* algorithm

Overview:

- Basic mathematical symbols and example
- A* Algorithm at a glance
- Two-part cost function
- Principle of graph expansion
- Example of graph expansion
- Example of applying A* algorithm
- Heuristics in A*-based search
- Origin of name A*

Basic mathematical symbols

Let $\boldsymbol{\mathcal{S}} := \{s_0, s_1, ..., s_m\}$ be the nodes of a graph $\boldsymbol{G}$.

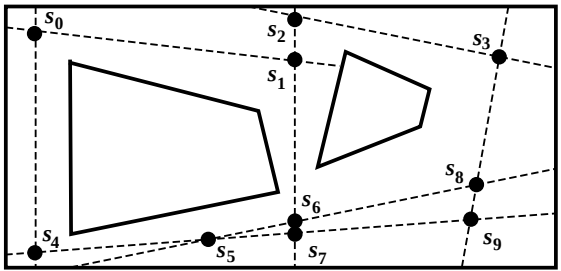
Without loss of generality, let $\boldsymbol{s_0}$ be the Start node, and $\boldsymbol{s_m}$ the Target node of the graph.

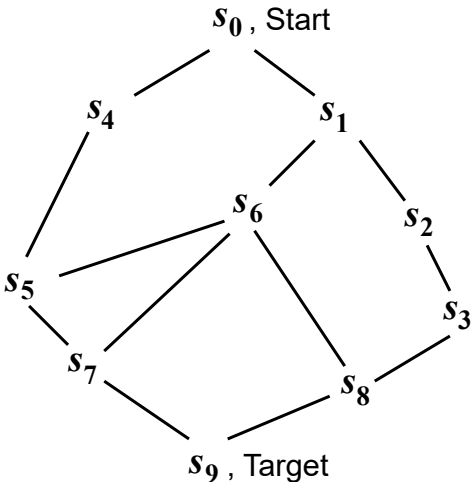
Let $\boldsymbol{W^i} := (s_0^i, s_1^i, ..., s_n^i)$ be a partial path in the graph from Start node $\boldsymbol{s_0^i} := \boldsymbol{s_0}$ to an end node $\boldsymbol{s_n^i}$, with $\boldsymbol{s_j^i \in \mathcal{S},\ j \in \{0, 1, \cdots, n\}}$.

The end node of $\boldsymbol{W^i}$ may be an intermediate node in the graph $\boldsymbol{G}$, or as a special case, be the Target node $\boldsymbol{s_m}$.

Example for a state graph representation

Navigation from start position to target position, e.g. from s_0 to s_9, using a state graph.





A* Algorithm at a glance

- Modification/Change of a set $\{\boldsymbol{W}^1, ..., \boldsymbol{W}^p\}$ of partial paths.
- All these partial paths start with $\boldsymbol{s_0}$, but have different end nodes.
- „Expand“ the most promising, cost-effective path.
- Abort the iteration, if the end node of the most cost-effective path is the Target node of the graph.
- The selected path through the graph represents the optimal motion plan for the robot.
- Continually expanding the most promising path and early stopping of graph expansion realises a highly efficient motion planning.

Two-part cost function

Let f be a cost function, which considers by addition:

- the costs of the already passed partial path (part g), i.e. the incurred costs between s_0 and s_n^i,
- and a heuristic estimation of the remaining costs, i.e. the lower bound of the costs from intermediate node s_n^i to Target node s_m (part h).

$$f(W^i) := g(W^i) + h(W^i)$$

An example for defining the function part h is the Euclidean distance between the positions indicated by the states s_n^i and s_m.

Principle of graph expansion

Assume, in a certain phase of the running algorithm, the set of partial paths, to be managed, may consist of $\{W^1, ..., W^p\}$.

Without loss of generality, let W^p be the most cost-effective path, i.e. $W^p := argmin_{W^i}\{f(W^i)\}$.

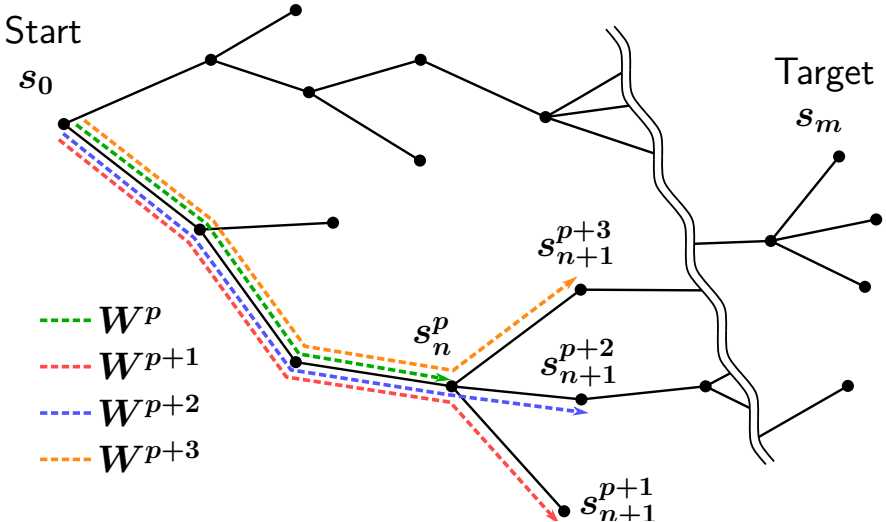
Path W^p will be expanded at its end node s_n^p.

This leads to r new paths $\{W^{p+1}, ..., W^{p+r}\}$ with respective end nodes $s_{n+1}^{p+1}, ..., s_{n+1}^{p+r}$.

After deleting W^p, the further management includes the paths

$$\{W^1, ..., W^{p-1}, W^{p+1}, ..., W^{p+r}\}$$

Example of graph expansion



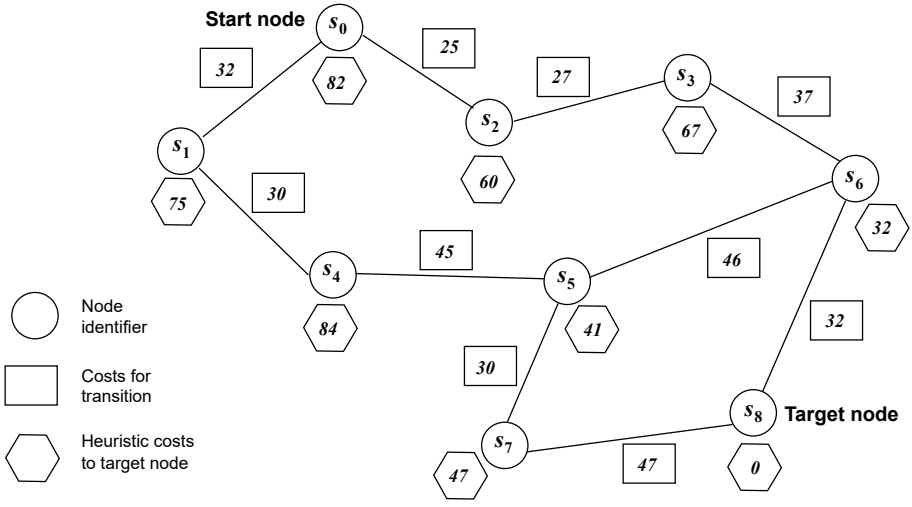
Principle of graph expansion

If by expansion a node is obtained, which can also be reached by another path in the set $\{\boldsymbol{W}^1, ..., \boldsymbol{W}^{p-1}, \boldsymbol{W}^{p+1}, ..., \boldsymbol{W}^{p+r}\}$, then the path with higher costs will be deleted.

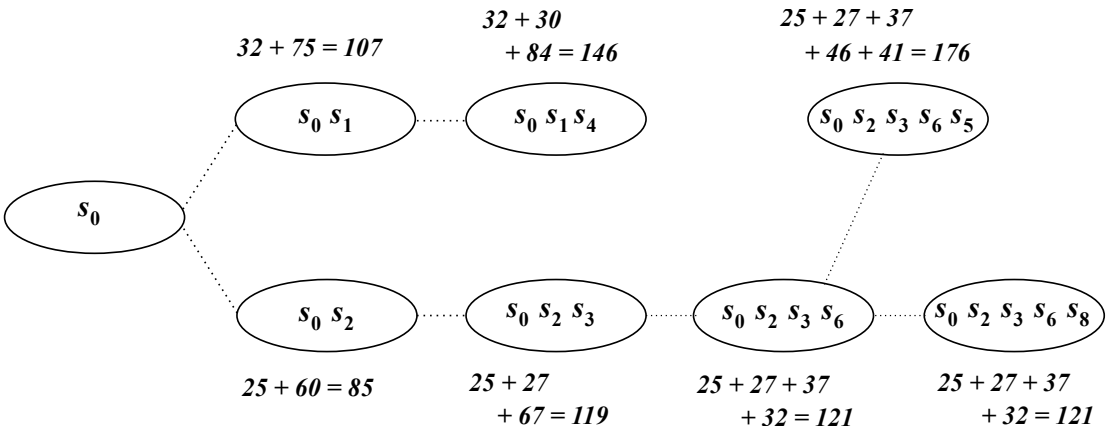
If after expansion the most cost-effective path $\boldsymbol{W}^l$ is the one, whose end node is equal to the Target node $\boldsymbol{s_m}$ of the graph, then stop the procedure and output this path.

Else continue with expansion of this most cost-effective path $\boldsymbol{W}^l$, as above.

Example of applying A* algorithm



Example of applying A* algorithm



Heuristics in A*-based search

- The length of the actual stretch of way to a Target position is equal or longer than the Euclidean distance.
- This leads to a strategy of early pruning branches/options in the search space (see also Pearl, 1985).

Origin of name A*

- Symbol **A** stands for admissible search algorithm. It makes use of admissible heuristics.
- Symbol $*$ stands for optimality, i.e. search for an optimal path, and time-efficient search.

6.3 Dynamic optimisation

Overview:

- DO algorithm at a glance
- Example of applying DO algorithm
- Specification of DO algorithm
- Implementation of DO algorithm
- Heuristics in DO-based search
- Remarks to DO algorithm
- Origin of name Dynamic Optimisation
- Common features of DO and A* in motion planning

DO algorithm at a glance

- Organisation of all nodes of the state graph into N layers.
- The first layer contains the Start node, the N-th layer contains the Target node.
- Beginning with the Target (!) node, determine the costs from all nodes in layer $(N-1)$ to Target node.
- Store cost value and pointer to Target node as attributes in the relevant node of layer $(N-1)$.

DO algorithm at a glance

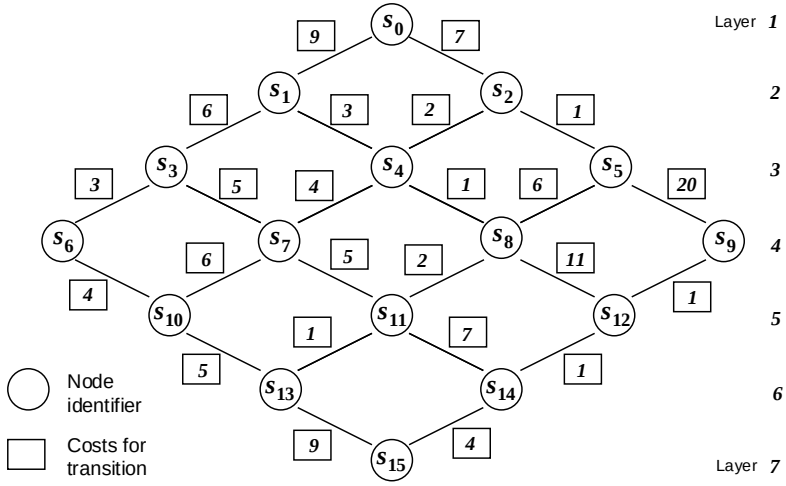
- Continue with processing all nodes in layer $(N-2)$.
- Determine the minimal cost respectively of all nodes in layer $(N-2)$ to Target node by summing up the cost to nodes in layer $(N-1)$ and cost from there to Target node.
- Store in each node of layer $(N-2)$ this minimal cost value and the pointer to the relevant/optimal node of layer $(N-1)$.

DO algorithm at a glance

- Iteratively repeat the procedure and continually use the collected information until the first layer (i.e. the Start node).
- Finally, follow the sequence of pointers (arrows) beginning with the Start node downwards over all layers to the Target node.
- The cost attached at the Start node and the visited intermediate nodes represent the optimal motion plan.

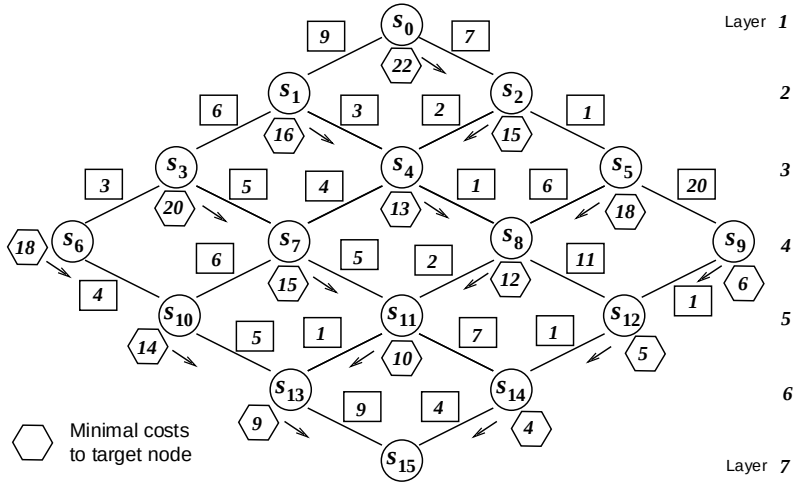
Example of applying DO algorithm

Exemplary, layered state graph.



Example of applying DO algorithm

Result of DO leading to the optimal path from Start to Target node.



Specification of DO algorithm

- Define N variables x_i for N layers, $i \in \{1, ..., N\}$.
- A variable will accept the possible states of the respective layer. E.g.: $x_1 \in \{s_0\}$, $x_2 \in \{s_1, s_2\}$, $x_3 \in \{s_3, s_4, s_5\}$, etc.
- A function $f(x_1, ..., x_N)$ defines the costs for a path from Start state to Target state, e.g. here from s_0 to s_{15}, and depends on the sequence of states allocated to the variables $x_1, ..., x_N$.

Specification of DO algorithm

- In order to minimise $f(x_1, ..., x_N)$, one is searching for specific values $x_1^*, ..., x_N^*$, i.e. a specific sequence of possible, intermediate states.
- It is assumed, that the cost function can be sub-divided into:

$$f(x_1, ..., x_N) := f_1(x_1, x_2) + f_2(x_2, x_3) + ... + f_{N-1}(x_{N-1}, x_N)$$

Implementation of DO algorithm

- Initialisation: $\boldsymbol{F_N(x_N) := 0}$
- Iteration for $\boldsymbol{i}$ beginning with $\boldsymbol{N}$ and successively decrementing until $\boldsymbol{2}$:
  - $\boldsymbol{F_{i-1}(x_{i-1}) := min_{x_i}\{f_{i-1}(x_{i-1}, x_i) + F_i(x_i)\}}$

    /*

    Calculate $\boldsymbol{F_{i-1}}$ for all possible states $\boldsymbol{x_{i-1}}$ of layer $\boldsymbol{(i-1)}$.

    Minimise over all states $\boldsymbol{x_i}$ of layer $\boldsymbol{(i)}$, which can be reached from $\boldsymbol{x_{i-1}}$.

    /*

Heuristics in DO-based search

- Every sub-path of a globally optimal path is locally optimal.
- This leads to a sequential divide-and-conquer strategy when exploring the search space.

Remarks to DO algorithm

- DO applies a sequential divide-and-conquer strategy, i.e. a higher-dimensional cost function is partitioned into a set of lower-dimensional partial cost functions, which together form by addition the original function. Specifically, a sequence of optimisations takes place respectively in a one-dimensional space.
- For a more complex cost function, maybe a somewhat more complex partitioning is possible, e.g. leading to a sequence of optimisations respectively in a one- or two-dimensional space.

Origin of name Dynamic Optimisation

Synonym: Dynamic Programming. See Wiki.

Method developed by R. Bellman in the 1950s.

Originally applied to economics.

Has many other applications, e.g.:

- Cognitive Robot Systems:
  - Robot motion planning
  - Robot online learning
- Image Analysis: e.g. extraction of object contours.
- Etc.

Common features of DO and A* in motion planning

- For DO, the scene must be known, in order to search through the graph beginning at the Target state.
- For A*, the scene must be known as well, in order to search through the graph beginning at the Start state, and including heuristic function (based on Target state).
- Both methods are based on a graph-like scene/environment representation (see chapter 5).
- Usually, these kind of motion planning takes place off-line prior to the actual robot movement.

6.4 Cost function for navigation

Overview:

- Compromise of interests for navigation
- Example of a grid-like scene
- Components of the cost function
- Navigation costs for two exemplary paths

Compromise of interests for navigation

Example of competing optimisation criteria for navigation:

- For resource-efficiency, take the shortest path from Start to Target position.
- For safety reason, take a spacious bypass from the obstacle.

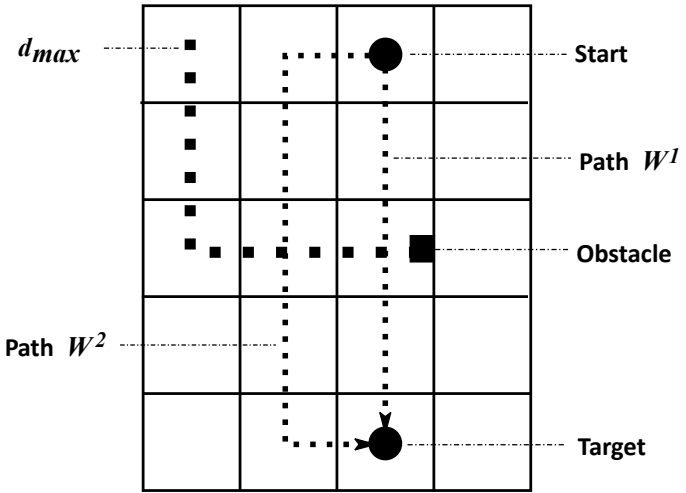
The cost function should include an individual weighting to enable an application-dependent compromise.

Example of a grid-like scene

- Assume a grid-like scene.
- Movement from cell to cell with elementary cost $\mathbf{1}$, in horizontal or vertical direction, not diagonal.
- These elementary costs are based on the so-called city block metric (syn. Manhattan distance).
- The robot and the obstacle are small compared to the size of a cell.
- This enables the robot to bypass the obstacle at a short distance, i.e. also pass through the partially occupied cell.
- However, this narrow bypassing bears some risk of collision.

Example of a grid-like scene

Two exemplary paths from Start to Target.



Components of the cost function

- Refer to notation from section 6.2 (A* algorithm).
- Let $\boldsymbol{W^i}$ be a path from Start node $\boldsymbol{s_0^i}$ to an end node $\boldsymbol{s_n^i}$, but the latter not necessarily equal with the Target node $\boldsymbol{s_m}$ of the overall state graph.
- Each node represents a cell in the grid-like scene.
- Cost function: $\boldsymbol{f(W^i) := g(W^i) + h(W^i)}$
- Heuristic function $\boldsymbol{h(W^i)}$ represents a lower bound of costs from $\boldsymbol{s_n^i}$ to Target $\boldsymbol{s_m}$, but will be neglected here.

Components of the cost function

- The incurred costs of path $\boldsymbol{W^i}$ are in turn split in two parts:
  $\boldsymbol{g(W^i) := g_1(W^i) + \alpha \cdot g_2(W^i)}$
- $\boldsymbol{g_1(W^i)}$: Sum of elementary costs from $\boldsymbol{s_0^i}$ to $\boldsymbol{s_n^i}$.
- $\boldsymbol{g_2(W^i)}$: Sum of elementary risks of collision.

  E.g., elementary risk of collision at node $\boldsymbol{s_n^i}$: $\boldsymbol{(d_{max} - d(s_n^i))}$,
  with distance $\boldsymbol{d(s_n^i)}$ between $\boldsymbol{s_n^i}$ and nearest obstacle,
  and $\boldsymbol{d_{max} := max\{d(s_j)|j \in \{1, ..., m\}\}}$ the maximal
  distance to an obstacle (in slide 35: $\boldsymbol{m = 20}$, $\boldsymbol{d_{max} = 4}$).

Components of the cost function

- Weighting factor $\boldsymbol{\alpha}$ for defining the compromise between shortest path and spacious bypass.
- If $\boldsymbol{\alpha}$ is high,
  then components $(\boldsymbol{d_{max}} - \boldsymbol{d}(\boldsymbol{s_j^i}))$ must be small, in order to keep $\boldsymbol{g_2}(\boldsymbol{W^i})$ low. This means spacious bypass of obstacle.
- If $\boldsymbol{\alpha}$ is low,
  then components $(\boldsymbol{d_{max}} - \boldsymbol{d}(\boldsymbol{s_j^i}))$ can be large, because a large $\boldsymbol{g_2}(\boldsymbol{W^i})$ scarcely effects $\boldsymbol{g}(\boldsymbol{W^i})$. This means narrow bypass but short path.

Navigation costs for two exemplary paths

Cost for path $\boldsymbol{W^1}$ from Start to Target cell:

$$\boldsymbol{g(W^1) := (1+1+1+1) + \alpha \cdot ((4-1)+(4-0)+ (4-1)+(4-2)) = 4 + \alpha \cdot 12}$$

Cost for path $\boldsymbol{W^2}$ from Start to Target cell:

$$\boldsymbol{g(W^2) := (1+1+1+1+1+1) + \alpha \cdot ((4-3)+(4-2)+ (4-1)+(4-2)+(4-3)+(4-2)) = 6 + \alpha \cdot 11}$$

$\boldsymbol{\alpha} = 3$: cost 40 of path $\boldsymbol{W^1}$, cost 39 of path $\boldsymbol{W^2}$.

$\boldsymbol{\alpha} = 1$: cost 16 of path $\boldsymbol{W^1}$, cost 17 of path $\boldsymbol{W^2}$.

6.5 Vector field navigation of robot arm

Overview:

- Challenge of navigating a robot hand
- Geometric description of a robot arm
- Collision situation or grasping situation
- Alternative grasping situations
- Construction of the vector field
- Simulated movements of robot hand
- Illustration of navigating a robot hand

Challenge of navigating a robot hand

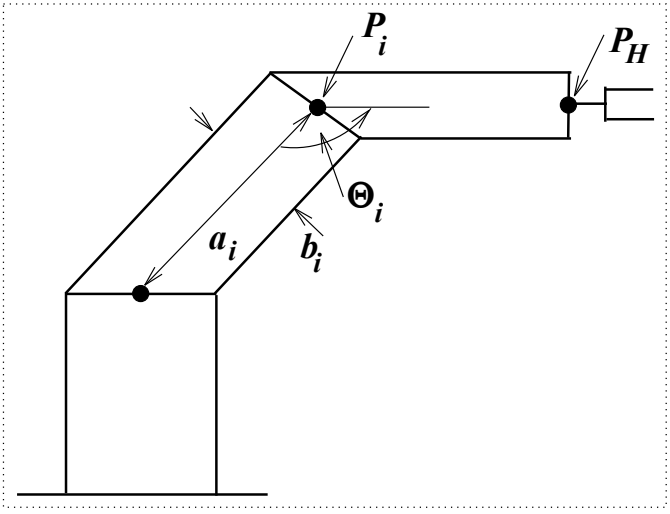
Challenge of navigating a robot hand

Task:

- Construction and usage of a vector field representation of the scene for goal-directed, obstacle-avoiding navigation of the robot hand.
- Due to its agility, the robot arm occupies a non-rigid, variable volume, including the agile fingers, hand, links.
- An holistic obstacle avoidance is required.

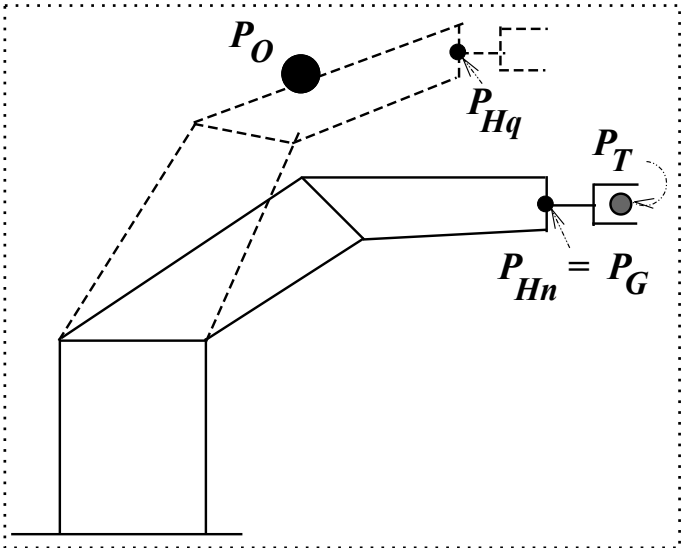
Geometric description of a robot arm

Fixed features S_f,
variable features S_v.



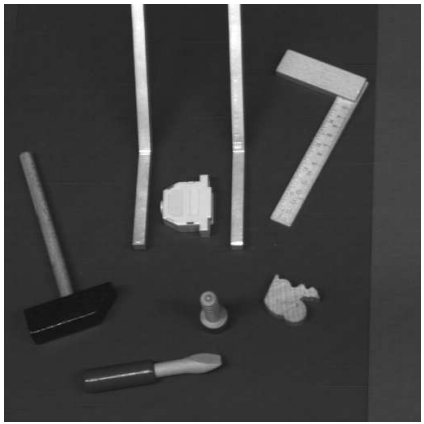
Collision situation or grasping situation

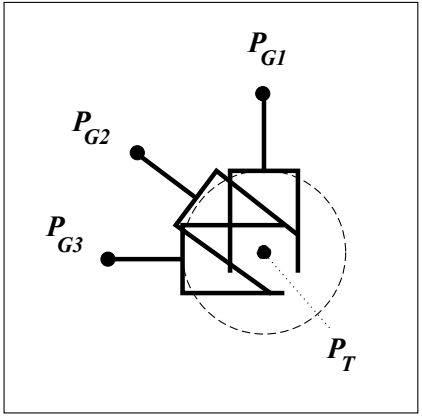
Grasping situation for robot hand at position P_{Hn}.
Collision at position P_O for robot hand at position P_{Hq}.



Alternative grasping situations

Gray-level image of a grasping situation (left), several options of grasping situations (right).





Construction of the vector field

Given:

- A set $\boldsymbol{OB}$ of 3D points, belonging to an obstacle object (e.g. its corner points).
- Desired grasping position $\boldsymbol{P_G}$ of the robot hand, for grasping an object located at position $\boldsymbol{P_T}$.
- Fixed features $\boldsymbol{S_f}$ of the robot arm, e.g. diameter and length of lower and upper arm respectively.

Construction of the vector field

Control:

- Variable features $\boldsymbol{S_{v_j}}$ of the robot arm, e.g. joint angles. Index $\boldsymbol{j}$ is an identifier for variable arm configurations.
- Direct (Forward) kinematics, leading to current position of robot hand $\boldsymbol{P_{H_j}}$ and current volume $\boldsymbol{VL_{R_j}}$ of the entire robot arm (occupied space);
  $\boldsymbol{P_{H_j}}$ and $\boldsymbol{VL_{R_j}}$ belong to variable features $\boldsymbol{S_{v_j}}$ at configuration $\boldsymbol{j}$.

Construction of the vector field

Requirements:

- Determine trajectory for movement of the robot hand to the grasping position: $< \boldsymbol{P_{H_1}}, \cdots, \boldsymbol{P_{H_n}} >, \quad \boldsymbol{P_{H_n}} = \boldsymbol{P_G}$
- Avoid collisions of robot arm and obstacle objects:
$$\boldsymbol{VL_{R_j}} \cap \boldsymbol{OB} = \{\}, \quad \forall \boldsymbol{j} \in \{1, \cdots, \boldsymbol{n}\}$$
- Acquire control function $\boldsymbol{f_D}$ for a step-wise movement of the robot hand, taking the above two conditions into account. This leads to transition: $\boldsymbol{P_{H_{j+1}}} := \boldsymbol{P_{H_j}} + \boldsymbol{f_D}(\boldsymbol{P_{H_j}}, \boldsymbol{P_G}, \boldsymbol{OB}, \boldsymbol{VL_{R_j}}, \boldsymbol{S_f})$

Construction of the vector field

Method:

1. Initialise the vector field by creating an attractor centered at grasping position $\boldsymbol{P_G}$.
2. Superpose repellor fields, respectively centered at each obstacle point $\boldsymbol{P_O}$ from set $\boldsymbol{OB}$.
3. Robot hand is initially located at a collision-free position $\boldsymbol{P_{H_1}}$.
4. Define a cuboid subspace $\boldsymbol{E_{H_1}}$ centered at $\boldsymbol{P_{H_1}}$ and determine from it a finite, regular set of positions.

Construction of the vector field

5. Simulate movements of the robot hand to all discrete positions in $\boldsymbol{E_{H_1}}$. At each simulated position, compute the intersection of the simulated volume of the robot arm with the obstacle points from set $\boldsymbol{OB}$. If the intersection is non-empty, then create a further repellor field centered at the simulated hand position.

6. Superpose the new set of repellor fields with the previous vector field. This results in a new vector field $\boldsymbol{V(p)}$, e.g. containing also a movement vector for current robot hand position $\boldsymbol{P_{H_1}}$. See slide 32 of chapter 5.

Construction of the vector field

7. Define control function:

   $$f_D(P_{H_1}, P_G, OB, VL_{R_1}, S_f) := V(P_{H_1})$$

8. Actually move robot hand from P_{H_1} to P_{H_2} according to:
   $$P_{H_2} := P_{H_1} + f_D(P_{H_1}, P_G, OB, VL_{R_1}, S_f)$$

9. If the new robot hand position P_{H_2} is not the grasping position, then continue iteration at step 4, given this new hand position. Else stop movement and grasp the object.

Simulated movements of robot hand

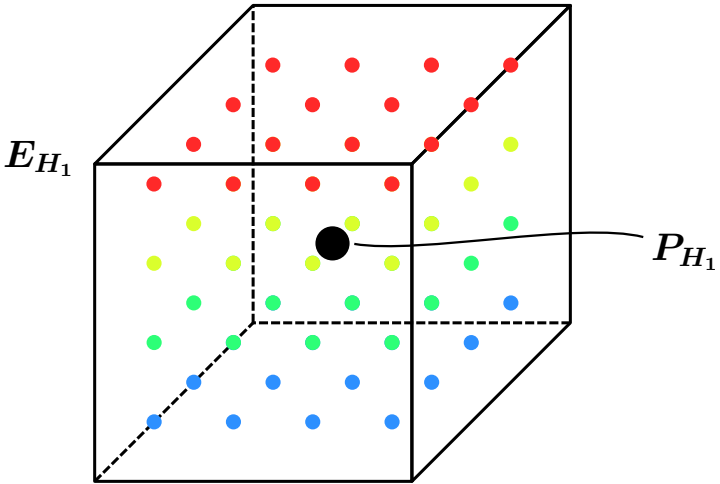
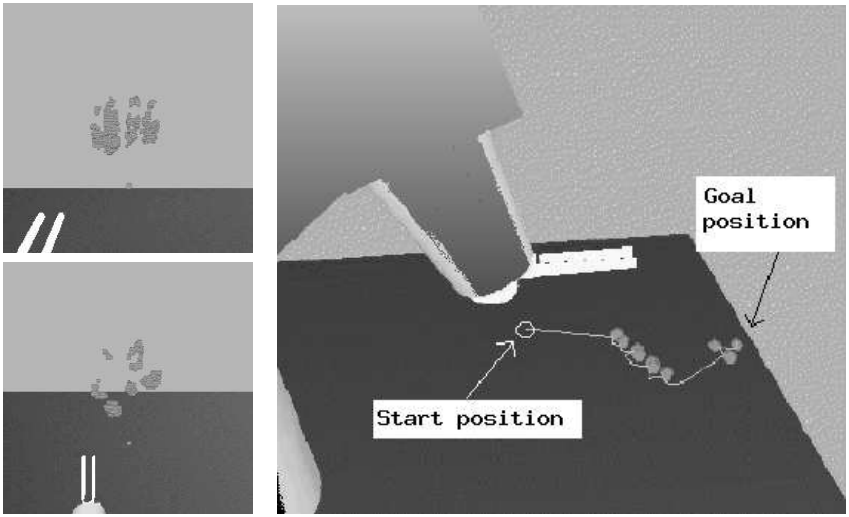


Illustration of navigating a robot hand



Literature

- J.-C. Latombe: Robot Motion Planning; Kluwer Academic Publishers, Boston, USA, 1991.
- J. Pearl: Heuristics - Intelligent Search Strategies for Computer Problem Solving; Addison-Wesley, 1985.
- Wiki A* search algorithm: `https://en.wikipedia.org/wiki/A*_search_algorithm`
- P. Lester: A* Pathfinding for Beginners; `https://csis.pace.edu/~benjamin/teaching/cs627/webfiles/Astar.pdf`
- Xueqiao Xu: Pathfinding Algorithms in Javascript; `https://qiao.github.io/PathFinding.js/visual/`
- Wiki Dynamic Programming: `https://en.wikipedia.org/wiki/Dynamic_programming`

7. Probabilistic Robot Localisation

Overview:

7.1 Motivation for probabilistic concepts

Overview:

- Sources of uncertainty

Sources of uncertainty

There are at least four sources of uncertainty.

- Sensor/Camera:
  - Noise
  - Limited resolution
  - Limited field of view
- Robot:
  - Loss due to tire friction
  - Unstable gripper fingers

Sources of uncertainty

- Environment:
  - Dynamics
  - Imponderable movements
  - Slippery floor
- Mathematical model:
  - Inaccurate approximations
  - Wrong assumptions

The purpose is to take these factors in robot localisation into account, using probabilistic concepts.

7.2 Basics of probability theory

Overview:

- Basic terms
- Basic axioms
- Bayes formula
- Normalisation factor
- Exemplary application of Bayes formula
- Extended Bayes formula

Basic terms

$\boldsymbol{L}$: Random variable, or random variables vector,

e.g., $\boldsymbol{L} := (\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\Theta})$

$\boldsymbol{P}(\boldsymbol{A})$: Probability of event $\boldsymbol{A}$,

e.g., $\boldsymbol{P}(\boldsymbol{L} = (\mathbf{100}, \mathbf{200}, \mathbf{90})) = \mathbf{10\%} \equiv \mathbf{0.1}$

$\boldsymbol{P}(\boldsymbol{A}|\boldsymbol{B})$: Probability of event $\boldsymbol{A}$ under the condition that event $\boldsymbol{B}$ is true (i.e. already happened)

Basic axioms

- $0 \leq P(A) \leq 1$
- $P(A \vee \neg A) = 1$, $P(A \wedge \neg A) = 0$
- $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$
- Product rule: $P(A \wedge B) = P(A|B) \cdot P(B)$
- Commutativity: $P(A|B) \cdot P(B) = P(A \wedge B) = P(B \wedge A) = P(B|A) \cdot P(A)$

Bayes formula

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

$P(A)$: A priori probability, no evidence from other events, e.g. no measurements

$P(A|B)$: A posteriori probability, including evidence from other events, e.g. measurements

For the complementary event $\neg A$, it holds:

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

Normalisation factor

The following constraint should hold:

$$1 = P(A|B) + P(\neg A|B) = \frac{P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A)}{P(B)}$$

The constraint is fulfilled, if:

$$P(B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A)$$

With this equation, the denominator $P(B)$ of Bayes formula serves as a normalisation factor.

Exemplary application of Bayes formula

Event $\boldsymbol{A}$ represents state „door is open“. Event $\boldsymbol{B}$ represents a measurement (e.g. distance to an obstacle.)

Example for probability values:

$$\boldsymbol{P}(\text{open}) = \mathbf{0.6} = \boldsymbol{P}(\boldsymbol{A}) \;, \quad \boldsymbol{P}(\text{closed}) = \mathbf{0.4} = \boldsymbol{P}(\neg \boldsymbol{A}) \;,$$

$$\boldsymbol{P}(\mathbf{300}\boldsymbol{cm}|\text{open}) = \mathbf{0.8} = \boldsymbol{P}(\boldsymbol{B}|\boldsymbol{A}) \;,$$

$$\boldsymbol{P}(\mathbf{300}\boldsymbol{cm}|\text{closed}) = \mathbf{0.1} = \boldsymbol{P}(\boldsymbol{B}|\neg \boldsymbol{A}) \;.$$

Applying Bayes formula:

$$\boldsymbol{P}(\text{open}|\mathbf{300}\boldsymbol{cm}) = \boldsymbol{P}(\boldsymbol{A}|\boldsymbol{B}) = \frac{\mathbf{0.8} \cdot \mathbf{0.6}}{\mathbf{0.8} \cdot \mathbf{0.6} + \mathbf{0.1} \cdot \mathbf{0.4}} = \mathbf{0.92}$$

Evidence from $\boldsymbol{B}$ leads to increased probability for $\boldsymbol{A}$.

Extended Bayes formula

A sequence of past measurements (evidences) $B_n, ..., B_1$, obtained at time points $t_n, ..., t_1$ with t_1 the farthest point back in time and t_n the current time point, should be included.

Therefore, an extension of Bayes formula is needed:

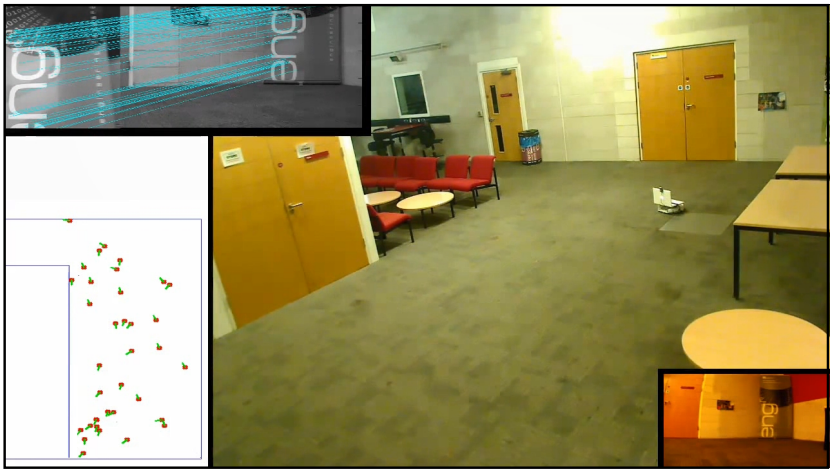
$$P(A|B_n, ..., B_1) = \frac{P(B_n|A, B_{n-1}, ..., B_1) \cdot P(A|B_{n-1}, ..., B_1)}{P(B_n|B_{n-1}, ..., B_1)}$$

7.3 Belief function for embedded robots

Overview:

- Illustration of Probabilistic Robot Localisation
- Requiring a recursive belief function
- Recursive belief equation
- Extended Bayes formula as belief function
- Integration over variable for previous state
- Independencies in probability functions
- Definition of recursive belief function

Illustration of Probabilistic Robot Localisation



Video from MikeHMurray's channel: `https://www.youtube.com/watch?v=QdU5Q5AKfQc`, MobileRobotLocalisationParticleFilter.avi

Requiring a recursive belief function

See Thrun (2001).

- Incessantly acting in the environment (embedding) allows incessant perception, which leads to incrementally improving the state estimation for a robot.
- The method is based on a so-called belief function for the validity (presence) of a certain state of the robot.
- The belief function will be formulated recursively, and then transformed into an iteration, for an incessant acquisition of environmental measurements and data fusion in the course of time.

Requiring a recursive belief function

Definition of a so-called belief function f_B:

$$f_B(s_t) := P(s_t|m_t, a_{t-1}, m_{t-1}, ..., m_2, a_1, m_1),$$

with state s_t, measurement m_t, action a_{t-1}, $\cdots$, respectively at time $t, t-1, \cdots$.

A recursive formulation is needed: $f_B(s_t) \doteq \mathcal{F}(f_B(s_{t-1}))$

The following reformulations and simplifications will lead to a definition for the functional $\mathcal{F}$.

Extended Bayes formula as belief function

Applying extended Bayes formula to belief function f_B:

$$f_B(s_t) := \frac{P(m_t|s_t, a_{t-1}, ..., m_1) \cdot P(s_t|a_{t-1}, m_{t-1}, ..., a_1, m_1)}{P(m_t|a_{t-1}, ..., m_1)}$$

With $P(m_t|s_t, a_{t-1}, ..., m_1) = P(m_t|s_t)$,

and $1/P(m_t|a_{t-1}, ..., m_1) =: \eta$, it follows:

$$f_B(s_t) := \eta \cdot P(m_t|s_t) \cdot P(s_t|a_{t-1}, m_{t-1}, ..., a_1, m_1)$$

Integration over variable for previous state

Relevant equation from probability theory, i.e. expansion of $\boldsymbol{P(A)}$ by integration over event $\boldsymbol{B}$:

$$\boldsymbol{P(A) = \int P(A \wedge B) dB = \int \big(P(A|B) \cdot P(B)\big) dB}$$

See also theorem of total probability in next slide.

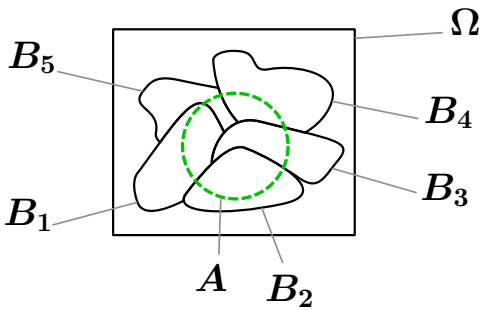
Expansion of $\boldsymbol{P(s_t|a_{t-1}, m_{t-1}, ..., a_1, m_1)}$ by integration over state variable $\boldsymbol{s_{t-1}}$:

$$\boldsymbol{f_B(s_t) := \eta \cdot P(m_t|s_t) \cdot \int \big(P(s_t|s_{t-1}, a_{t-1}, m_{t-1}, ..., a_1, m_1) \cdot P(s_{t-1}|a_{t-1}, m_{t-1}, ..., a_1, m_1)\big) ds_{t-1}}$$

Integration over variable for previous state

Theorem of total probability: For $i \neq j, B_i \cap B_j = \{\}$, i.e. events $B_1, \ldots, B_5$ are pairwise disjunct, and $A \subseteq B_1 \cup B_2 \cdots \cup B_5$.

$$\Rightarrow P(A) = \sum_{i=1}^{n} P(A \wedge B_i) = \sum_{i=1}^{n} P(A|B_i) \cdot P(B_i)$$



Independencies in probability functions

Consider:

$$\boldsymbol{P(s_t|s_{t-1}, a_{t-1}, m_{t-1}, ..., a_1, m_1) = P(s_t|a_{t-1}, s_{t-1})}$$

i.e. the most recent state and action completely determine the current state (Markov property).

Moreover:

$$\boldsymbol{P(s_{t-1}|a_{t-1}, m_{t-1}, ..., m_1) = P(s_{t-1}|m_{t-1}, a_{t-2}, ..., a_1, m_1)}$$

i.e. earlier state $\boldsymbol{s_{t-1}}$ is independent from subsequent action $\boldsymbol{a_{t-1}}$.

Definition of recursive belief function

Consequently:

$$\boldsymbol{f_B(s_t) := \eta \cdot P(m_t|s_t) \cdot \int \big(P(s_t|a_{t-1}, s_{t-1}) \cdot P(s_{t-1}|m_{t-1}, a_{t-2}, ..., m_1)\big) ds_{t-1}}$$

According to definition of belief function in slide 15, it holds

$$\boldsymbol{P(s_{t-1}|m_{t-1}, a_{t-2}, ..., m_1) =: f_B(s_{t-1})}$$

Thus the Recursive Belief Function (RBF) follows:

$$\boldsymbol{f_B(s_t) := \eta \cdot P(m_t|s_t) \cdot \int \big(P(s_t|a_{t-1}, s_{t-1}) \cdot f_B(s_{t-1})\big) ds_{t-1}}$$

7.4 Perception and movement models

Overview:

- Description of a statistical perception model
- Acquisition of a statistical perception model
- Examples for a statistical perception model
- Applying measurement to perception model
- Description of statistical movement model
- Acquisition of a statistical movement model
- Examples for a statistical movement model

Description of a statistical perception model

- Statistical perception (SP) model $\boldsymbol{P(m_t|s_t)}$ is a component of the RBF.
- Time index $\boldsymbol{t}$ indicates the time of the current state respectively the current measurement.
- SP model is assumed to be valid independent from time, i.e. a so-called stationary model.
- Model acquisition requires exemplary measurements in the offline phase.
- For this purpose, a sensor/camera, mounted on the robot, will perceive its environment.

Acquisition of a statistical perception model

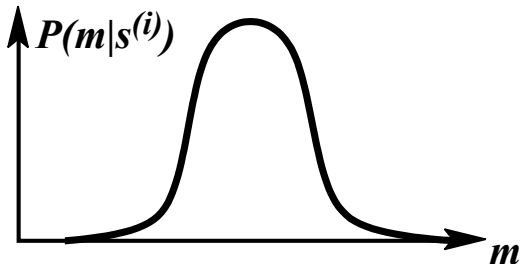
- Place the robot at a position in space, i.e. manually put the robot into a certain state $\boldsymbol{s}^{(i)}$.
- During a certain interval of time a spectrum of measurements is obtained, which results in the probability function $\boldsymbol{P}(\boldsymbol{m}|\boldsymbol{s}^{(i)})$, i.e. the statistical perception model at state $\boldsymbol{s}^{(i)}$ with variable $\boldsymbol{m}$.
- Repetition of this procedure for all (discrete) positions in the space, which will lead to the probability function $\boldsymbol{P}(\boldsymbol{m}|\boldsymbol{s})$ with variables $\boldsymbol{m}, \boldsymbol{s}$.

Examples for a statistical perception model

Example 1:

LASER ray with the device at position $\boldsymbol{p}^{(i)}$ in direction $\boldsymbol{\alpha}^{(i)}$, i.e. state $\boldsymbol{s}^{(i)} := (\boldsymbol{p}^{(i)}, \boldsymbol{\alpha}^{(i)})$.

Measurements during a certain time interval leads to the distribution $\boldsymbol{P}(\boldsymbol{m}|\boldsymbol{s}^{(i)})$.

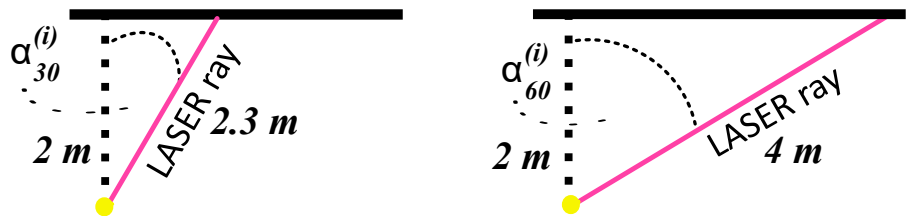


Examples for a statistical perception model

Example 2:

LASER scan of rays in interval $[-90, 90]$ for angle α, discretised in $1°$.

Define state $s^{(i)} := (p^{(i)}, \beta^{(i)})$, with orientation $\beta^{(i)}$ of the scanner.



Probability function with 181 components in the measurement part:

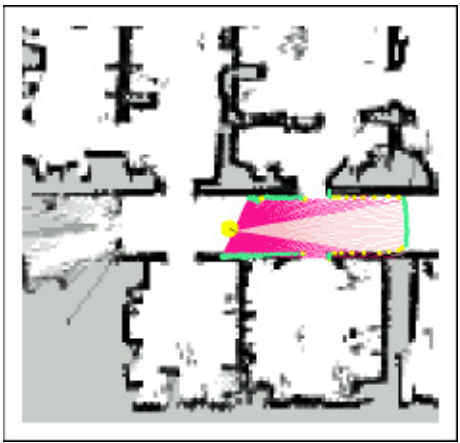
$$P(m|s^{(i)}) := P\big((m_{-90}, \cdots, m_{90})|s^{(i)}\big)$$

Examples for a statistical perception model

Example 2:

LASER scan at position $\boldsymbol{p}^{(i)}$ and orientation $\beta^{(i)}$, for modeling purpose.

(from Thrun 2001)

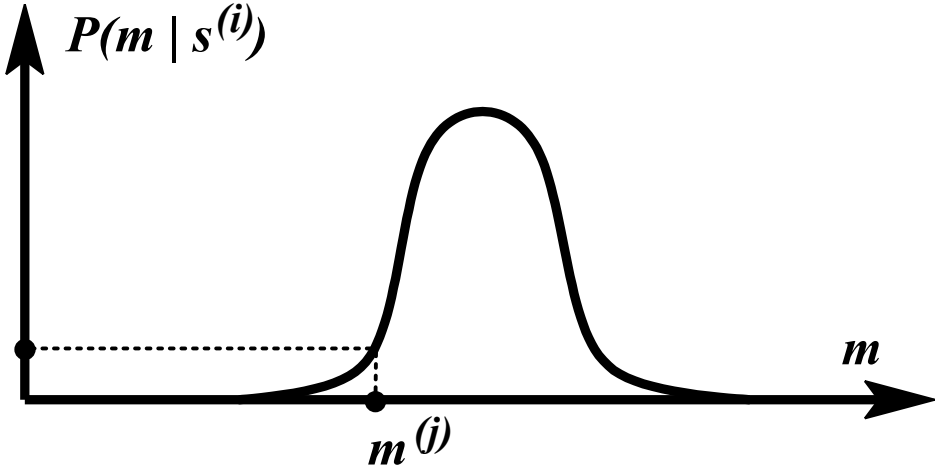


Applying measurement to perception model

- In the application phase, a measurement $\boldsymbol{m}^{(j)}$ will be taken in an unknown state $\boldsymbol{s}^{(j)}$.
- The measurement will be used as input in the probability function $\boldsymbol{P}(\boldsymbol{m}^{(j)}|\boldsymbol{s}^{(i)})$, for hypothesizing $\boldsymbol{s}^{(i)}$ as the current robot state.
- The resulting probability value is a measurement-based belief of the robot being in the state $\boldsymbol{s}^{(i)}$.

Applying measurement to perception model

Refer to example 1:



Applying measurement to perception model

Refer to example 2:

- Take LASER scans at all discrete positions $\boldsymbol{p}^{(j)}$ in the free areas of the office environment (rooms, corridor).
- Put the measurements respectively into the SP model for state $\boldsymbol{s}^{(i)}$, i.e. from slide 26.
- The probability value can be mapped into a gray-level in interval $[\mathbf{0}, \mathbf{255}]$ of natural numbers, with high values („dark pixels“) indicating high belief in the hypothesized state, respectively low values („bright pixels“) indicating low belief.
- ...

Applying measurement to perception model

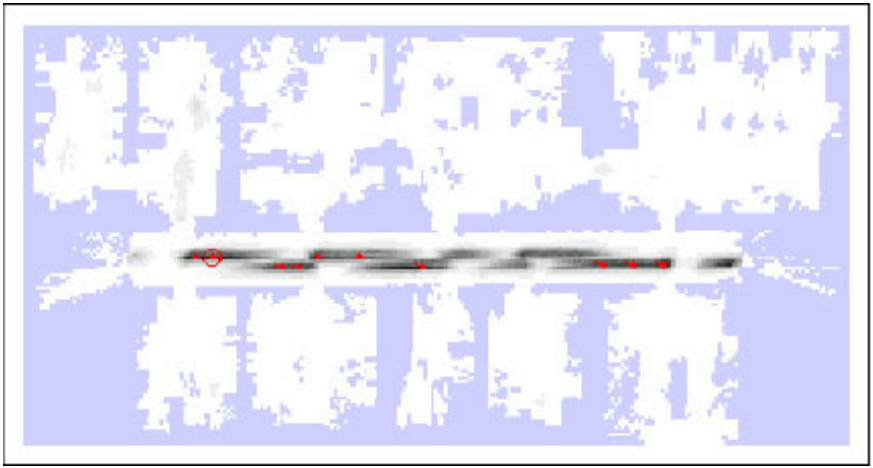
Refer to example 2:

- ...
- The darker the gray-level (assumed close to 255 !) the higher the belief of the robot being located in the position $\boldsymbol{p}^{(i)}$.
- Discussion of two wave-like bands in corridor center.

Applying measurement to perception model

Beliefs of the robot, being located in position $\boldsymbol{p}^{(i)}$.

(from Thrun 2001)



Description of statistical movement model

- Statistical movement (SM) model $\boldsymbol{P(s_t|a_{t-1}, s_{t-1})}$ is a component of the RBF.
- Time index $\boldsymbol{t-1}$ indicates the previous time of considered state respectively action, and index $\boldsymbol{t}$ indicates the current time of the state.
- SM model is assumed to be valid independent from time, i.e. again a stationary model.
- Model acquisition requires exemplary movements in the offline phase.

Acquisition of a statistical movement model

- Place the robot at a position in space, i.e. manually put robot into a certain state $\boldsymbol{s}^{(i)}$.
- Command the robot to perform action $\boldsymbol{a}^{(j)}$, then robot moves and finally reaches a new state $\boldsymbol{s}'$.
- Capture and save the new state.
- Manually transfer the robot into original state $\boldsymbol{s}^{(i)}$.

Acquisition of a statistical movement model

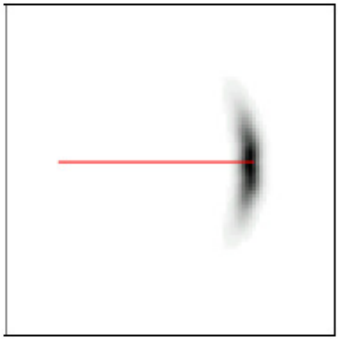
- Repetition of this procedure will lead to the probability function $\boldsymbol{P}(\boldsymbol{s'}|\boldsymbol{a}^{(j)}, \boldsymbol{s}^{(i)})$ with variable $\boldsymbol{s'}$.
- Repetition of this procedure for all (discrete) states $\boldsymbol{s}^{(i)}$ and actions $\boldsymbol{a}^{(j)}$ will lead to the probability function $\boldsymbol{P}(\boldsymbol{s'}|\boldsymbol{a}, \boldsymbol{s})$ with variables $\boldsymbol{s'}, \boldsymbol{a}, \boldsymbol{s}$, i.e. probability of state $\boldsymbol{s'}$ following the state $\boldsymbol{s}$ when trying to execute action $\boldsymbol{a}$.

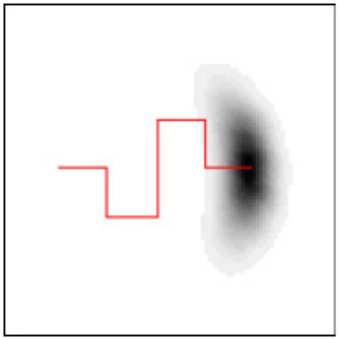
Examples for a statistical movement model

Example 1:

Degree of uncertainty of final robot location depends on the course of travelled path.

(from Thrun 2001)



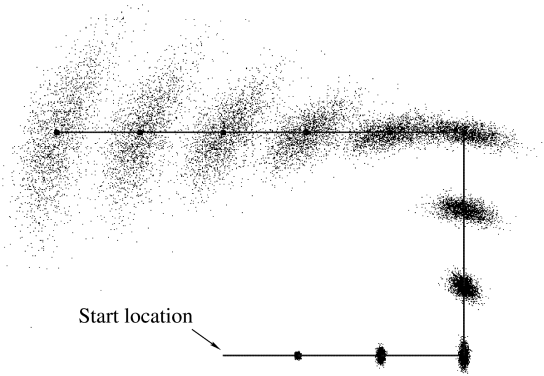


Examples for a statistical movement model

Example 2:

Degree of uncertainty of final robot location increases proportional with the length of travelled path.

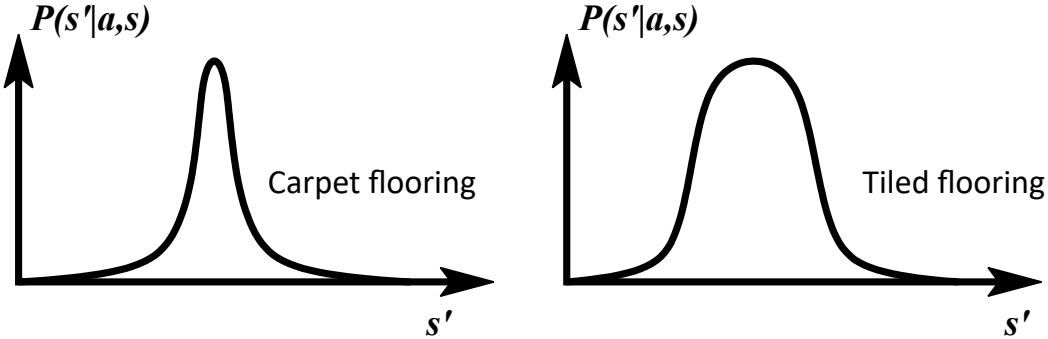
(from Thrun 2001)



Examples for a statistical movement model

Example 3:

Degree of uncertainty of final robot location depends on the material of the ground, e.g. carpet or tiled flooring.



7.5 Approach at a glance

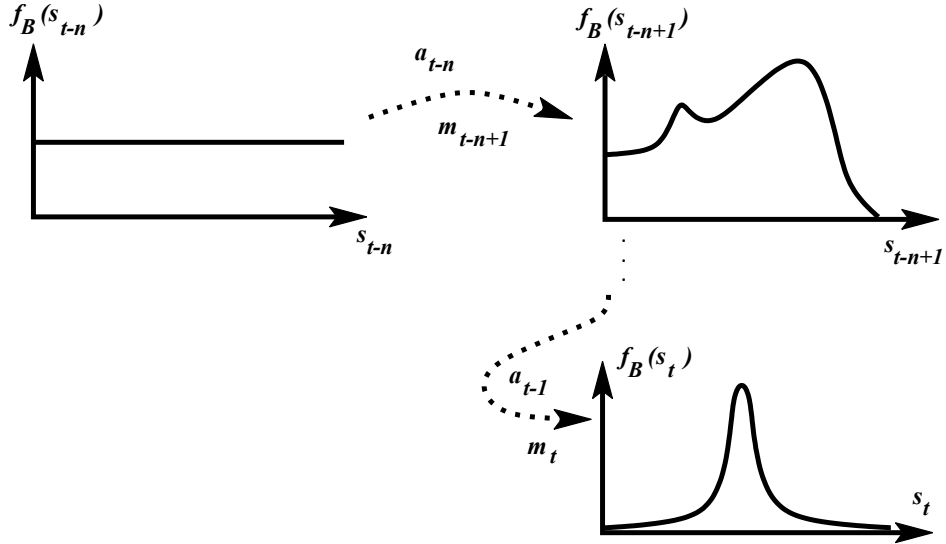
Overview:

- Iterative adaption of belief function
- Basic processes involved in an iteration
- Basic processes in Standard Kalman Filter
- Approximation of distributions by particles
- Approximating belief function by particles
- Using SP model for weighting particles

Iterative adaption of belief function

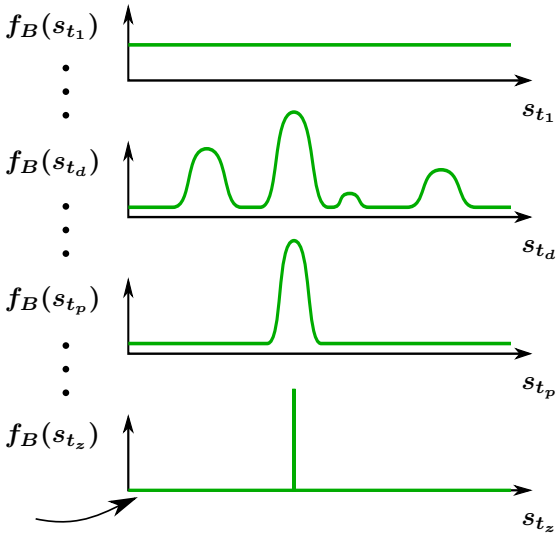
- At first, robot doesn't know where it is, that means a uniform distribution of possible states.
- Through incessantly performing action and perception in the local environment, the spectrum of possible states will be narrowed.

Iterative adaption of belief function



Iterative adaption of belief function

Belief functions in the course of time

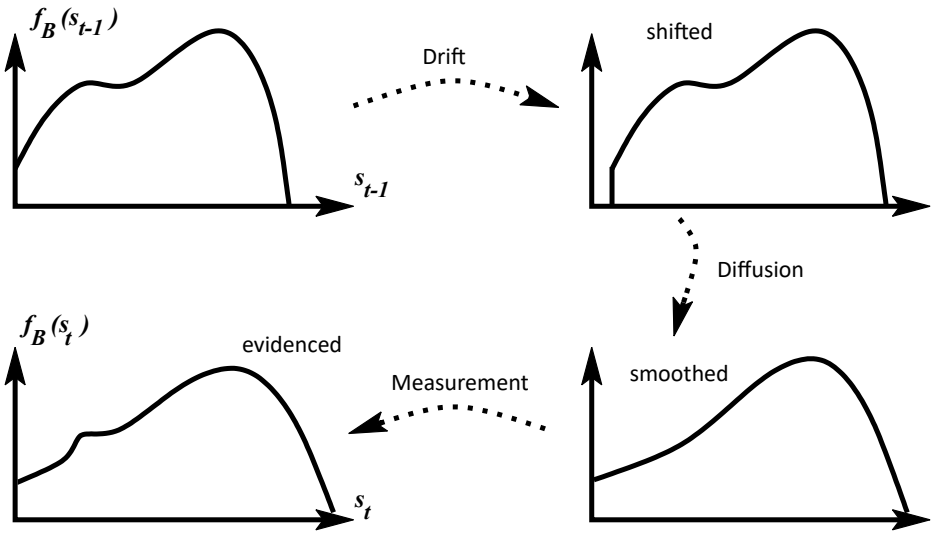


Desirable but unrealistic

Basic processes involved in an iteration

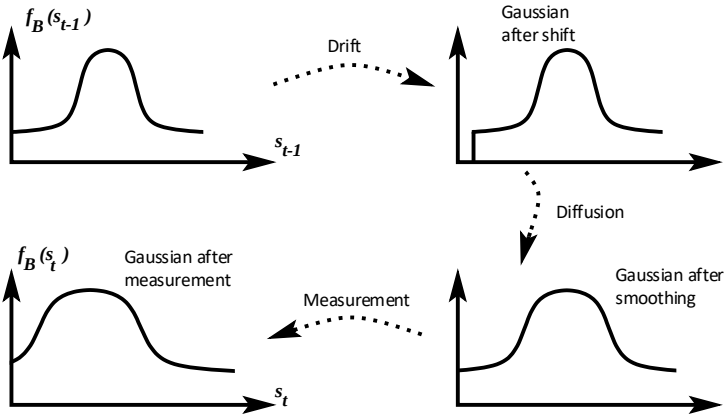
- Deterministic drift; through action $\boldsymbol{a_{t-1}}$ the $\boldsymbol{f_B}$ function is shifted.
- Stochastic diffusion; according to SM model the successive state is uncertain which results in smoothing the $\boldsymbol{f_B}$ function.
- Effect of measurement; in the new state the measurement $\boldsymbol{m_t}$ of the environment has an effect (evidence) on the $\boldsymbol{f_B}$ function via the SP model.

Basic processes involved in an iteration



Basic processes in Standard Kalman Filter

The so-called „Standard Kalman Filter“ applies the same three basic processes, but enforces Gaussian distributions.

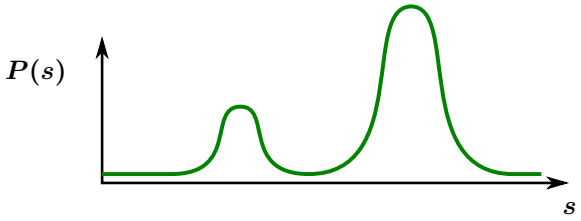


Approximation of distributions by particles

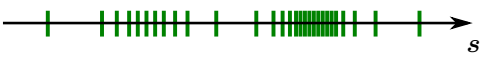
- SP model and SM model must not be in analytical form, but can be approximated as sets of sample elements (equivalent to sets of particles). The same holds for the belief function $\boldsymbol{f_B}$.
- A particle-based representation is useful when the generic types/forms of the SP model, SM model, and of the $\boldsymbol{f_B}$ function are unclear/unknown. E.g. Gaussian distributions may be not appropriate.
- Generally, the implementation of the iterative adaption of the RBF is carried out by processing and modifying a set of particles. The approach is called „particle filter“ (as opposed to „Kalman filter“).

Approximation of distributions by particles

Analytical form



Approximation
with particles



Approximating belief function by particles

$$f_B(s_t) := \eta \cdot P(m_t|s_t) \cdot \int \big(P(s_t|a_{t-1}, s_{t-1}) \cdot f_B(s_{t-1})\big) ds_{t-1}$$

Approximation through a set of weighted particles:

$f_B(s) := \{(s^{(i)}, w^{(i)})\}_{i=1,....,k}$,

$s^{(i)}$ is a sample element representing a possible discrete state s,

$w^{(i)}$ is an additional weighting factor for the sample element.

Approximating belief function by particles

Distribution of sample elements (particles) according to:

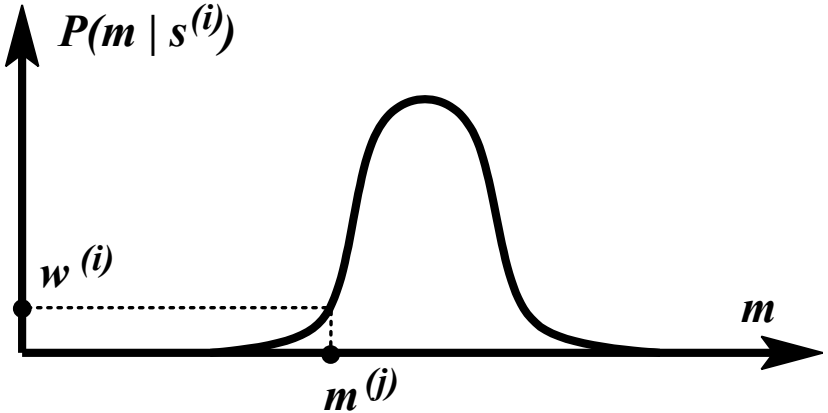
$$\int \big(\boldsymbol{P}(\boldsymbol{s}_t|\boldsymbol{a}_{t-1}, \boldsymbol{s}_{t-1}) \cdot \boldsymbol{f}_B(\boldsymbol{s}_{t-1})\big) d\boldsymbol{s}_{t-1} =$$

$$\boldsymbol{P}(\boldsymbol{s}_t|\boldsymbol{a}_{t-1}, \boldsymbol{m}_{t-1}, \boldsymbol{a}_{t-2}, ..., \boldsymbol{m}_1) =: \boldsymbol{P}_{apriori}(\boldsymbol{s}_t)$$

Defining weights for the particles according to: $\boldsymbol{w}^{(i)} := \boldsymbol{P}(\boldsymbol{m}^{(j)}|\boldsymbol{s}^{(i)})$, i.e. determine probability value by applying current measurement $\boldsymbol{m}^{(j)}$ in the SP model of state $\boldsymbol{s}^{(i)}$.

Using SP model for weighting particles

Acquired SP model at state $s^{(i)}$. Current measurement $m^{(j)}$ will result in weight $w^{(i)}$ for this assumed state.



Using SP model for weighting particles

- Related to slide 49, the measurement $\boldsymbol{m}^{(j)}$ yields a low probability/weight $\boldsymbol{w}^{(i)}$ for particle $\boldsymbol{s}^{(i)}$.
- Therefore, it is unlikely that the robot currently is in state $\boldsymbol{s}^{(i)}$.
- Consequently, the state hypothesis $\boldsymbol{s}^{(i)}$ should only play a low role in the adaption of the set of particles (approximating the belief function).

7.6 Particle filter algorithm and applications

Overview:

- Specifying the particle filter algorithm
- Implementation of particle filter algorithm
- Particle-based illustration of basic processes
- Localisation of robot in lab environment
- Localisation of robot in crowded museum
- Particle filter for object tracking
- Related technical terms for the approach
- Recent research to enhance efficiency

Specifying the particle filter algorithm

- At time $t-1$, the function $f_B(s_{t-1})$ is approximated by k weighted particles $\{(s^{(i)}, w^{(i)})\}, i \in \{1, \cdots, k\}$.
- Some particles may be considered multiple times, while other particles may be totally ignored. This depends on the weights of the particles.
- The considered particles are subject to deterministic drift and stochastic diffusion.
- The drift is caused by a movement command sent to the robot.

Specifying the particle filter algorithm

- For diffusing the particles the SM model is applied.
- The new set of particles has again the cardinality k and will serve as approximation of the a priori part of the new belief function at the new time t.
- In the new state, a new measurement will taken.
- The measurement will be used in the SP model to obtain weights respectively for each new particle (i.e. for each new state hypothesis), leading to the final version of the approximation of the new belief function at the new time t.

Implementation of particle filter algorithm

Input:

Weighted particle set $S := \{(s^{(i)}, w^{(i)})\}_{i=1,\cdots,k}$

Action a

Measurement m after performing the action

Output:

New particle set S'

Implementation of particle filter algorithm

Procedure $PFA(S, a, m)$

$S' := \{\}$

For $i := 1$ **to** k

Begin

Select probabilistic. $(s, w) \in S$, based on the weights $w^{(1)}, ..., w^{(k)}$

Select probabilistically s', according to the distribution $P(s'|a, s)$

$w' := P(m|s')$

Refresh $S' := S' \cup (s', w')$

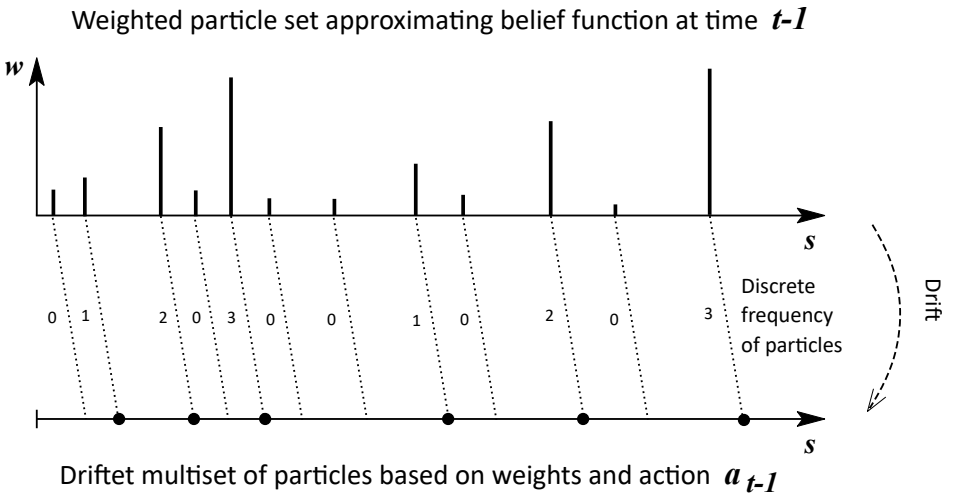
End

Normalise weights of particle set in S'

Return S'

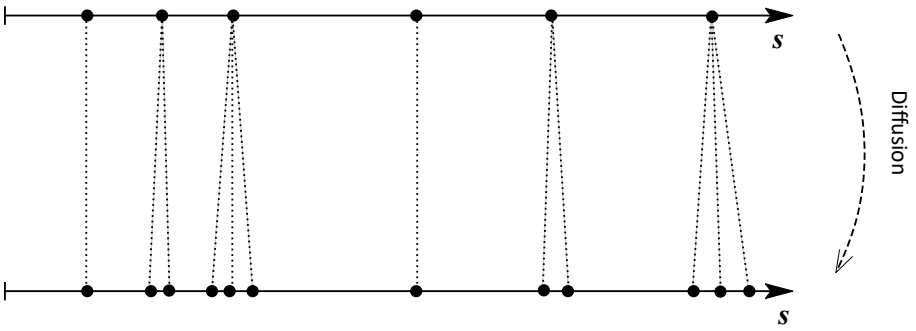
Particle-based illustration of basic processes

Illustration of particle filter algorithm for 12 particles.



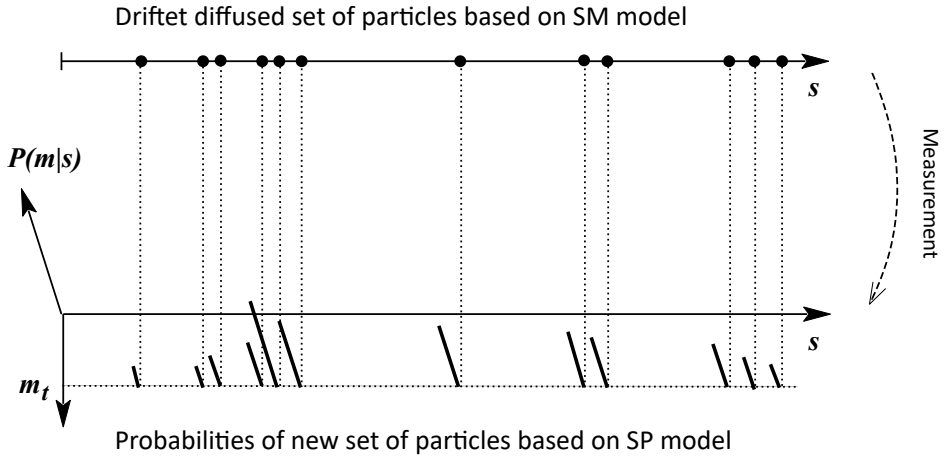
Particle-based illustration of basic processes

Driftet multiset of particles based on weights and action a_{t-1}

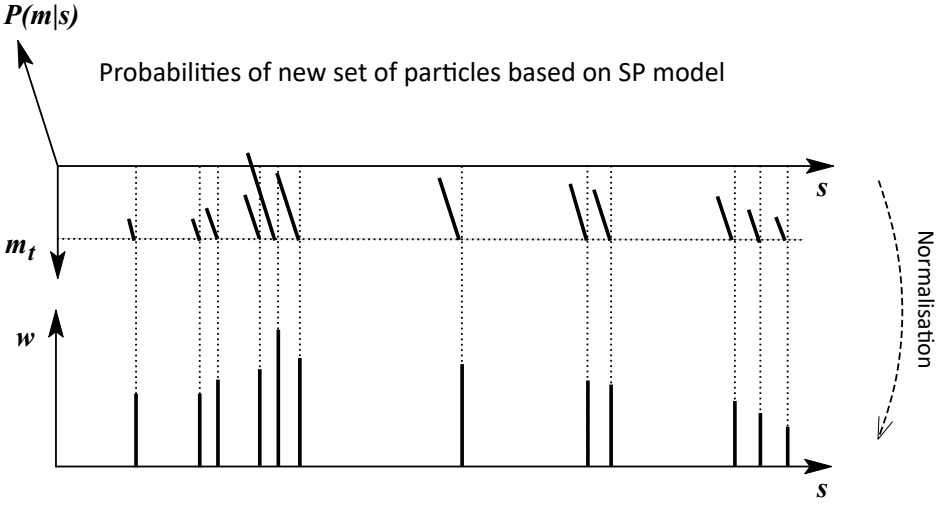


Driftet diffused set of particles based on SM model

Particle-based illustration of basic processes

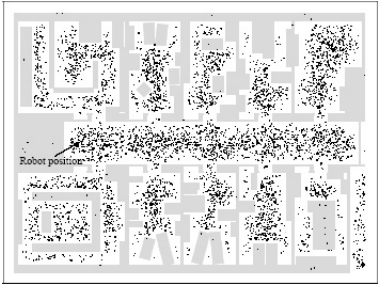


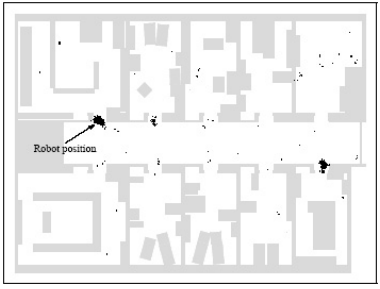
Particle-based illustration of basic processes

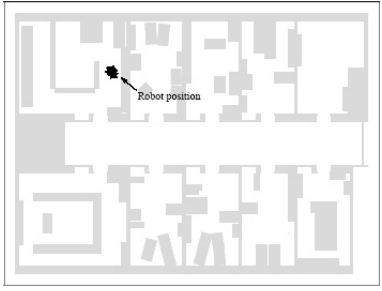


Weighted particle set approximating belief function at time t

Localisation of robot in lab environment



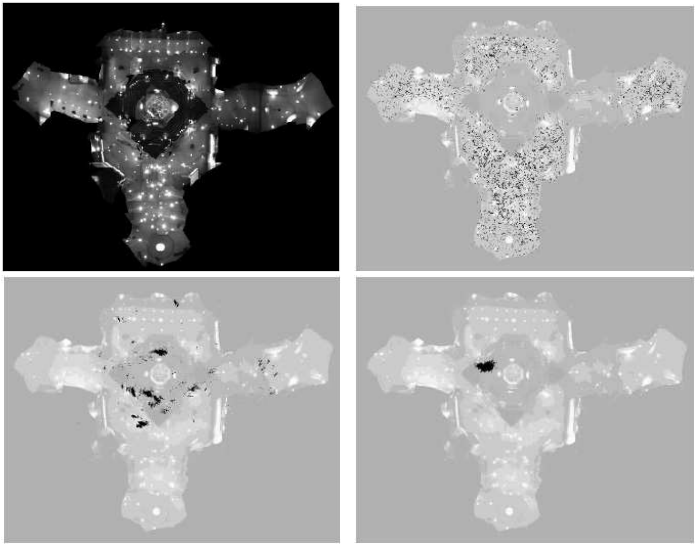




(from Thrun 2001)

Localisation of robot in crowded museum

(from Thrun 2001)



Particle filter for object tracking

- Particle filter algorithm can also be used for tracking objects in an image sequence.

  See M. Isard, et al.: CONDENSATION - CONditional DENSity estimATION; Int. Journal of Computer Vision, 29:5-28, 1998.

Related technical terms for the approach

- Iterative adaption of the belief function
- Particle Filter
- **CON**ditional **DEN**Sity estim**ATION**
- **S**equential **M**onte **C**arlo method

Recent research to enhance efficiency

- Selection of appropriate actions for quick convergence of belief function $\boldsymbol{f_B}$.
- Dynamic variation of cardinality of particle set.
- Approximation of distribution function through a sum of weighted Gaussian functions with individual support (i.e. Radial Basis Function Network).

Literature

- G. Kitigawa: Monte-Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models; Journal of Computational and Graphical Statistics, 5:1-25, 1996.
- S. Thrun, et al.: Robust Monte Carlo Localization for Mobile Robots; Artificial Intelligence, 128:99-141, 2001.
- Wikipedia (-> Sequential Monte Carlo (SMC) Methods / Particle Filter).
- MikeHMurray's channel (uploaded to YouTube): Mobile Robot Localisation using a Particle Filter, `https://www.youtube.com/watch?v=QdU5Q5AKfQc` .

8. Online Learning for Robot Navigation

Overview:

8.1 Task and problems of robot navigation

Overview:

- Task of robot navigation
- Problems of robot navigation

Task of robot navigation

- After an environmental map has been created (see chap. 5),
- and a path from start to target position has been planned (see chap. 6),
- and the robot has localised itself (see chap. 7),
- then the robot should carry out a deliberative task, e.g. navigating to certain target position(s).

Problems of robot navigation

- Recognition and localisation of landmarks.
- Incessant self-localisation during robot movement.
- Recognition of obstacles and strategies for bypassing.
- Online learning of optimal robot behaviors in case of imponderable new situations and degradation of the quality of the path plan.

8.2 Principle of Online Learning

Overview:

- Prerequisites based on path planning
- Time reference in the operational phase
- State transitions and elementary benefits
- Definition of the total benefit function
- Role of the total benefit function
- ...

8.2 Principle of Online Learning

Overview:

- ...
- Requiring a recursive total benefit function
- Constructing a recursive total benefit function
- Methodological reference to Dynamic Optimisation
- Iteration of total benefit function

Prerequisites based on path planning

- States $\mathcal{S}$
- Actions $\mathcal{A}$
- State transition function f_T
- Decision function f_D

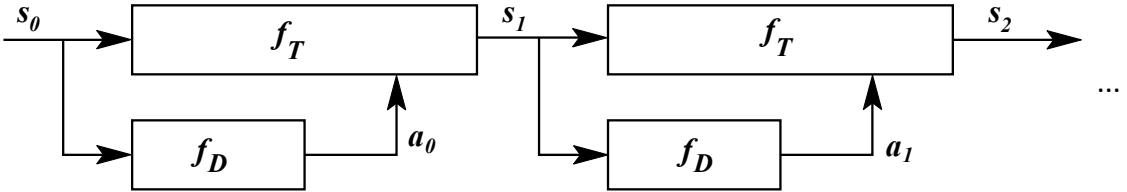
The decision function must be adapted, expanded, improved in the online (operational) phase, i.e. „Online Learning“.

For simplification, the states are assumed to be directly available in the computer, i.e. perception function f_P is the identity.

Time reference in the operational phase

- In the operational phase, i.e. in real robot operation, the states and actions are time-related.
- Actual states $s_t \in \mathcal{S}$
- Actual actions $a_t \in \mathcal{A}$
- The relation between them is through the actual decision function $a_t := f_D(s_t)$.
- After a constant time interval (assumed $\Delta t = 1$), a new state is obtained: $s_{t+1} := f_T(s_t, a_t)$.

State transitions and elementary benefits



- For the current point of time, $t = 0$. For some time in future, $t > 0$.
- Single state transition leads to an elementary benefit, which is formalised by function $f_{EB}(s_t, a_t)$.

  Elementary benefit (utility), e.g. avoiding collision with an obstacle and getting a bit closer to the target position.

Definition of the total benefit function

The total benefit function f_{TB} is defined as an infinite summation of weighted elementary benefits:

$$f_{TB}(s_0, f_D) := \sum_{t=0}^{\infty} \gamma^t \cdot f_{EB}(s_t, a_t)$$

- Discount factor γ, with $0 < \gamma < 1$.
- The expression γ^t is considered as weight of an elementary benefit at the time t.
- The lower t, i.e. the closer to the current time, the more important the to be gathered elementary benefit.
- The higher t, i.e. the farther in the future, the less relevant the expected elementary benefit.

Definition of the total benefit function

- With increasing time t, the weight γ^t is decreasing, and thus the total benefit function should converge to a finite value.
- The alteration of the decision function f_D will have an effect on this finite value of the total benefit function f_{TB}.
- The decision function f_D has to be automatically adapted to f_{D_*}, i.e. optimised/learned, such that for each initial state $s_0 \in \mathcal{S}$, it holds:

$$f_{TB}(s_0, f_{D_*}) := max_{f_D}\{f_{TB}(s_0, f_D)\}$$

Definition of the total benefit function

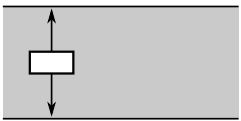
The total benefit function $\boldsymbol{f_{TB}}$ could include, for example:

- number of collisions (the fewer the better), and
- size of covered area for the purpose of surveillance (the bigger the better).

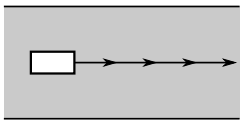
And these criteria are subject to weights γ^t in the course of time $\boldsymbol{t}$.

Role of the total benefit function

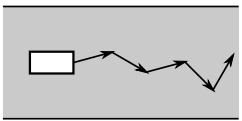
Examples of total benefits, based on different decision functions f_{D_j}.



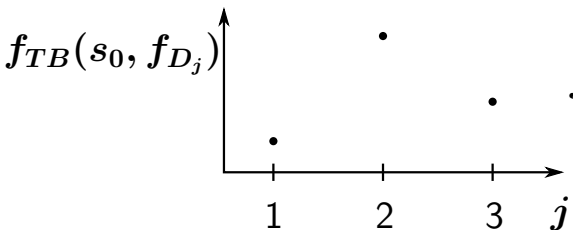
$f_{TB}(s_0, f_{D_1})$: low



$f_{TB}(s_0, f_{D_2})$: high



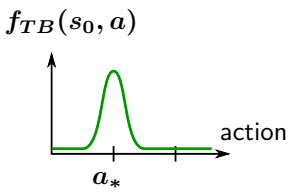
$f_{TB}(s_0, f_{D_3})$: medium

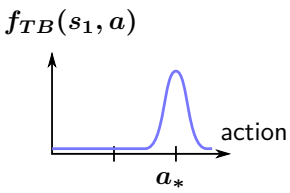


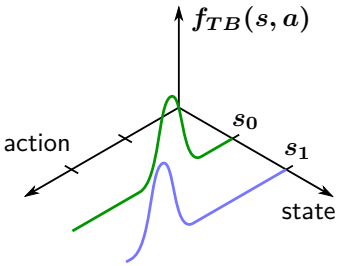
$f_{D_*} := f_{D_2}$

Role of the total benefit function

Action selection, assuming known total benefit function.
(For short, $\boldsymbol{a} := \boldsymbol{f_D}(\boldsymbol{s})$ is used in signature of $\boldsymbol{f_{TB}}$.)







Regrettably, $\boldsymbol{f_{TB}}$ is not known due to infinite summation.

Role of the total benefit function

Problem:

Obtain optimal robot behavior (i.e. find optimal decision function $\boldsymbol{f_{D_*}}$, which leads to the optimal actions $\boldsymbol{a_*}$).

But the total benefit function $\boldsymbol{f_{TB}}$ is explicitly not known.

Requiring a recursive total benefit function

For learning the optimal decision function f_{D_*}, the robot must explore the environment and thereby integrate elementary benefits obtained through transitions between certain states.

A recursive formulation of f_{TB} is needed, which enables the incessant integration of elementary benefits in the course of time:

$$f_{TB}(s, f_D) \doteq \mathcal{F}(f_{TB}(s', f_D))$$

With state s', following the state s when applying decision function f_D and performing the determined action.

Constructing a recursive total benefit function

$$
\begin{aligned}
f_{TB}(s_0, f_D) &:= \sum_{t=0}^{\infty} \gamma^t \cdot f_{EB}(s_t, a_t) \\
&= f_{EB}(s_0, a_0) + \gamma \cdot \sum_{t=0}^{\infty} \gamma^t \cdot f_{EB}(s_{t+1}, a_{t+1}) \\
&= f_{EB}(s_0, a_0) + \gamma \cdot f_{TB}(s_1, f_D)
\end{aligned}
$$

In the following, symbols are used in short form:

$$s := s_0 \ , \quad s' := s_1 \ , \quad a := a_0 \ , \quad f_{TB}(s, a) := f_{TB}(s, f_D(s)) \ ,$$

$$f_{TB_*}(s) := f_{TB}(s, f_{D_*}(s)) = f_{TB}(s, a_*)$$

Methodological reference to Dynamic Optimisation

Recursive total benefit function, based on optimal decision function f_{D_*}:

$$f_{TB_*}(s) := max_{a \in \mathcal{A}}\{f_{EB}(s, a) + \gamma \cdot f_{TB_*}(s'))\}$$

Remarks:

- Compare above recursive equation with the recursive equation in Dynamic Optimisation for Path Planning (subchapter 6.3).
- $\Rightarrow$ Formal analogy, apart from maximising benefits versus minimising costs.

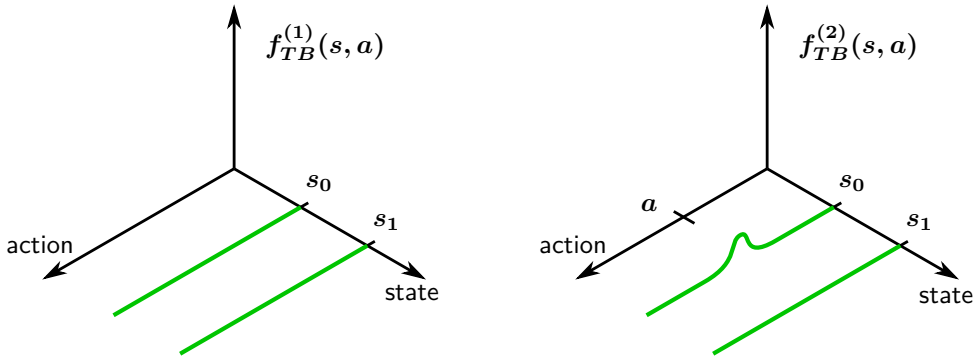
Methodological reference to Dynamic Optimisation

- Here, however, $\boldsymbol{f_{EB}(s, a)}$ can be determined only for the current combination of $\boldsymbol{s}$ and $\boldsymbol{a}$, whereas in Path Planning the values for all expressions $\boldsymbol{f_{i-1}(x_{i-1}, x_i)}$ are known in advance.
- Consequently, the specific type of Dynamic Optimisation, i.e. beginning with target position and iteratively backtrack to the start position, can not be applied here (for online learning).
- Instead, the concept of so-called „Function Iteration“ is applicable.

Methodological reference to Dynamic Optimisation

- Iteratively, the total benefit function is updated by including the current elementary benefit, and a decision is made as to the next action on the basis of the current estimation of the total benefit function.
- This approach is another type of a wide range of possible Dynamic Optimisation techniques (see next section and Barto et al. 1995).
- The approach also has similarities with so-called „Expectation-Maximisation“ algorithms (see in engl. Wiki).

Iteration of total benefit function



$f_{TB}^{(1)}$ is updated to $f_{TB}^{(2)}$, by incuding elementary benefit f_{EB} obtained through an action a in state s_0.

8.3 Types of Dynamic Optimisation

Overview:

- Synchronous Dynamic Optimisation (SDO)
- Gauss-Seidel Dynamic Optimisation (GDO)
- Asynchronous Dynamic Optimisation (ADO)
- Online Learning by ADO

Synchronous Dynamic Optimisation

Definition of function iteration according to SDO:

- Only function values from iteration i will be used (see index i in right side of eqn.) to compute function values for iteration $i+1$.
- The values of function $f_{TB_*}^{(i)}(s)$ will be updated for all states s of $\mathcal{S}$ in arbitrary order (or in parallel).

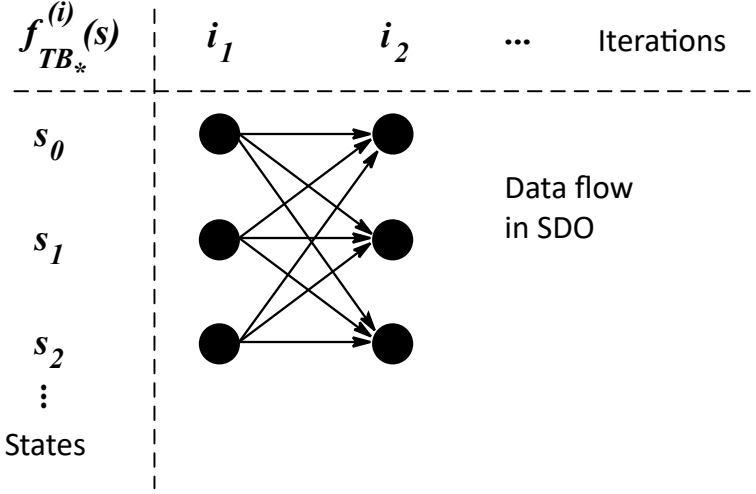
$$f_{TB_*}^{(i+1)}(s) := max_{a \in \mathcal{A}} \{ f_{EB}(s,a) + \gamma \cdot f_{TB_*}^{(i)}(s') \},$$

with $s' := f_T(s,a)$

Hint: In the following three tables, the identifiers for states are taken arbitrarily and the order of the states does not represent a trajectory.

Synchronous Dynamic Optimisation

Illustration of function iteration according to SDO:



Gauss-Seidel Dynamic Optimisation

Definition of function iteration according to GDO:

- Function values both from iteration $\boldsymbol{i}$ and $\boldsymbol{i+1}$ (considered in symbol $\sim$ in right side of equation) will be used to compute function values in iteration $\boldsymbol{i+1}$.
- The states $\boldsymbol{s}$ are assumed to be serially numbered according to a function $\boldsymbol{f_I}$.
- The values of function $\boldsymbol{f_{TB_*}^{(i)}(s)}$ will be updated for states $\boldsymbol{s}$ of $\boldsymbol{\mathcal{S}}$ in the succession given by function $\boldsymbol{f_I}$.
- Thereby the most recent available function values of $\boldsymbol{f_{TB_*}}$ can be used.

Gauss-Seidel Dynamic Optimisation

$$f_{TB_*}^{(i+1)}(s) := max_{a \in \mathcal{A}}\{f_{EB}(s,a) + \gamma \cdot \tilde{f_{TB_*}}(s')\},$$

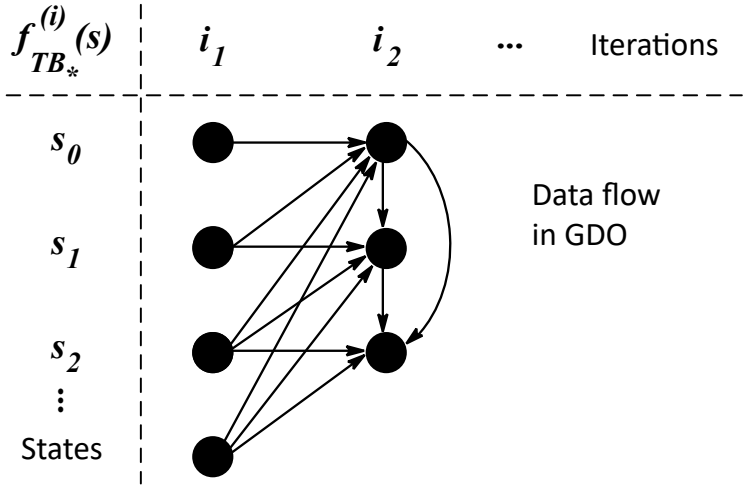
with $s' := f_T(s,a)$.

If $(f_I(s') > f_I(s))$, then $\tilde{f_{TB_*}}(s') := f_{TB_*}^{(i)}(s')$.

If $(f_I(s') < f_I(s))$, then $\tilde{f_{TB_*}}(s') := f_{TB_*}^{(i+1)}(s')$.

Gauss-Seidel Dynamic Optimisation

Illustration of function iteration according to GDO:



Asynchronous Dynamic Optimisation

Definition of function iteration according to ADO:

- The values of function $\boldsymbol{f}^{(i)}_{\boldsymbol{TB}_*}(\boldsymbol{s})$ will be updated only for a subset of states in the next iteration $\boldsymbol{i+1}$.
- This subset typically will change in the progression of iterations, i.e. depending on the iteration counter.
- As a special case the subset may only consist of one state.

Asynchronous Dynamic Optimisation

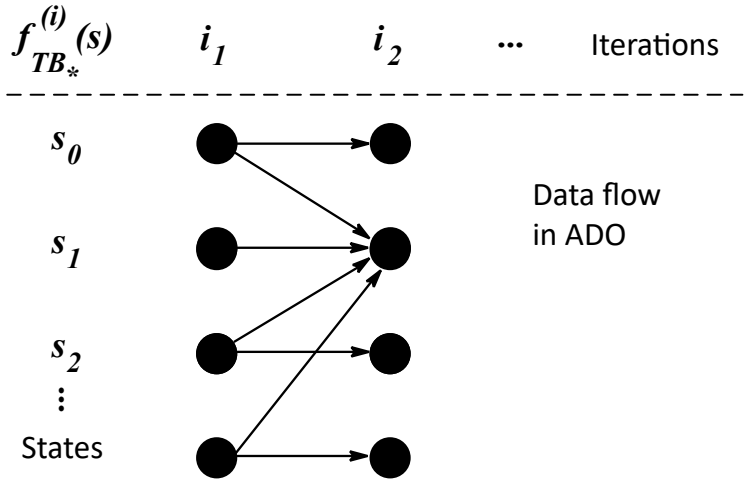
Let $\mathcal{S}^{(i+1)} \subset \mathcal{S}$ be a subset of states to be considered in iteration $i+1$.

If $s \in \mathcal{S}^{(i+1)}$, then $f_{TB_*}^{(i+1)}(s) := max_{a \in \mathcal{A}}\{f_{EB}(s,a) + \gamma \cdot f_{TB_*}^{(i)}(s')\}$, else $f_{TB_*}^{(i+1)}(s) := f_{TB_*}^{(i)}(s)$.

- Functions $f_{TB_*}^{(i+1)}(s)$ and $f_{TB_*}^{(i)}(s)$ differ only for states belonging to $\mathcal{S}^{(i+1)}$.
- In the following example, $\mathcal{S}^{(i_2)}$ consists only of one state, namely e.g. s_1.

Asynchronous Dynamic Optimisation

Illustration of function iteration according to ADO:



Online Learning by ADO

- Online Learning can be realised through ADO.
- The set $\mathcal{S}^{(i+1)}$ only consists of one state, i.e. the current state (position and orientation) of the robot.
- The total benefit function will updated iteratively only for the current state by including the current elementary benefit.
- This approach for Online Learning (originally published as „Q-Learning“ by Watkins 1992) is just one type of so-called „Reinforcement Learning“.
- A wide range of methods of Reinforcement Learning is available (see Sutton et al. 2020).

8.4 Reinforcement Learning (Q-Learning)

Overview:

- Introduction to Q-Learning
- Implementation of Q-Learning
- Explanatory remarks to the algorithm

Introduction to Q-Learning

- Watkins published the seminal article on „Q-Learning“ in the early nineties.
- Further approaches of Online Learning have been published by other researchers (e.g. Sutton, Barto) in the nineties.
- Finally, in the late nineties, Sutton and Barto wrote the first textbook on Reinforcement Learning, which contains a systematic description of a wide range of Online Learning techniques, and also introduces an unification of technical terms.

Introduction to Q-Learning

- Symbol $\boldsymbol{Q}$ stands for „quality“, and is equivalent with the total benefit. Another synonym is „return“.
- Total benefit function $\boldsymbol{f_{TB}}$ is represented by the so-called „$\boldsymbol{Q}$-table“, with one axis for a finite set $\boldsymbol{S}$ of possible states, and the other axis for a finite set $\boldsymbol{A}$ of possible actions.
- Each component in the table is a real number approximating the total benefit, if in the certain state the certain action is executed and the subsequent robot actions are determined by the current version of the decision function $\boldsymbol{f_D}$.

Introduction to Q-Learning

$Q^{(i)}(s,a)$	a_1	$\dots$	a_m
s_1			
$\vdots$			
s_n			

The table component $Q^{(i)}(s_k, a_l)$ is an estimation of the total benefit at iteration i, given that the robot is in state s_k and executes action a_l.

Introduction to Q-Learning

- The robot is moving and perceiving, i.e. is experiencing the environment. Each state transition leads to an elementary benefit. A synonym is „reward“.
- The elementary benefit is used to refresh the $\boldsymbol{Q}$-table, i.e. obtain an improved approximation of the total benefit function.
- Certain state-action pairs will be reinforced, while others will be weakened. The incessant refreshment of the $\boldsymbol{Q}$-table should end up in a convergence.
- Finally, the converged $\boldsymbol{Q}$-table includes the approximation of the total benefit function $\boldsymbol{f_{TB_*}}$, and enables the robot to use the optimal decision function $\boldsymbol{f_{D_*}}$.

Implementation of Q-Learning

1. Initialisation

   (a) Initialise $i := 1$ to set up the iteration index for the Q-table

   (b) Initialise $t := 0$ to set up the time index

   (c) Initialise $Q^{(i)}(s, a) := 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, i.e. assuming path planning knowledge is not available

   (d) Initialise Δt to set up a uniform time interval needed for executing an action

   (e) Observe environmental state s_t

Implementation of Q-Learning

2. Repetition until fulfilled stopping criterion:

   (a) Take current, environmental state $\boldsymbol{s_t}$

   (b) Determine optimal action $\boldsymbol{a_t := a_*}$, according to
   $$\boldsymbol{Q^{(i)}(s_t, a_*) = max_{a \in \mathcal{A}}\{Q^{(i)}(s_t, a)\}}$$

   (c) Execute action $\boldsymbol{a_t}$ in the environment, which takes time interval $\boldsymbol{\Delta t}$

   (d) Obtain the elementary benefit $\boldsymbol{r_t := f_{EB}(s_t, a_t)}$

   (e) Observe new, environmental state $\boldsymbol{s_{t'} := s_{t+\Delta t}}$

Implementation of Q-Learning

2. cont.

(f) With probability $(1-\epsilon)$ compute:

$$Q^h(s_t, a_t) := r_t + \gamma \cdot max_{a \in \mathcal{A}}\{Q^{(i)}(s_{t'}, a)\}$$

Else with probability ϵ compute:

$$Q^h(s_t, a_t) := r_t + \gamma \cdot rand_{a \in \mathcal{A}}\{Q^{(i)}(s_{t'}, a)\}$$

(g) Refreshment

$$Q^{(i+1)}(s_t, a_t) := Q^{(i)}(s_t, a_t) + \xi \cdot (Q^h(s_t, a_t) - Q^{(i)}(s_t, a_t))$$

(h) Change $i := i + 1$, $t := t + \Delta t$

Explanatory remarks to the algorithm

Parameters in the algorithm:

- Term *rand* implies the occasional selection of a „random“ action, as opposed to the usual selection of the most promising action, in order to escape from local maxima in the current estimation of the total benefit function. Term „occasional“ is synonymous to „probabilistic“ which is formalised by a randomness parameter ϵ.
- Parameter $\xi \in \{0, 1\}$ defines a so-called „learning rate“ and controls the speed of learning. E.g. in extreme settings, for $\xi = 0$ there is no learning at all, but for $\xi = 1$ the refreshment rate is highest.

Explanatory remarks to the algorithm

Stratified stopping criteria of the algorithm:

- Life-long learning to enable adaptation of robot behavior to any unexpected situational change.
- Learning as long as the components in $\boldsymbol{Q}$-table are changing significantly, i.e. for any state the most promising action is not yet fixed. In other words, learning until convergence of the decision function.
- Learning until robot reaches a certain target position.
  The underlying sequence of iterations may be called „episode of iterations“. Typically, many episodes of iterations are needed to obtain the optimal decision function.

8.5 Exemplary Q-Learning

Overview:

- Results of Q-Learning of a simulated robot

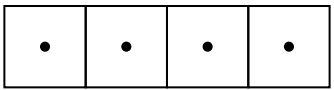
Results of Q-Learning of a simulated robot

Description of the simulated robot:

- Mobile robot
- $\mathbf{16}$ different states
- State characterised by position and orientation
- $\mathbf{4}$ possible positions on horizontal axis
- $\mathbf{4}$ possible orientations $\mathbf{0°, 90°, 180°, 270°}$
- States visualised by 4×4 matrix of numbered fields

Results of Q-Learning of a simulated robot

4 possible robot positions:



and 4 possible robot orientations:

$\Rightarrow$ There are $\mathbf{4 \cdot 4 = 16}$ possible robot states.

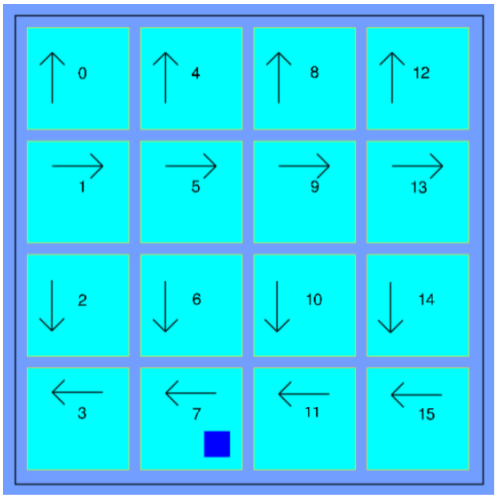
Results of Q-Learning of a simulated robot

Description of the simulated robot:

- Discrete rotation by $+90°$ or $-90°$
- Discrete translation according to robot orientation
- Instead of elementary benefits, use elementary costs:
  $f_{EC}(s, a) := 1$, if new state is not the target state
  $f_{EC}(s, a) := 0$, if new state is the target state
- Apply Q-Learning, but minimise the total costs
- E.g. find shortest sequence of actions between states 7 and 11

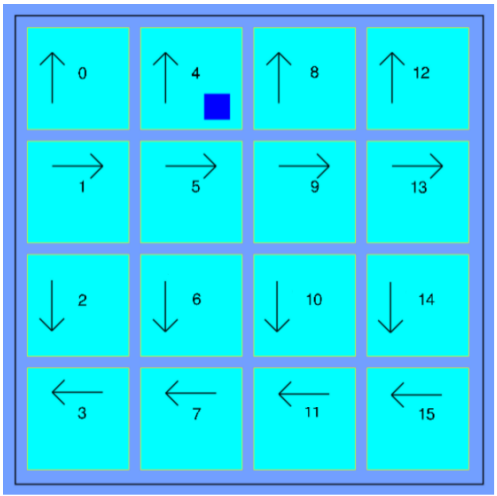
Results of Q-Learning of a simulated robot

State transition of the robot:



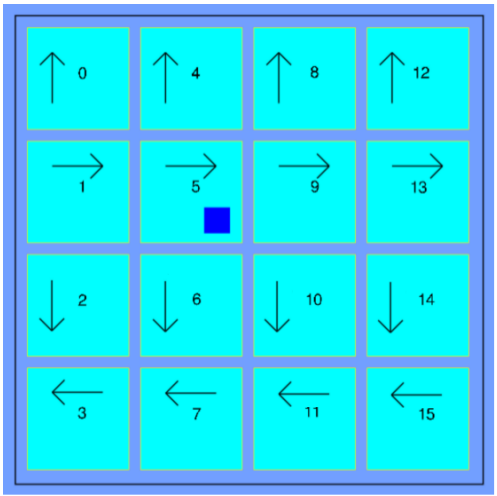
Results of Q-Learning of a simulated robot

State transition of the robot:



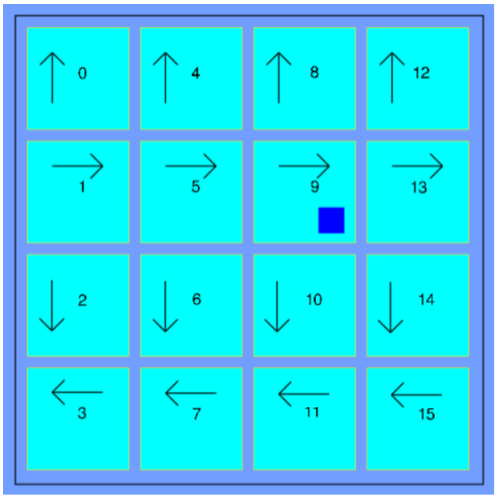
Results of Q-Learning of a simulated robot

State transition of the robot:



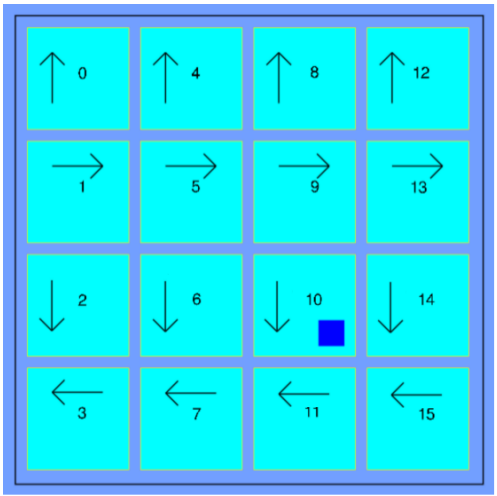
Results of Q-Learning of a simulated robot

State transition of the robot:



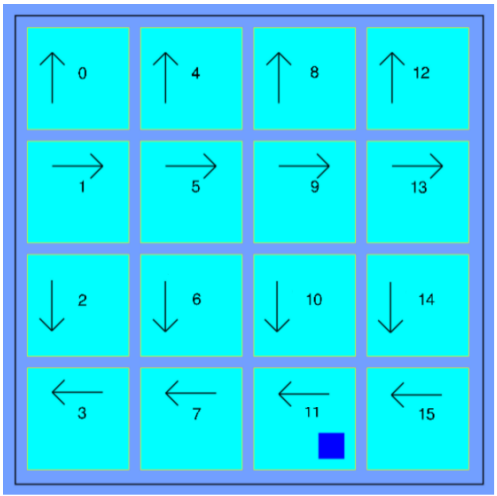
Results of Q-Learning of a simulated robot

State transition of the robot:



Results of Q-Learning of a simulated robot

State transition of the robot:



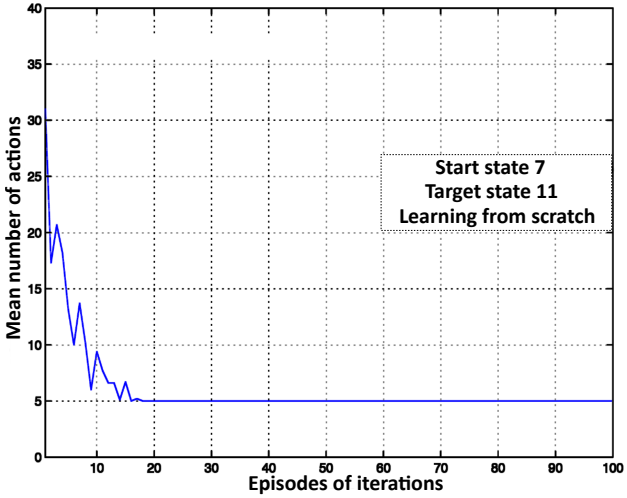
Results of Q-Learning of a simulated robot

Experimental result 1:

$\mathbf{7 \Rightarrow 11}$

Optimal sequence: $\mathbf{7, 4, 5, 9, 10, 11}$

$\gamma := \mathbf{0.8}$
$\xi := \mathbf{0.3}$



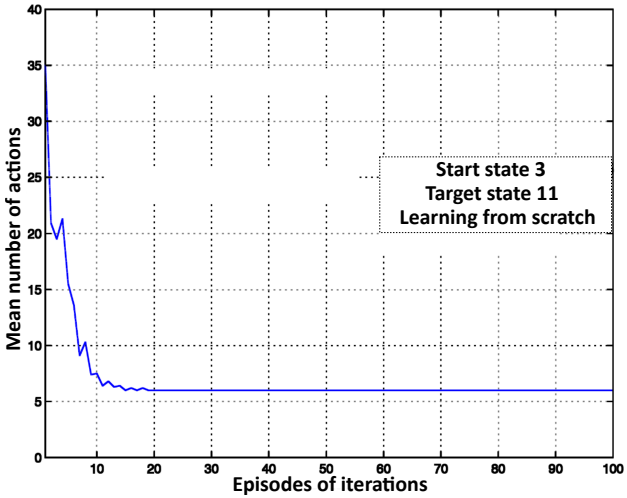
Results of Q-Learning of a simulated robot

Experimental result 2:

$\mathbf{3 \Rightarrow 11}$

Optimal sequence: $\mathbf{3, 0, 1, 5, 9, 10, 11}$

$\gamma := \mathbf{0.8}$
$\xi := \mathbf{0.3}$



Results of Q-Learning of a simulated robot

Experimental result 3:

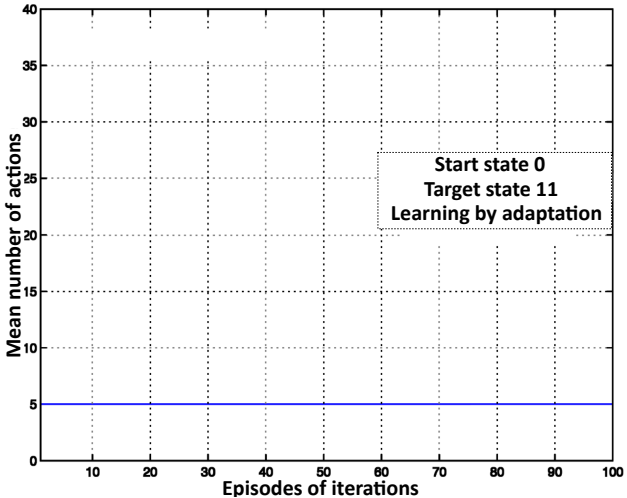
$0 \Rightarrow 11$

Optimal sequence: $0, 1, 5, 9, 10, 11$

Use learned behavior $(3 \Rightarrow 11)$

$\gamma := 0.8$

$\xi := 0.3$



Results of Q-Learning of a simulated robot

Experimental result 4:

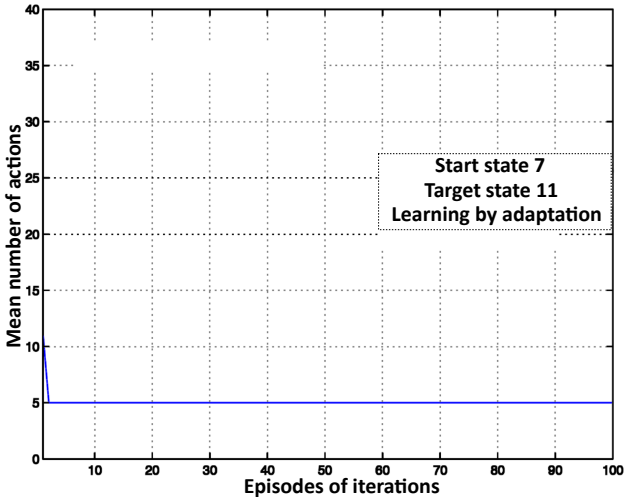
$7 \Rightarrow 11$

Optimal sequence: $7, 4, 5, 9, 10, 11$

Use learned behavior $(3 \Rightarrow 11)$

$\gamma := 0.8$

$\xi := 0.3$



8.6 Recent research trends in Reinforcement Learning

Overview:

- Challenges in Reinforcement Learning
- Sophisticated elementary benefit function
- Combination of onboard and offboard learning

Challenges in Reinforcement Learning

Uncertain states:

- It is assumed that the states can be observed with certainty, which is not realistic.
  In the real application, probabilistic distribution functions concerning the current state must be considered.

Challenges in Reinforcement Learning

Analog states:

- It is assumed, that the new state is included in the finite set of possible states, which is not realistic.
  In real application, the Q-table and its adaptation mechanism should be replaced by a representation which enables analog states.
  As an example, so-called „Radial Basis Function Neural Networks“ together with its Online Learning technique seem to be appropriate (see Papierok et al. 2008).

Challenges in Reinforcement Learning

Learning from scratch:

- The Q-table can be initialised with zeros, i.e. no knowledge available. Nevertheless, the robot can learn optimal behaviors in unknown environments.
  However, learning from scratch is time-consuming, i.e. many iterations are needed.

Challenges in Reinforcement Learning

Learning not from scratch:

- An interesting question of research is, how to exploit planning knowledge for a sophisticated initialisation of the $\boldsymbol{Q}$-table.
- The reuse and adaptation of learned robot behaviors from one situational context into a similar context is advantageous, and is known under the term „Transfer Learning“. E.g. robot navigation under displaced target or obstacle objects.

Challenges in Reinforcement Learning

Sophisticated elementary benefit function:

- A sophisticated elementary benefit function will reduce the number of iterations until convergence, i.e. leads to efficient robot learning.

Sophisticated elementary benefit function

Example of a sophisticated elementary benefit function, expecting a fast convergence of the Q-table, i.e. fast online learning.

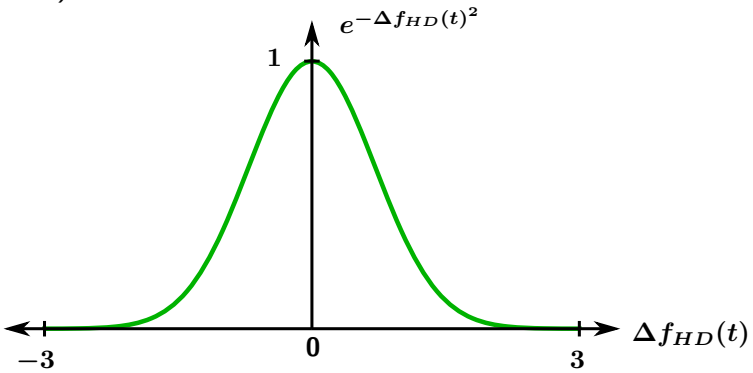
$$f_{EB}(s_t, a_t) := f_{CD}(s_t, s_{t+1}) \cdot f_{OD}(s_{t+1}) \cdot e^{-(f_{HD}(s_{t+1}) - f_{HD}(s_t))^2}$$

with f_{CD} the covered distance, f_{OD} the robots distance to nearest obstacle, f_{HD} the robot heading (orientation).

A large covered distance, and a large distance to obstacles, and constant robot orientation are value-added properties in the definition of elementary benefit function.

Sophisticated elementary benefit function

Visualisation of exponential part of elementary benefit function (see previous slide).



$$\Delta f_{HD}(t) := f_{HD}(s_{t+1}) - f_{HD}(s_t)$$

Combination of onboard and offboard learning

Further ideas to enable effective robot learning:

- A realistic simulation of the robot effector and of the robot sensor enables machine learning on a separate, high-performance, parallel-processing computer, and would be based on probabilistic movement and perception models.
- The mentioned offboard learning should be mixed with onboard learning, synchronously or asynchronously, relying on a wireless robot connection.

Combination of onboard and offboard learning

Learned collision avoiding robot navigation through a corridor

Literature

- A. Barto, S. Bradtke, S. Singh: Learning to Act using Real-time Dynamic Programming; Artificial Intelligence, 72:81-138, 1995.
- S. Papierok, A. Noglik, J. Pauli: Application of Reinforcement Learning in a Real Environment using RBF Networks; Proceedings of the International Workshop on Evolutionary Learning for Autonomous Robot Systems, pp. 17-22, 2008.
- R. Sutton, A. Barto: Reinforcement Learning - An Introduction, MIT Press, 2020. `http://incompleteideas.net/book/RLbook2020.pdf`
- C. Watkins: Q-Learning; Machine Learning Journal, 8:279-292, 1992.

9. Robotics Simulator

Overview:

9.1 Introduction

Overview:

- Purpose of robotics simulators
- Overview to robotics simulators

Purpose of robotics simulators

- Purpose of a robotics simulator is to develop an application for a real robot without depending on the actual machine, i.e. in order to check feasibility, and save cost and time.
- Robots should be programmed off-line, leading to a simulation of robot behaviors. This reduces down-time, e.g. of a robot-based assembly line.
- It is desired/hoped, that these programs can be transferred onto the real robot without or with minor modifications.

Overview to robotics simulators

- See `https://en.wikipedia.org/wiki/Robotics_simulator`
- E.g. Gazebo, http://gazebosim.org, developed by Open Source Robotics Foundation (OSRF), https://www.openrobotics.org
- E.g. CoppeliaSim, formerly known as Virtual Robot Experimentation Platform (VREP), developed by Coppelia Robotics, `https://www.coppeliarobotics.com`.
  Download and install 'CoppeliaSim Edu'; e.g. in Windows 10:
  C:\Program Files\CoppeliaRobotics\CoppeliaSimEdu
- etc.

9.2 Main features of CoppeliaSim

Overview:

- Cross-Platform and Portable
- Customizable Simulator
- Regular API and Remote API
- Robotic Scenes, Building Blocks, Hierarchy
- Robotic Models, Model Browser, Interaction
- Forward/Inverse Kinematics
- ...

9.2 Main features of CoppeliaSim

Overview:

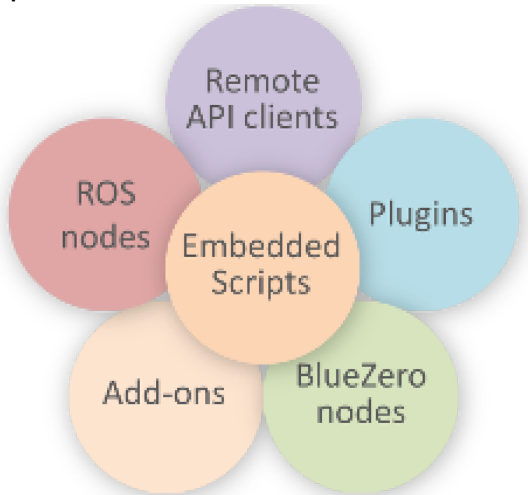
- ...
- Dynamics and Physics Engines
- Vision and Proximity Sensor Simulation
- Path/Motion Planning
- Data Import/Export
- Data Recording and Visualization

Cross-Platform and Portable

Creation of portable simulator (i.e. portable file), including a functional model, scene, control code.

Customizable Simulator

Six programming approaches to customize the simulator.



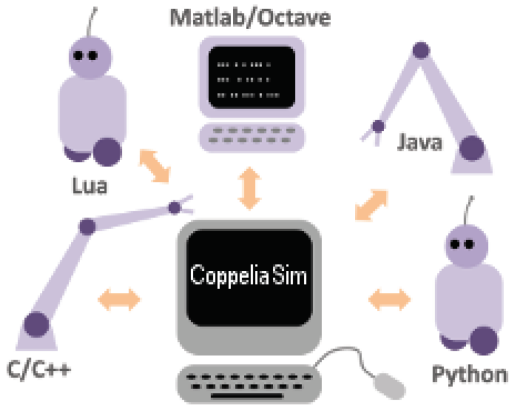
Regular API

Hundreds of functions (Lua and in C/C++) to implement a simulator.

- File operations (e.g. sim.saveScene)
- General functionality handling (e.g. sim.handleChildScripts)
- Collision detection (e.g. sim.checkCollision)
- Dynamics (e.g. sim.readForceSensor)
- Proximity sensors (e.g. sim.readProximitySensor)
- Cameras (e.g. sim.adjustView)
- Point clouds (e.g. sim.createPointCloud)
- Paths (e.g. sim.createPath)
- Generic dialogs (e.g. sim.displayDialog)
- Import/export (e.g. sim.importShape)
- ...

Remote API

Simulator can be extended with remote functions (e.g. running on a real robot or at another PC), possibly implemented in different languages (e.g. C, Python, Java, Matlab, Lua).



Robotic Scene composed of Building Blocks

Robot system and its environment is called a scene.

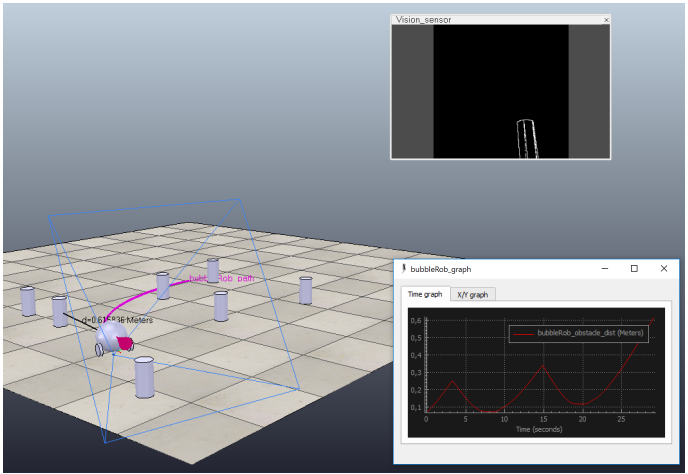
A scene is built from basic objects, called models, and includes various functionality.

A scene is customizable, and is represented in a *.ttt file.

See 'Help -> Help topics -> Tutorials -> BubbleRob tutorial'.

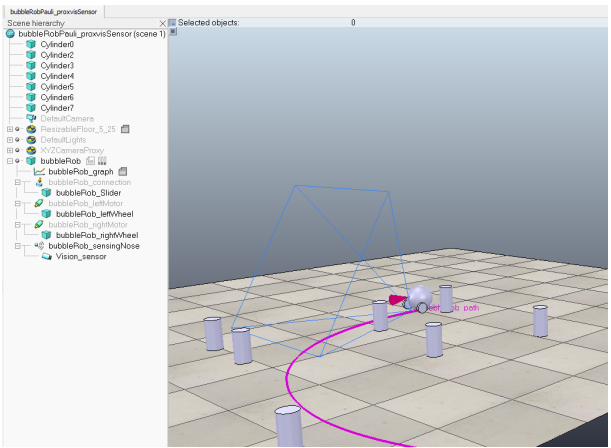
Robotic Scene composed of Building Blocks

E.g. bubbleRobPauli_proxvisSensor.ttt



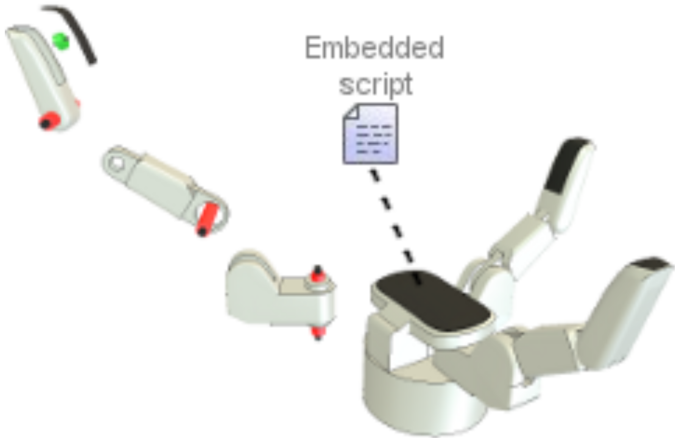
Robotic Scene organised in a Hierarchy

A scene is a nested organisation of models. A model is represented by its type, name, and associated embedded control script.



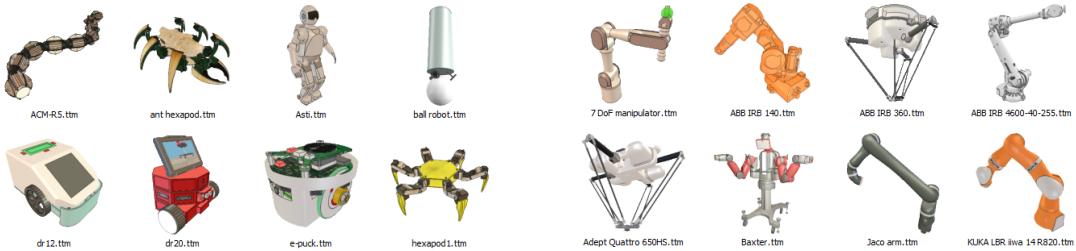
Robotic Scene organised in a Hierarchy

A model may include components and even sub-components.



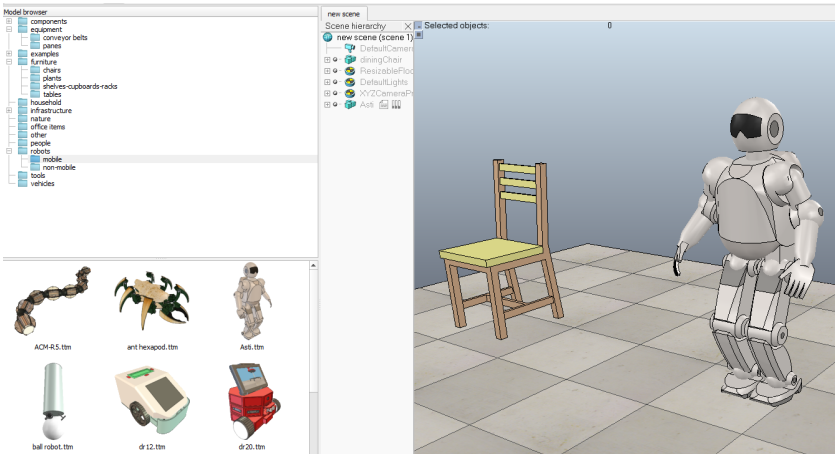
Models

Predefined and customizable models, represented in *.ttm files: Components (actuators, grippers, sensors, etc.), Equipment (e.g. conveyor belts), Furniture (chairs, tables, etc.), Infrastructrue (floors, doors), People, Robots (mobile, non-mobile), Vehicles.



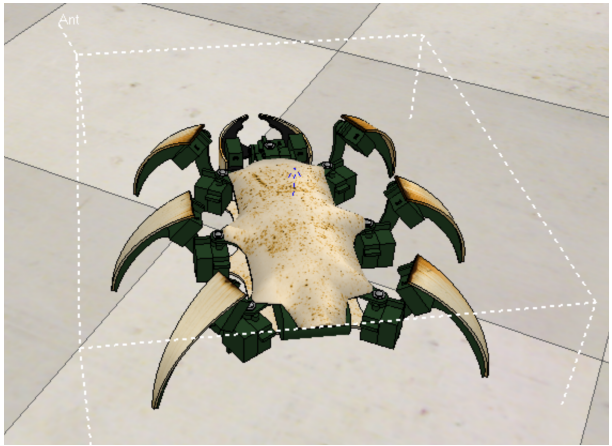
Model Browser

Drag-and-Drop operations for scene composition.



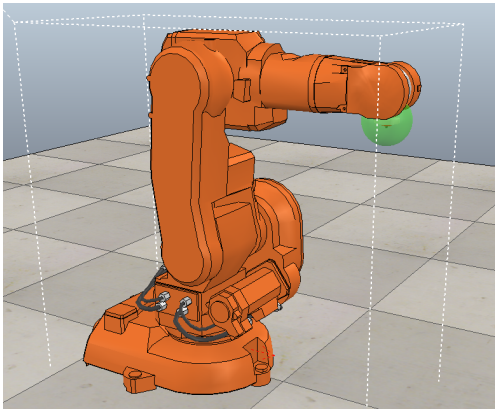
Interaction and Custom User Interfaces

Models can be translated, rotated, etc., and their behavioral parameters changed.



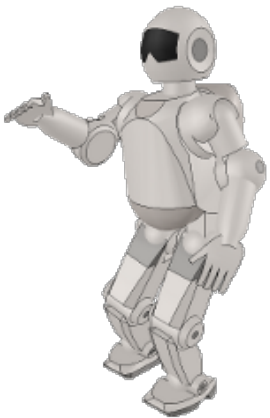
Forward/Inverse Kinematics

Forward/Inverse kinematics calculations for many robot types, e.g. articulation robot arm (kinematic chain) ABB-IRB-140.ttm



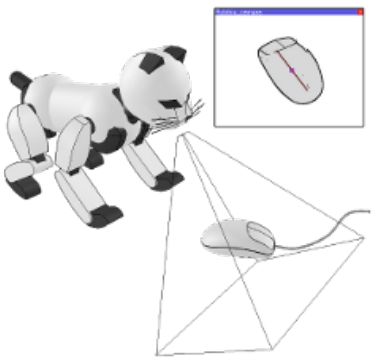
Dynamics and Physics Engines

Dynamics calculations to simulate real mechanics and object interactions, e.g. velocities, accelerations, forces.



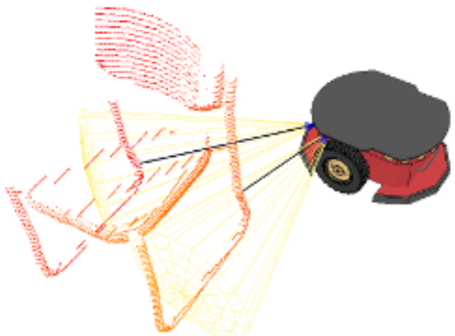
Vision Sensor Simulation

Simulation of vision sensors, with frustum field-of-view, and built-in image processing operators, customizable and extendable.
In bubbleRobPauli_proxvisSensor.ttt, see sequence of images with gray-value edges.



Proximity Sensor Simulation

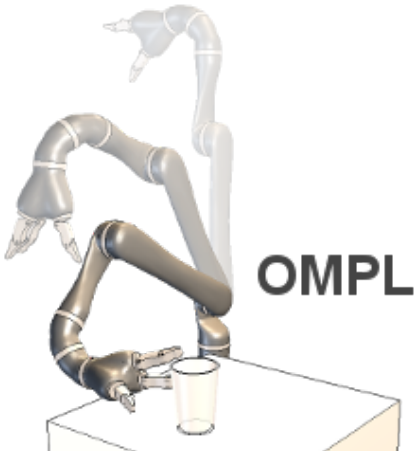
Minimum distance calculation, based on e.g. octrees or point clouds, assuming a certain type of view.



See also 3D-LaserScanner_TwoChairs.ttt

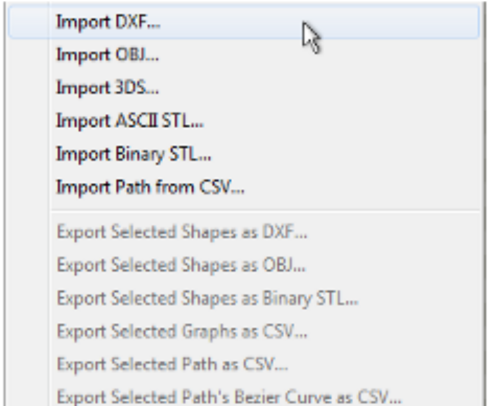
Path/Motion Planning

Motion planning is supported via the Open Motion Planning Library (OMPL) wrapped in a plugin. See `https://ompl.kavrakilab.org`



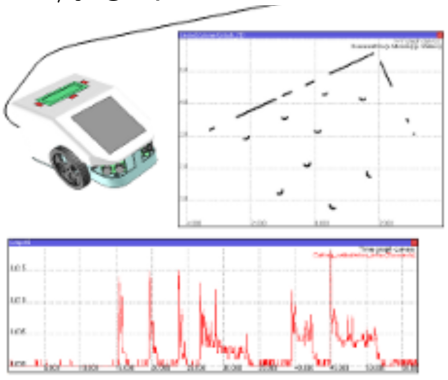
Data Import/Export

Examples for supported formats: Unified Robot Description Format (URDF), Drawing Interchange File Format (DXF), Three-Dimensional Geometric Shapes (OBJ), Comma-Separated Values (CSV), etc.



Data Recording and Visualization

Recordable data streams can be displayed as time-graphs, or combined with each other to form x/y-graphs.



See also bubbleRobPauli_proxvisSensor.ttt

9.3 Examples of CoppeliaSim Scenes and Models

Overview:

- CRS_2_CMP
- CRS_3_SEN
- CRS_4_VCR
- CRS_6_NAV
- CRS_9_SIM

Examples in CRS_2_CMP

- ABB-IRB-140.ttm
- ACM-R5.ttm
- hexapod.ttm
- MTB robot.ttm
- NAO.ttm
- OmnidirectionalWheels.ttt
- pioneer p3dx.ttm
- Stewart platform.ttm

Examples in CRS_3_SEN

- 2D_Camera_DiningChair.ttt
- 2D_LaserScanner_LargeBasket.ttt
- 3D-LaserScanner_TwoChairs.ttt
- bubbleRob.ttt
- bubbleRobPauli_proxSensor.ttt
- bubbleRobPauli_proxvisSensor.ttt
- Kinect_WalkingBill.ttt
- twoRobotArms_proxSensor.ttt

Examples in CRS_4_VCR

- ABB-IRB-140.ttm
- path_planning_and_grasping.ttt
- lineFollowingBubbleRob.ttt
- ThreeVisionBased_LineFollowing.ttt
- VisionProxSensor_LineFollowing_ObstacleAvoiding.ttt

Examples in CRS_6_NAV

- path_planning_and_grasping.ttt
- holonomic_path_planning_3dof.ttt
- twoRobotArms_proxSensor.ttt

Examples in CRS_9_SIM

- robotLanguageControl.ttt

9.4 Scripts in CoppeliaSim

Overview:

- Two-robot example
- Scripts and script languages
- CoppeliaSim scripts in Lua
- Embedded simulation scripts
- Main simulation loop
- Main script
- Child scripts

Two-robot example

- Take scene robotLanguageControl.ttt
- Each robot can grasp and carry an object from and to a conveyor belt or interstation.
- Each robot can generate and react to bit-signals, depending on its own actions or on the action of the other robot.
- Devices external to those robots (gripper, suction pad or conveyor belt sensors) can read or write those bit-signals too.
- Implementation is based on 'main script', and 'child scripts' for robots, conveyor belts, and sensors, respectively.

Scripts and script languages

- Scripts are small programs for solving non-complex, modular tasks.
- The programs are written in a 'script language', which is a simple programming language.
- Script languages can be learned easily due to the lack of complex language constructs, e.g. lack of data types for variables.
- Primitive statements are usually API calls.
- Examples of script languages are e.g. Lua, Python, Javascript.
- See LUA-CrashCourse.pdf

CoppeliaSim scripts in Lua

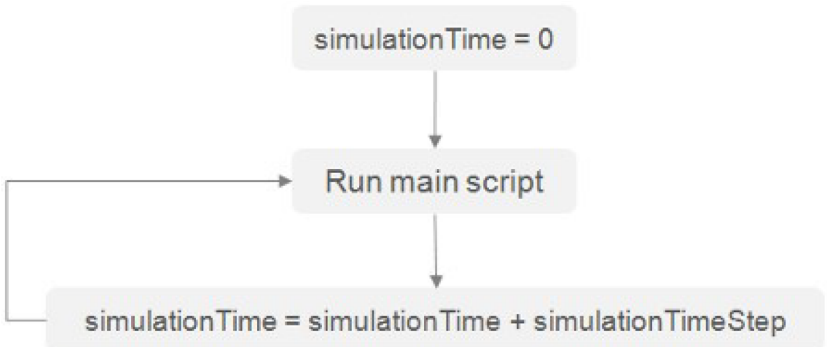
- CoppeliaSim is a highly customizable simulator, i.e. almost every step of a simulation is user-defined.
- This flexibility is allowed through an integrated script interpreter. The scripting language is Lua.
- CoppeliaSim specific Lua statements (API calls) can be recognized by their sim-prefixes (e.g. sim.CheckCollision).

Embedded simulation scripts

- An 'embedded script' is a script that is embedded in a scene (or model), i.e. a script that is part of the scene and that will be saved and loaded together with the rest of the scene (or model).
- A major type of embedded scripts are the 'simulation scripts', which are executed during simulation.
- Each scene is based on THE simulation 'main script', which is responsible for the main simulation loop.
- Models are controlled via simulation 'child scripts'.

Main simulation loop

- In the main simulation loop the main script is executed, repeatedly.
- Simulator operates by advancing the simulation time at constant time steps.



Main script

Main script is composed of three parts, see 'defaultMainScript.lua'.

- 'Initialization part': executed just once at simulation start, to prepare the simulation.
- 'Regular part': executed repeatedly at each simulation pass, to handle all the functionality of the simulator, and is organised in an 'Actuation part' and a 'Sensing part'.
- 'Restoration part': executed once at simulation end or at simulation pause, to save or restore a certain configuration and/or continue with simulation, and is organised in 'Cleanup part', 'Suspend part', 'Resume part'.

Main script

```
-- Initialization part --

function sysCall_init()
  sim.handleSimulationStart()
  ...
end
```

Main script

```
-- Regular part --

function sysCall_actuation()
  -- Actuation part --
  sim.handleChildScripts(sim.syscb_actuation)
  ...
end

function sysCall_sensing()
  -- Sensing part --
  sim.handleChildScripts(sim.syscb_sensing)
  ...
end
```

Main script

```
-- Restoration part --

function sysCall_cleanup()
  sim.handleChildScripts(sim.syscb_cleanup)
  ...
end

function sysCall_suspend()
  sim.handleChildScripts(sim.syscb_suspend)
  ...
end

function sysCall_resume()
  sim.handleChildScripts(sim.syscb_resume)
  ...
end
```

Child scripts

- According to the configuration of a main script, a child script also has Initialization part, Restoration part, as well as Actuation part and/or Sensing part.
- When called by the main script, the child scripts perform some task and then return to main script.
- These child scripts are so-called pass-through child scripts (also synonymous to non-threaded child scripts), i.e. operate as functions. Alternatively, also threaded child scripts can be implemented (see later).
- Sensing part and Actuation part realize a perception-action cycle in the simulation loop.

Child scripts

- One statement of the main script is of particular interest, i.e. the call of the regular API function sim.handleChildScripts, which in turn calls a function of a child script.
- E.g. sim.handleChildScripts(sim.syscb_actuation) calls the function sysCall_actuation of a child script.
- The function sysCall_actuation in main script differs from the function sysCall_actuation in the child script, in that the former includes the same general actuation-relevant code for all simulations, and the latter includes actuation-relevant code provided by the user to customize the simulator.
- Same is true for sysCall_init, sysCall_sensing and sysCall_cleanup.

Child scripts

```
-- Default child script --

function sysCall_init()
    -- do some initialization here
end

function sysCall_actuation()
    -- put your actuation code here
end

function sysCall_sensing()
    -- put your sensing code here
end

function sysCall_cleanup()
    -- do some clean-up here
end
```

Child scripts

- An unlimited number of child scripts is possible for each scene.
- Each child script is embedded in a model of the scene, and allows handling of a particular function in a simulation.
  E.g. see child scripts of boxSensor and suctionPad (of MTB_Robot).
- The script dialog, located at 'Menu bar -> Tools -> Scripts', shows the list of child scripts together with associated models (synonymous to 'objects') for the scene.

Child scripts

- Good portability: Script code is independent from Operating System version.
- Inherent scalability: If a model is duplicated, then also its child script will be duplicated, but installing new identifier.
- Script customization: For each duplicate of a model the inherent script can be customized individually.
- Easy synchronization: Synchronization of main script and child script in the main simulation loop.

Child scripts

- By launching a threaded child script, a thread will be created, started, or resumed.
- Threads are 'light-weight processes', i.e. the switch from one thread to another thread is time-efficient.
- Once a thread was suspended, it will resume execution at next simulation pass.
- 'Multi-threading' is useful to minimize the amount of computer resources, and handles (unexpected) robotics situations efficiently.

9.5 Plugins in CoppeliaSim

Overview:

- Plugins
- Usages of plugins
- Example for using a plugin

Plugins

- A 'plugin' is a shared library (e.g. a Dynamic Link Library (DLL)), that is automatically loaded by CoppeliaSim at program start-up, or dynamically loaded.
- It allows the functionality to be extended by user-written functions.
- The language can be any language able to generate a shared library (e.g. C/C++).

Usages of plugins

- A plugin can be used as a wrapper for running code written in other languages.
- A plugin can be used to provide a special functionality requiring either fast calculation capability (scripts are usually slower than compiled languages) or an interface to a hardware device (e.g. a real robot).
- ...

Example for using a plugin

- In scene robotLanguageControl.ttt, two robots are controlled by a specific robot language.
- The robots' respective programs are located in child scripts of MTB_Robot and MTB_Robot#0, respectively in the sections of lines beginning with 'program=[['.
- CoppeliaSim doesn't understand the native robot programs, but the 'simExtMTB.dll' plugin enables the translation.
- The application program 'mtbServer.exe' is the robot language interpreter. It is launched by the plugin for each robot in the scene, and compiles and executes the respective robot program.

Example for using a plugin

Excerpt of child script of MTB_Robot:

```
function sysCall_init()
  ...
  serverHandle=startRobotServer(
    robotHandle,program,{0,0,0,0},{0.1,0.4})
  ...
end

program=[[... PROGRAM_BEGIN_LABEL ...]]

startRobotServer = function(
  theRobotHandle,theProgram,initJoints,initVel)
  ...
end
...
```

Example for using a plugin

Excerpt of the robot program:

```
PROGRAM_BEGIN_LABEL
CLEARBIT 1
SETROTVEL 45
MOVE 0 0 0 0
IFBITGOTO 1 PICKUP_LOCATION2
IFBITGOTO 2 PICKUP_LOCATION1
GOTO PROGRAM_BEGIN_LABEL
PICKUP_LOCATION1
REM a box is on the podest (was signaled by robot 2):
MOVE -160 -43 0 -117
MOVE -160 -43 0.1815 -117
REM activate the suction pad:
SETBIT 1
WAIT 500
...
```

Example for using a plugin

Semantics of robot program statements:

- REM: starts a comment line
- SETLINVEL v: sets the prismatic joint velocity for next movements (v is in m/s)
- SETROTVEL v: sets the revolute joint velocity for next movements (v is in degrees/s)
- MOVE p1 p2 p3 p4: moves to joint positions (p1;p2;p3;p4) (in degrees except for p3 in meters)
- WAIT x: waits x milliseconds
- SETBIT y: sets the bit at position y (1-32) in the robot output port

Example for using a plugin

Semantics of robot program statements, cont.:

- CLEARBIT y: clears the bit at position y (1-32) in the robot output port
- IFBITGOTO y label: if bit at position y (1-32) in the robot input port is set, jump to 'label'
- IFNBITGOTO y label: if bit at position y (1-32) in the robot input port is not set, jump to 'label'
- GOTO label: jumps to 'label'

9.6 ROS interface to CoppeliaSim

- Several Interfaces between ROS and CoppeliaSim are available: ROS Interface, ROS2 Interface, CoppeliaSim ROS bridge, etc.
- See also: https://www.coppeliarobotics.com/helpFiles/en/rosInterfaces.htm
- CoppeliaSim may act as a ROS node, and other ROS nodes can communicate with it via services, publishers, subscribers, during a ROS application in the real environment.
- CoppeliaSim may serve as a platform for testing parts of a ROS application in advance in a simulated environment, before being executed in the real environment.

Literature

- Overview to robotics simulators:
  `https://en.wikipedia.org/wiki/Robotics_simulator`
- Virtual Robot Experimentation Platform (CoppeliaSim):
  `https://www.coppeliarobotics.com`
- Robot Operating System:
  `https://de.wikipedia.org/wiki/Robot_Operating_System`

10. Robot Operating System

Overview:

10.1 Challenges encountered in robotics

- Asynchronous world
- Robots must manage significant complexity
- Enable robotic Software reuse

Asynchronous world

Example of a sequential program:

```
...
goForward(1);
turnLeft(Math.PI/2);
Image image = camera.getImage();
double distance =
  computeDistanceToObject(image);
goForward(distance);
(x,y) =
  getMyPositionFromEncoderCounts();
...
```

Asynchronous world

Problems of Sequential Programming:

- What happens if an obstacle appears while robot is moving?
- How to make use of the encoder data while robot is moving?

Asynchronous world

Solution with Callbacks:

- Function is called whenever data is available for processing.
- An asynchronous callback can happen at any time.

Example: Run the relevant callback function whenever

- an image is read from the camera,
- the odometry sensor reports new data,
- etc.

Asynchronous world

Example program with Callbacks:

```
...
void imageCallback(
                ImageMessage image)
  // process the latest image

void odometryCallback(
                OdometryMessage data)
  // handle latest odometry data

...
```

Asynchronous world

Example program with Callbacks, cont.:

```
...
void main()
  initialize();
  subscribe("image_msgs",
            imageCallback);
  subscribe("odometry_msgs",
            odometryCallback);
...
```

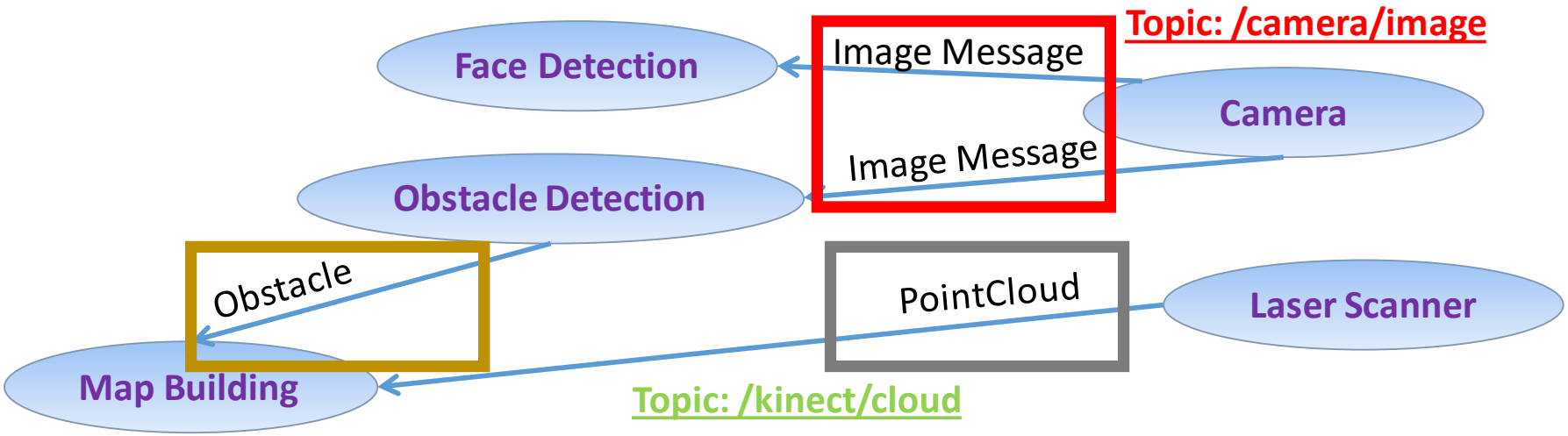
Managing complexity

Organising code, define processes, use interfaces:

- Cameras, Odometry, Laser Scanner, Map Building, etc., can all be separated out.
- The underlying Software processes are called „nodes“.
- The nodes interact or communicate through interfaces, which are called „topics“.
- Each process only receives data (messages) it requests.
- This is achieved by a „Publish/Subscribe“ mechanism.

Managing complexity

Example for managing complexity:

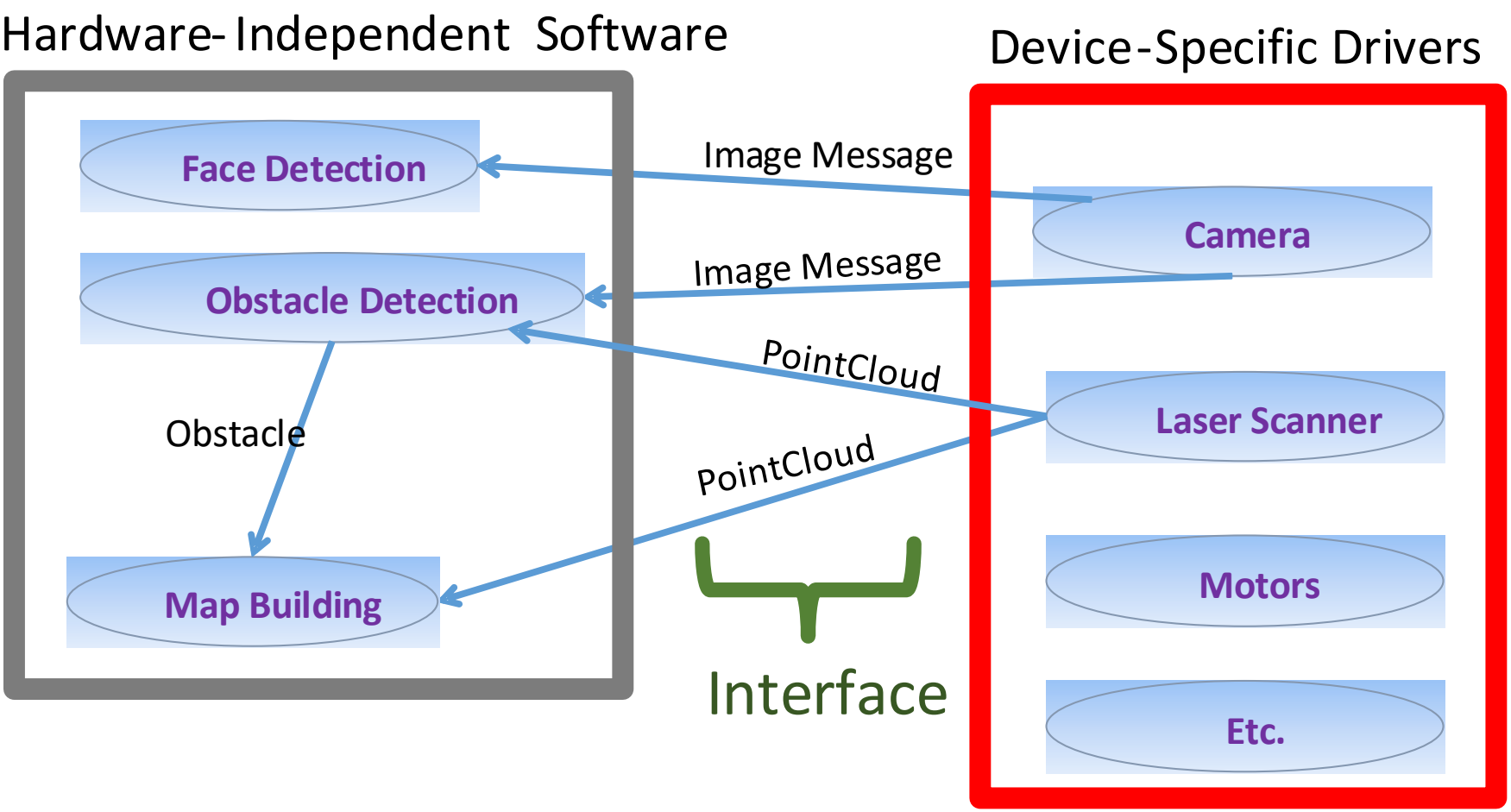


Enable Software reuse

- As many Software components as possible must not depend on the hardware.
- Robot hardware requires abstraction.

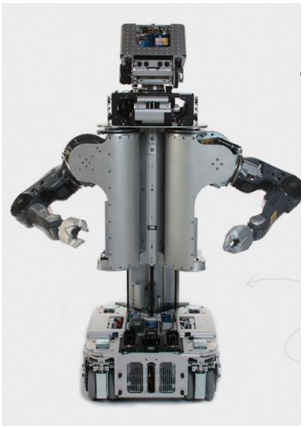
Enable Software reuse

Principle of hardware abstraction:



Enable Software reuse

Using the same Software for diverse robots:



PR2

Roomba

Care-O-bot 3

10.2 Overview to ROS

- History of ROS
- OSRF quotation concerning ROS
- Build and Communication Platform
- Layered architecture of a standard PC system
- Standard PC system vs. ROS-based robotic system
- Meta Operating System
- Application Building Blocks
- Operating System (system software)
- Levels of development

History of ROS

- Initiated in 2007 at Stanford Artificial Intellig. Lab.
- Developed in 2008 – 2013 by „Willow Garage“, a past robotics research lab at Menlo Park, California.
- Transition to „Open Robotics“ in 2017 (formerly „Open Source Robotics Foundation“, OSRF).
- ROS 1 in the past and now: recent distributions Noetic Ninjemys, Melodic Morenia, Lunar Loggerhead, Kinetic Kame, Jade Turtle, ...
- ROS 2 now and in future: supports real-time systems, current distribution Galactic Geochelone.
- URLs:
  - `https://www.ros.org`
  - `https://en.wikipedia.org/wiki/Open_Robotics`

OSRF quotation concerning ROS

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.
Why? Because creating truly robust, general-purpose robot software is hard. From the robot's perspective, problems that seem trivial to humans often vary wildly between instances of tasks and environments. Dealing with these variations is so hard that no single individual, laboratory, or institution can hope to do it on their own.
...

OSRF quotation concerning ROS

...
As a result, ROS was built from the ground up to encourage collaborative robotics software development. For example, one laboratory might have experts in mapping indoor environments, and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was designed specifically for groups like these to collaborate and build upon each other's work.

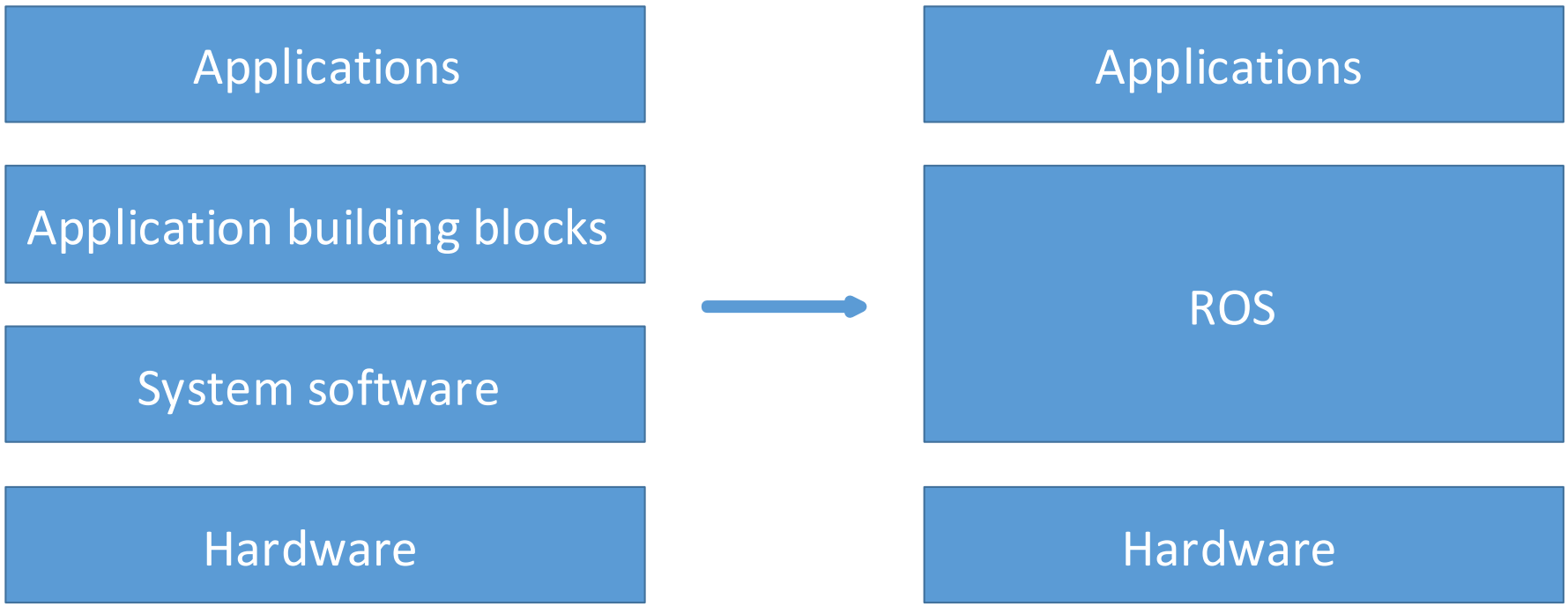
Build and Communication Platform

- ROS is a Build Platform
  - Configuration of an application system
  - Distribution and management of code
- ROS is a Communication Platform
  - ROS network graph
  - Connecting and running nodes

Layered architecture of a standard PC system

- Standardised layers
- System software abstracts hardware
- Applications leverage other applications
- Widely existent sets of libraries

Standard PC system versus ROS-based robotic system



Meta Operating System

- ROS runs primarily in Linux (Ubuntu), and is secondarily supported on Windows, MacOS, Android.
- ROS is also called a Meta Operating System.
- It combines Operating System (system software, core) and Application Building Blocks (libraries, capabilities), in order to create robotic applications.
- ROS is for free and is Open Source.

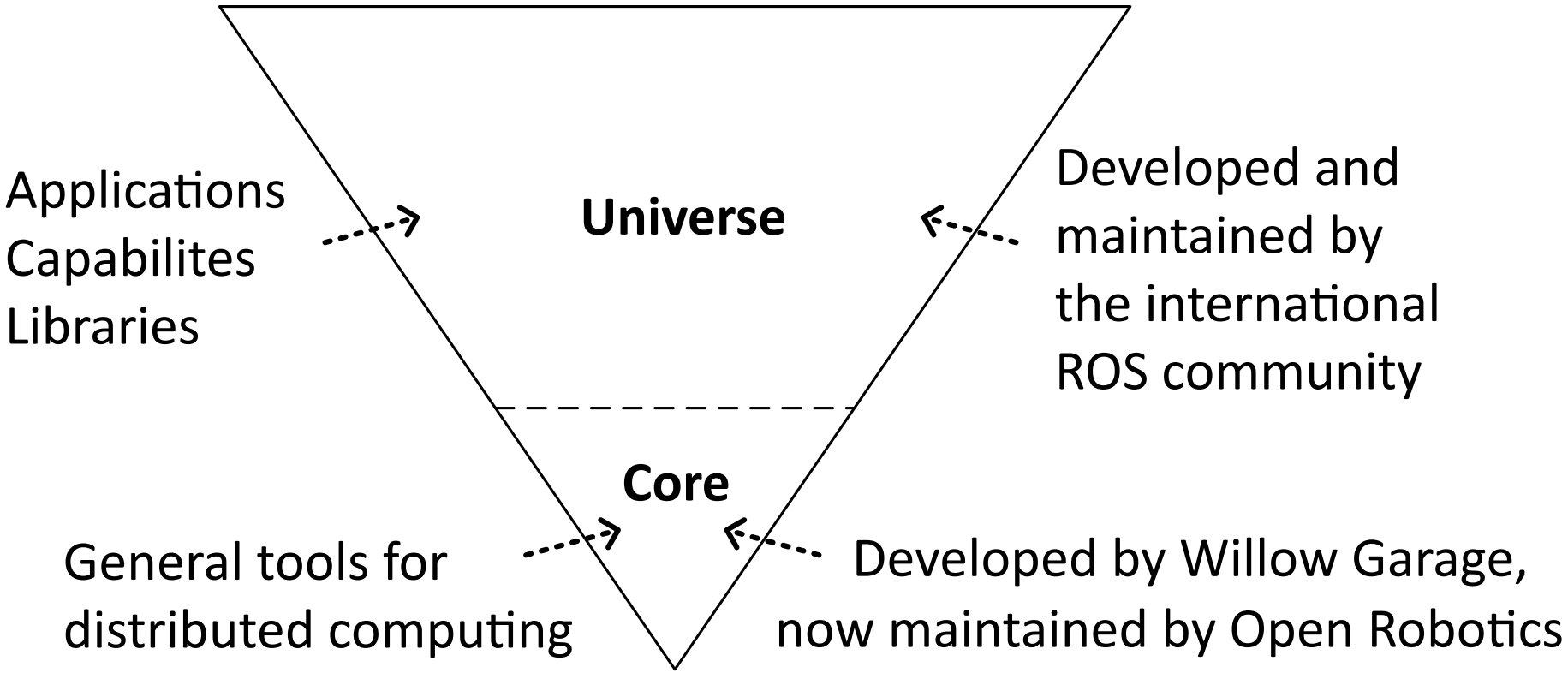
Application Building Blocks

- Coordinate system transforms
- Wheeled vehicle navigation
- Arm path planning
- Object recognition
- Visualization tools
- Debugging tools
- ...

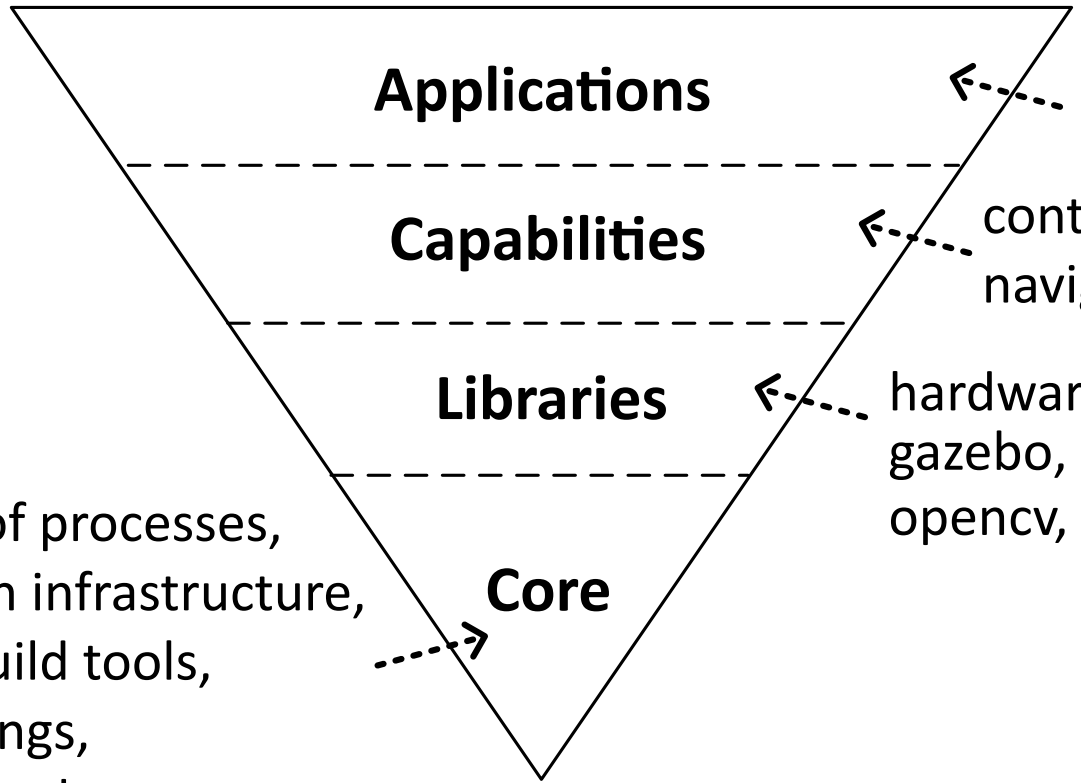
Operating System (system software)

- Manage hardware resources.
- Initialise, start, run, block, exit processes.
- Enable communication between processes.
- ...

Levels of development



Levels of development



fetching beer,
office cleaning, ...

control, path planning,
navigation, grasping, ...

hardware drivers, tf, kdl,
gazebo, coppeliasim,
opencv, pcl, ...

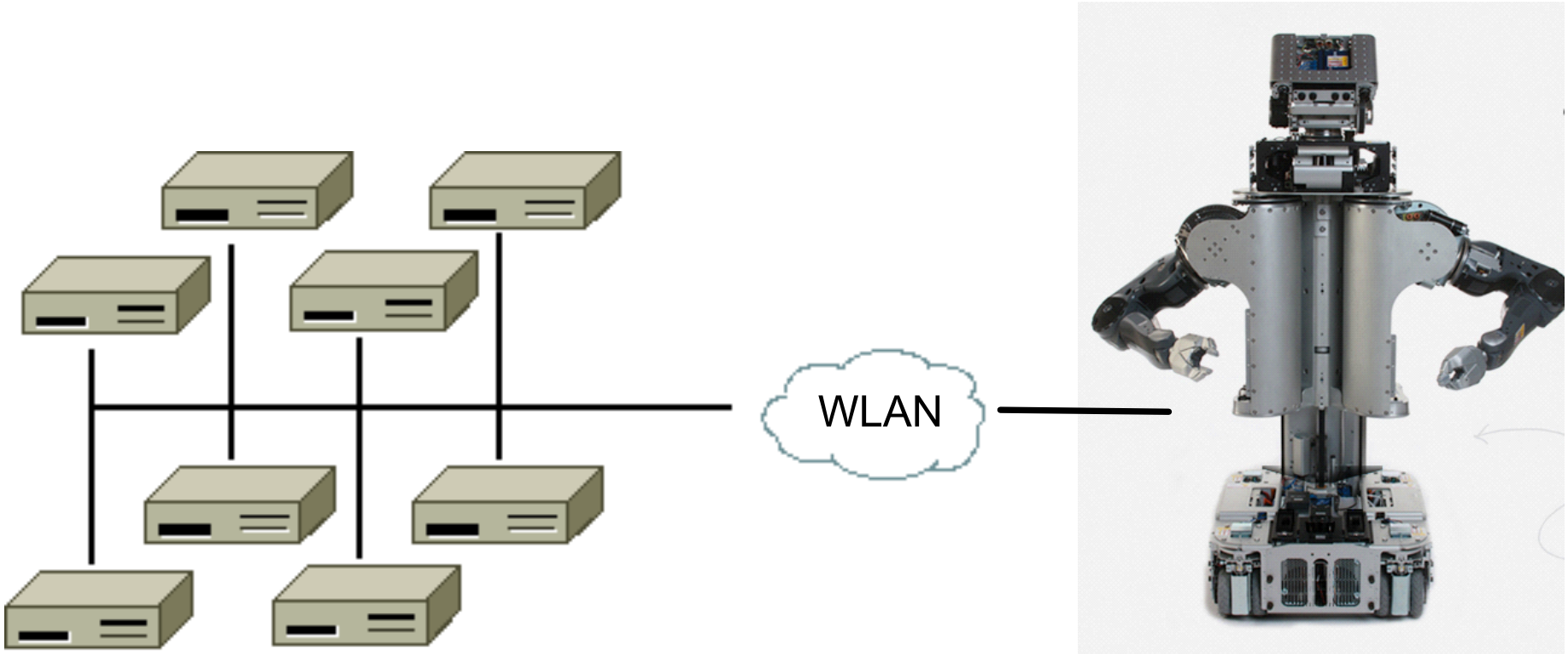
management of processes,
communication infrastructure,
packaging & build tools,
language bindings,
introspection tools, ...

10.3 Core and Libraries for ROS

- Communication infrastructure
- Language bindings
- Hardware drivers
- tf package
- kdl package
- gazebo package
- coppeliasim package
- opencv package
- pcl package

Communication infrastructure

Message passing between and among offboard and onboard machines



Communication infrastructure

Technical features:

- Peer-to-peer
- Publish and Subscribe mechanism
- Services via remote invocation
- Remote communication and control

Language bindings

- Numerous programming languages are supported, such as C++, Python, Lisp, Java.
- Different languages can be combined/bridged by so-called „language bindings“; Synonyms are „glue code“, „wrapper“, „adapter“.
- Reuse of code from a variety of other projects, e.g. OpenCV (Computer Vision Library), PCL (Point Cloud Library for 3D Data Processing).
- ROS core is said to be „light-weight“, i.e. ROS runs only at the *edge of user-provided modules*.

Hardware drivers

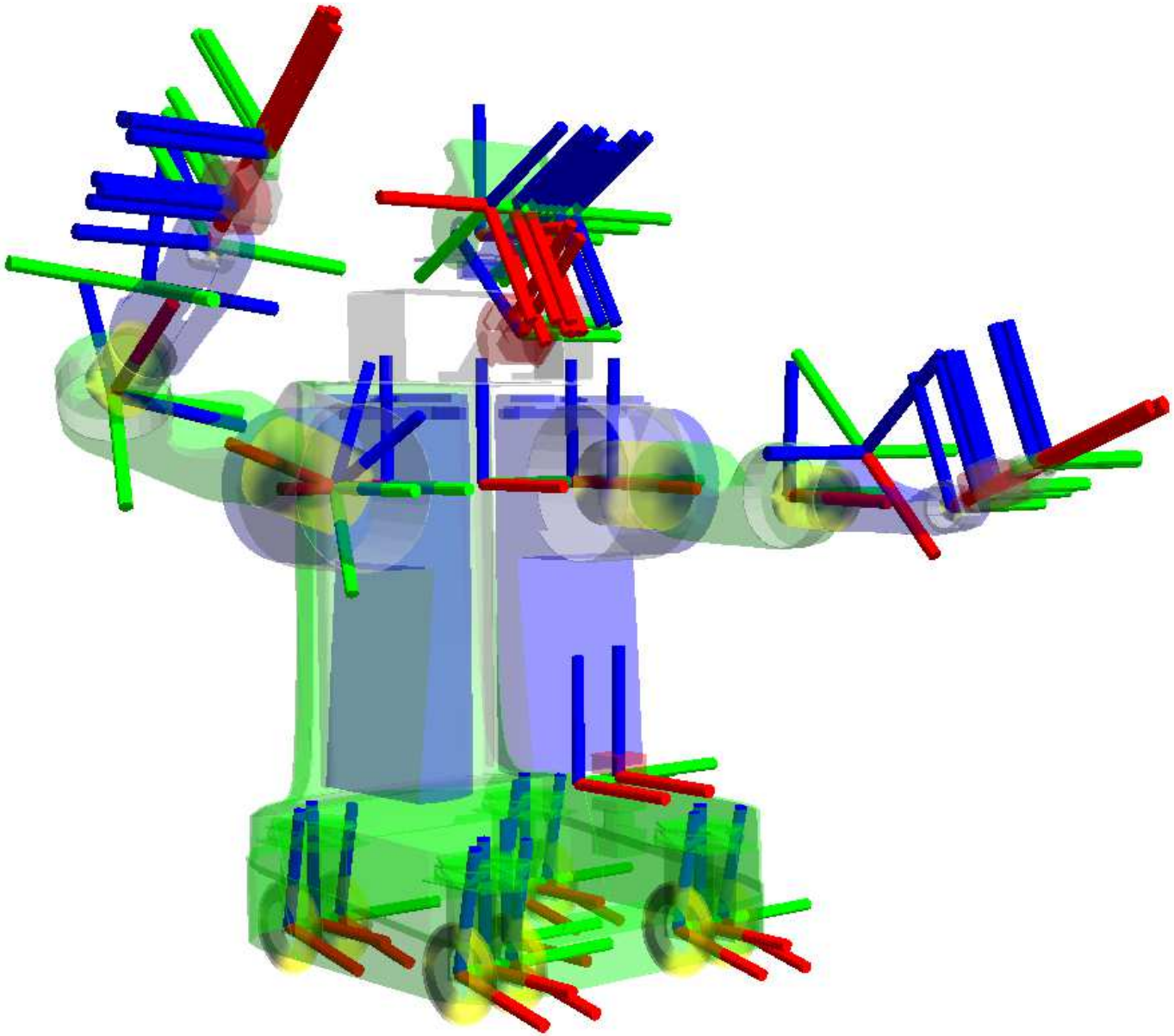
Device-specific drivers, i.e. Software to access devices (read sensor data, steer effectors). Examples of devices:

- GPS
- Camera
- Laser Scanner
- Odometry
- ...
- Joystick
- Wheels
- Joints of robot arm
- ...

tf package

- Transform points, lines, shapes, etc., within a coordinate system at any desired point in time.
- Keep track the relationship between coordinate systems over time. Allows to ask questions like:
  - What was the head pose relative to the world coordinate system, 5 seconds ago?
  - What is the pose of the object relative to the gripper coordinate system?
  - What is the current pose of the robot basis coordinate system in the map coord. sys.?
- URL: `http://wiki.ros.org/tf`

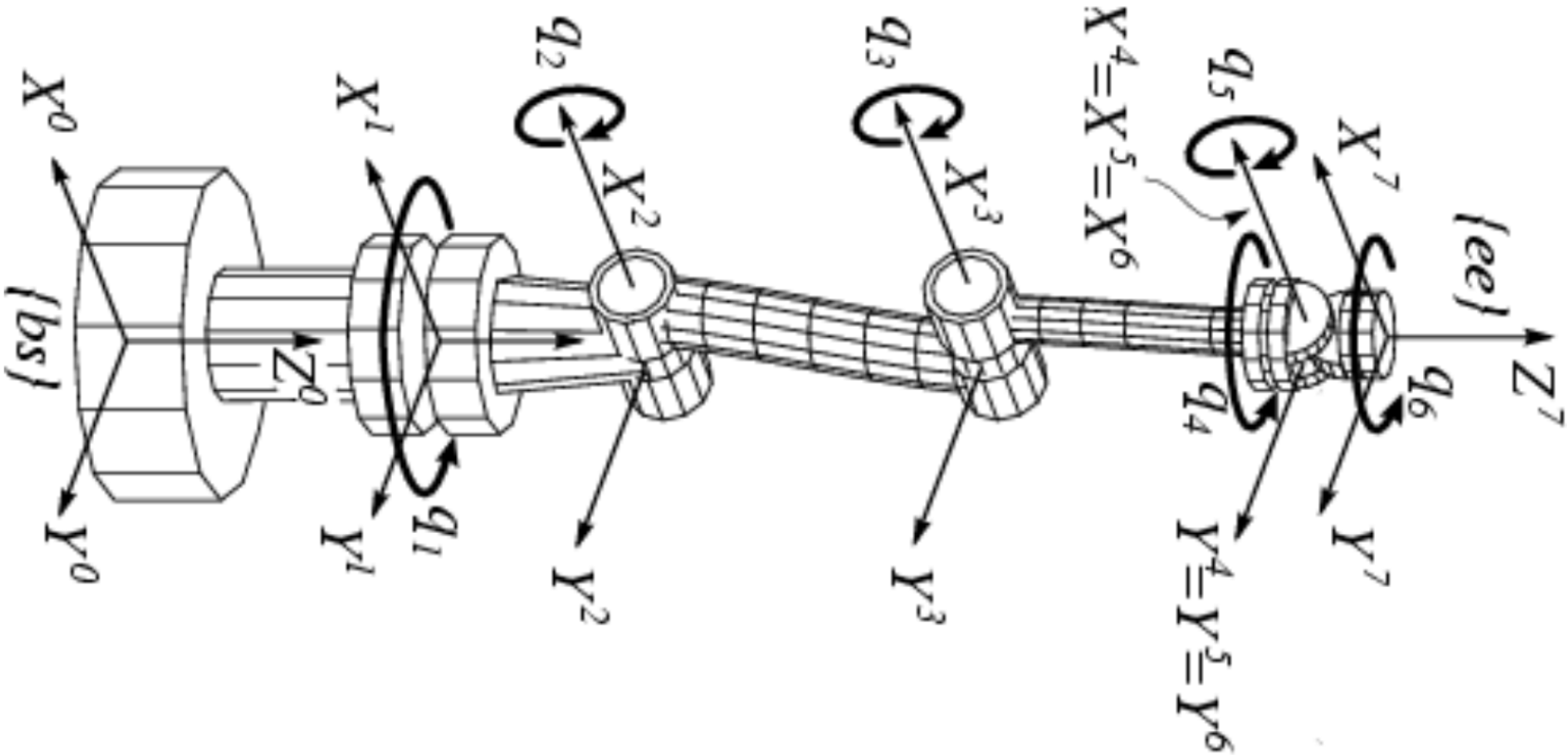
tf package



kdl package

- Kinematics and Dynamics of kinematic chains
- Solvers for forward and inverse kinematics
- Solvers for forward and inverse dynamics
- Change of reference point of a tool
- URL: `http://wiki.ros.org/kdl`

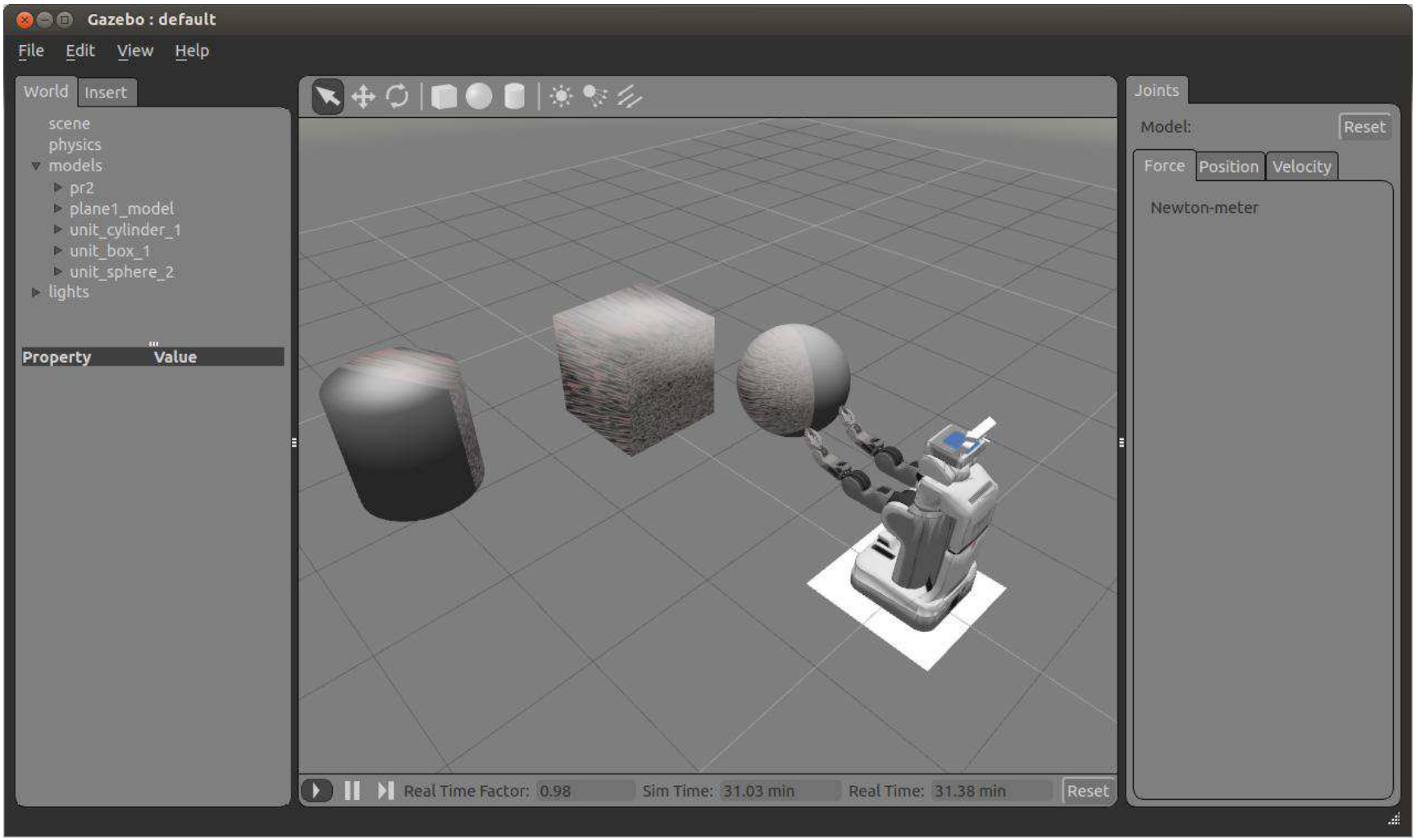
kdl package


{sq}
{ee}
X^0
Y^0
Z^0
X^1
Y^1
q_1
X^2
Y^2
q_2
X^3
Y^3
q_3
$X^4=X^5=X^6$
$Y^4=Y^5=Y^6$
q_4
q_5
X^7
Y^7
q_6
Z^7

gazebo package

- Gazebo for robot simulation (also an Open Robotics project).
- Design robots, make a map, rapidly test algorithms, train capabilities using realistic scenarios, etc.
- URLs:
  - `http://wiki.ros.org/gazebo_ros_pkgs`
  - `http://gazebosim.org`
  - `https://www.youtube.com/watch?v=k2wVj0BbtVk`

gazebo package



coppeliasim package

- Provide communication interface between ROS and CoppeliaSim, e.g. control a CoppeliaSim simulation externally using ROS messages (provided by ROS services).
- Several ROS interfaces available for CoppeliaSim, and the most flexible and natural approach is the so-called 'RosInterface'.
- URL: `https://www.coppeliarobotics.com/helpFiles/en/rosInterf.htm`

opencv package

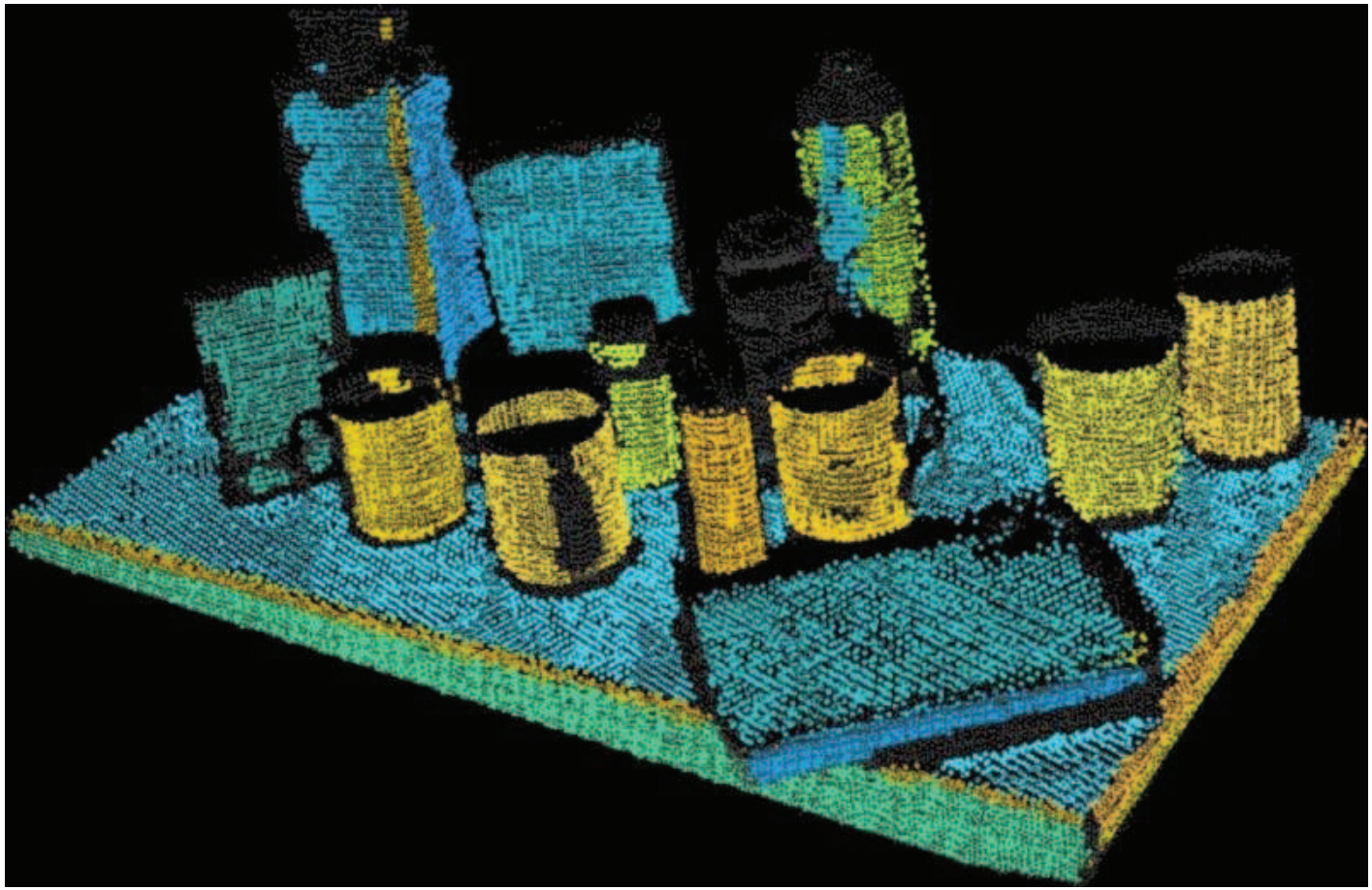
- Image Processing, Computer and Robot Vision, Machine Learning:
  preprocessing, segmentation, object recognition, image sequence processing, 3D reconstruction, camera calibration, deep learning; including specifically SIFT, YOLO, Expectation Maximisation algorithm, Artificial Neural Networks, Support Vector Machines, Random Forest, Kalman filter, etc.
- URLs:
  - `https://de.wikipedia.org/wiki/OpenCV`
  - `http://wiki.ros.org/opencv3`

pcl package

- Processing of 3D point clouds: filtering, feature extraction, surface reconstruction, registration, model fitting, segmentation.
- Possible applications: filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene.
- It is free for commercial and research use.
- URLs:
  - `http://wiki.ros.org/pcl`
  - `http://www.pointclouds.org`

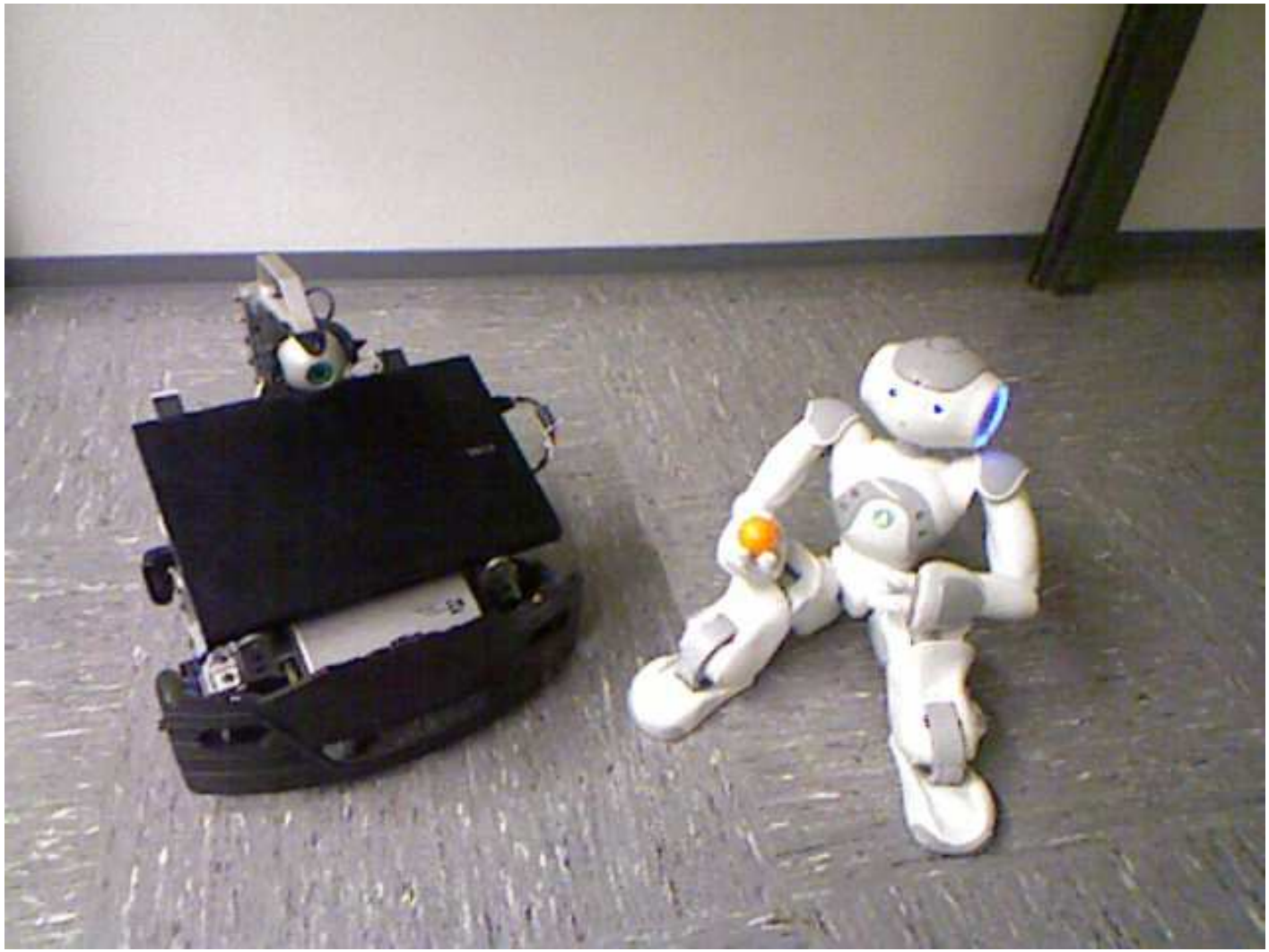
pcl package

Visualised point cloud:



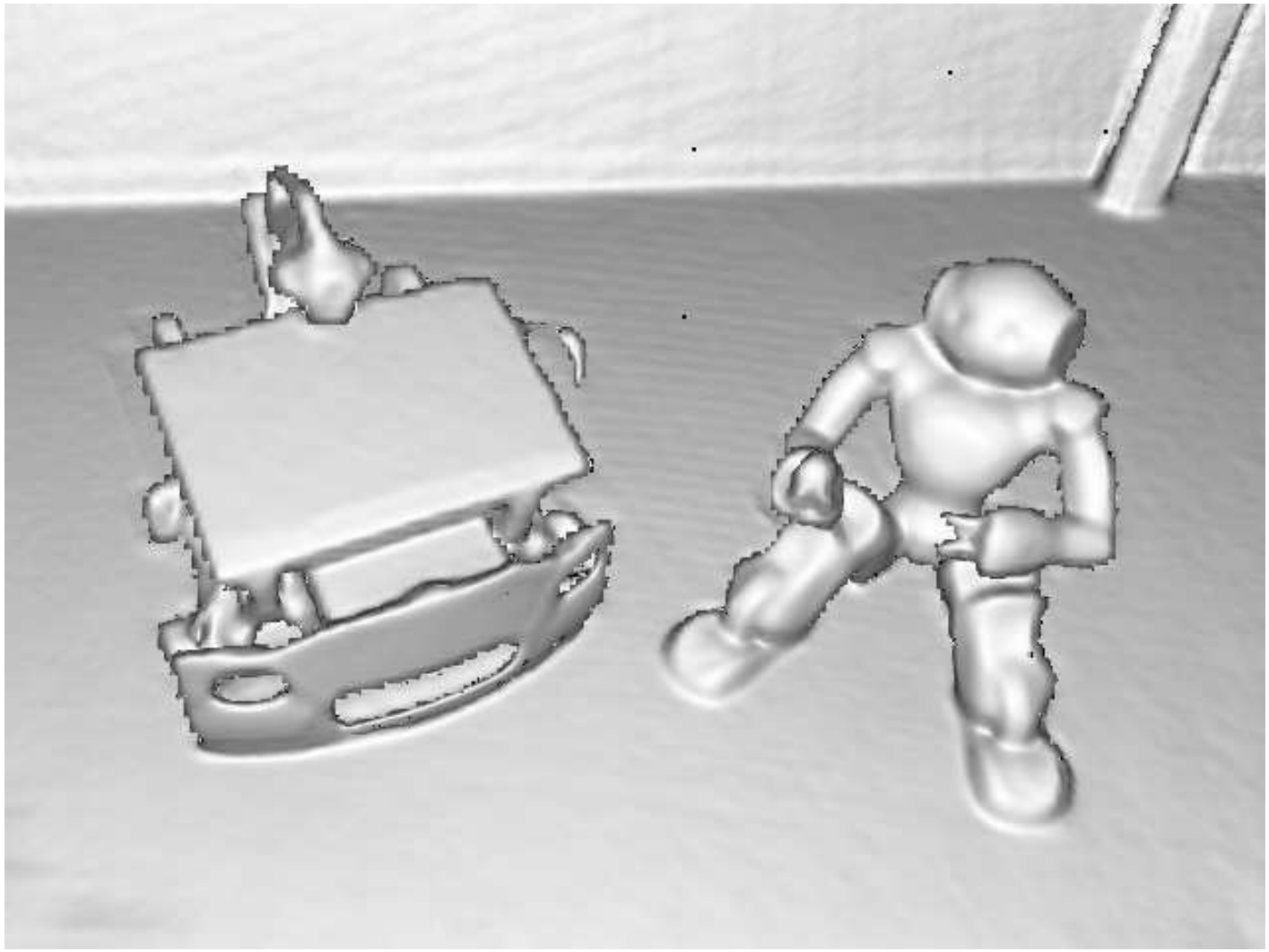
pcl package

Original color image:



pcl package

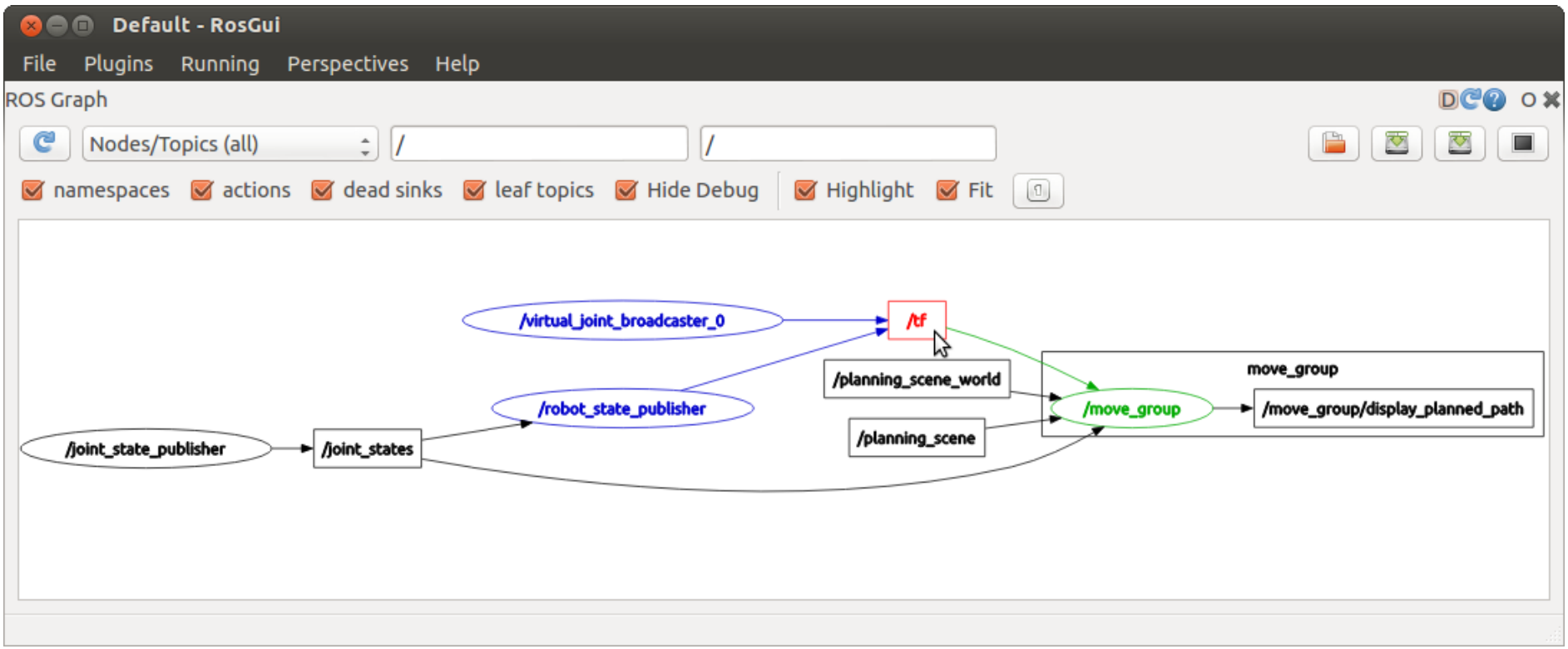
Visualised point cloud:



10.4 Network graph resources of ROS

- Example of a ROS network graph
- Overview to the network graph resources
- Nodes, Topics, Services, Parameters
- Namespaces of the network resources
- Example of a message structure

Example of a ROS network graph

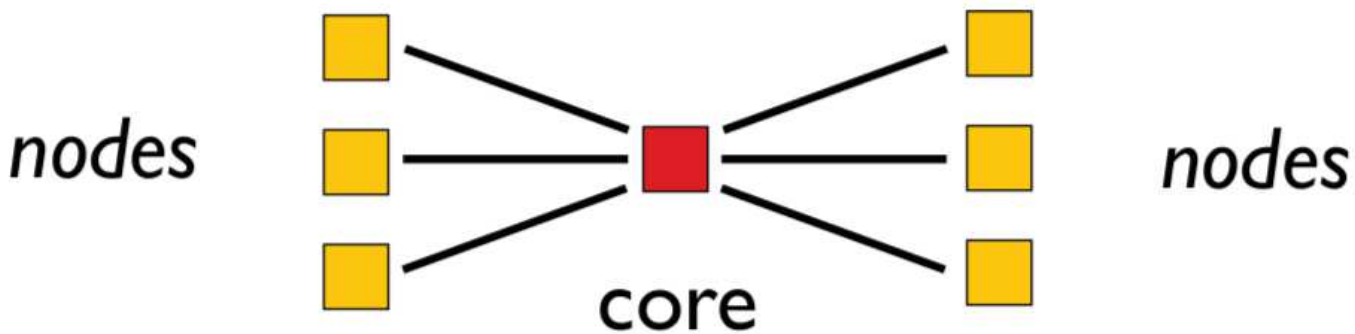


Overview to the network graph resources

- Capabilities and Applications are realised by networks of „nodes“ (processes), and communication between nodes takes place by message passing.
- An asynchronous many-to-many communication stream is called „topic“.
- A synchronous one-to-many communication stream is called „service“.
- The configurations of networks and nodes are represented by „parameters“.

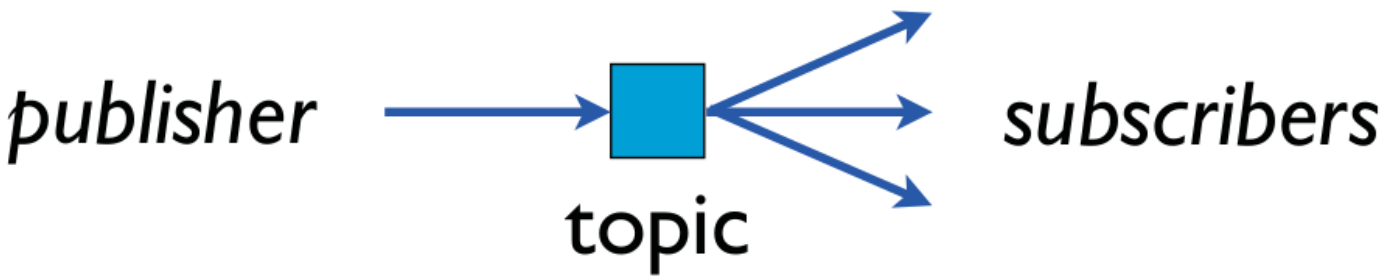
Nodes

- Node represents a process in the ROS network.
- Is a source or sink for data that is sent over the ROS network.
- May reside in different machines transparently
- Get to know one another via ROS core.



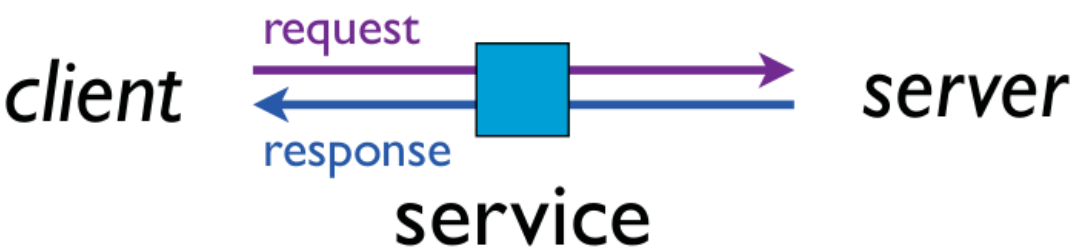
Topics

- Asynchronous many-to-many communication streams.
- Mechanism to send messages from a node to one or more other nodes.
- Follows a publisher-subscriber design pattern.
- Publish = to send a message to a topic.
- Subscribe = get called, whenever a message is published, and based that, execute a function.
- Published messages are broadcasted to all Subscribers.



Services

- Synchronous one-to-many communication streams.
- Mechanism for a node to send a request to another node and receive a response in return.
- Follows a request-response design pattern.
- A service is called with a request structure, and in return, a response structure is returned.
- Similar to a Remote Procedure Call (RPC).



Namespaces of the network resources

All ROS graph resources exist in namespaces, i.e. nodes, topics, services, parameters, messages.

The namespaces are hierarchically organised, to reduce naming collisions.

Excerpt of a namespace concerning 'camera':

```
[...]
/camera
/camera/rgb
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/depth
/camera/depth/image_raw
[...]
```

Example of a message structure

Structure of a Laser scan message:

```
Header header          # timestamp of the acquisition time of the
                       # first ray in the scan;
                       # coordinate system (frame), according to
                       # which the laser is spinning around the
                       # positive Z axis with the zero angle
                       # forward along the X axis

float32 angle_min      # start angle of the scan [rad]
float32 angle_max      # end angle of the scan [rad]
float32 angle_incr     # angular distance betw. measurements [rad]

float32 time_incr      # time between measurements [seconds]
float32 scan_time      # time between scans [seconds]

float32 range_min      # minimum range value [m]
float32 range_max      # maximum range value [m]

float32[] ranges       # range data (values < range_min
                       # or > range_max should be discarded)
float32[] intens       # maybe include intensity data
```

10.5 Exemplary implementations

- Example impl. of Publisher Node in C++
- Example impl. of Subscriber Node in C++
- Example impl. of Service Node in C++
- Example impl. of Client Node in C++

Example impl. of Publisher Node in C++

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "speaker");
  ros::NodeHandle n;
  ros::Publisher greeting_pub =
          n.advertise<std_msgs::String>("greeting", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    greeting_pub.publish(msg);
    loop_rate.sleep();
    ++count;
  }
}
```

Example impl. of Publisher Node in C++

Comments:

- speaker: Name of Publisher node
- NodeHandle n: Access point for communications
- advertise: Tell ROS core that the node intends to publish on a certain topic name
- greeting: Name of Topic
- 1000: Maximal number of messages the queue can store, which will be published step by step
- greeting_pub: Publisher object which allows to publish messages on that topic
- msg: Message object, to be filled with data, and then published, here hello world ...
- publish(msg): Send message

Example impl. of Subscriber Node in C++

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;
  ros::Subscriber sub =
          n.subscribe("greeting", 1000, greetingCallback);
  ros::spin();
}

void greetingCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}
```

Example impl. of Subscriber Node in C++

Comments:

- listener: Name of Subscriber node
- subscribe: Tell ROS core that the node wants to receive messages on a given topic
- greetingCallback: Callback function uses/processes/handles the message, which is sent from the publisher
- ROS_INFO: Similar to printf
- ros::spin(): In a loop, the greetingCallback function is called whenever the relevant message is obtained from the publisher.

Example impl. of Service Node in C++

```
int main(int argc, char **argv)
{ ros::init(argc, argv, "add_two_numbers_server");
  ros::NodeHandle n;

  ros::ServiceServer service =
          n.advertiseService("add_two_ints", add_int);
  ROS_INFO("Ready to add two ints.");
  ros::spin();
  return 0;
}

bool add_int(beginner_tutorials::AddTwoInts::Request  &req,
             beginner_tutorials::AddTwoInts::Response &res)
{ res.sum = req.a + req.b;
  ROS_INFO("request: x=%ld, y=%ld",
          (long int)req.a, (long int)req.b);
  ROS_INFO("sending back response: [%ld]",
          (long int)res.sum);
  return true;
}
```

Example impl. of Service Node in C++

Comments:

- add_two_numbers_server: Name of Service node
- advertiseService: Service is created and advertised
- add_two_ints: Name of Service
- add_int: Name of method, providing the Service
- AddTwoInts: Data type, to store two integers and the result of addition
- ros::spin(): In a loop, the Service node is waiting for addition requests from clients, and if called then the addition service will be provided.

Example impl. of Client Node in C++

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_numbers_client");
  if (argc != 3)
  { ROS_INFO("usage: add_two_numbers_client X Y");
    return 1;
  }
  ros::NodeHandle n;
  ros::ServiceClient client =
          n.serviceClient<beginner_tutorials::AddTwoInts>(
                  "add_two_ints");
  beginner_tutorials::AddTwoInts srv;
  srv.request.a = atol(argv[1]);
  srv.request.b = atol(argv[2]);
  if (client.call(srv))
    ROS_INFO("Sum: %ld", (long int)srv.response.sum);
  else
  { ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }
  return 0;
}
```

Example impl. of Client Node in C++

Comments:

- add_two_numbers_client: Name of Client node
- serviceClient: Creates a client object client for the add_two_ints service.
- srv: Variable of structured data type AddTwoInts
- client.call(srv): Client call of Service to add two integers, which are stored in srv.request.a and srv.request.b, and the result (provided service) is in srv.response.sum

10.6 Packages and stacks in ROS

- Application-related organisation of code
- Packages
- Stacks

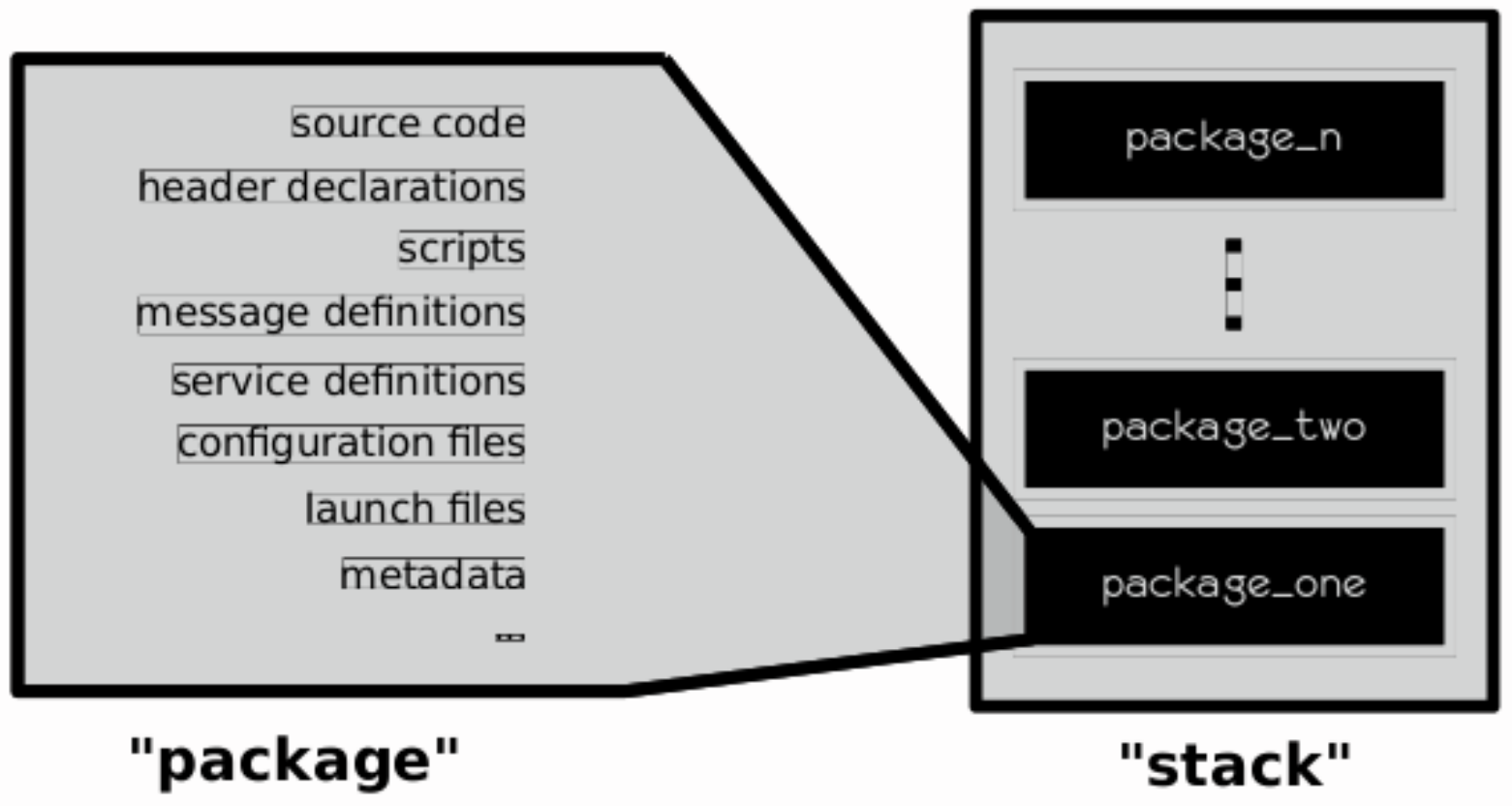
Application-related organisation of code

The application-related ROS code is grouped at three levels:

- Package
  - A named collection of software that is built and treated as atomic.
- Stack
  - A named collection of related Packages.
- Distribution
  - A Distribution is composed of Stacks.

Packages and Stack

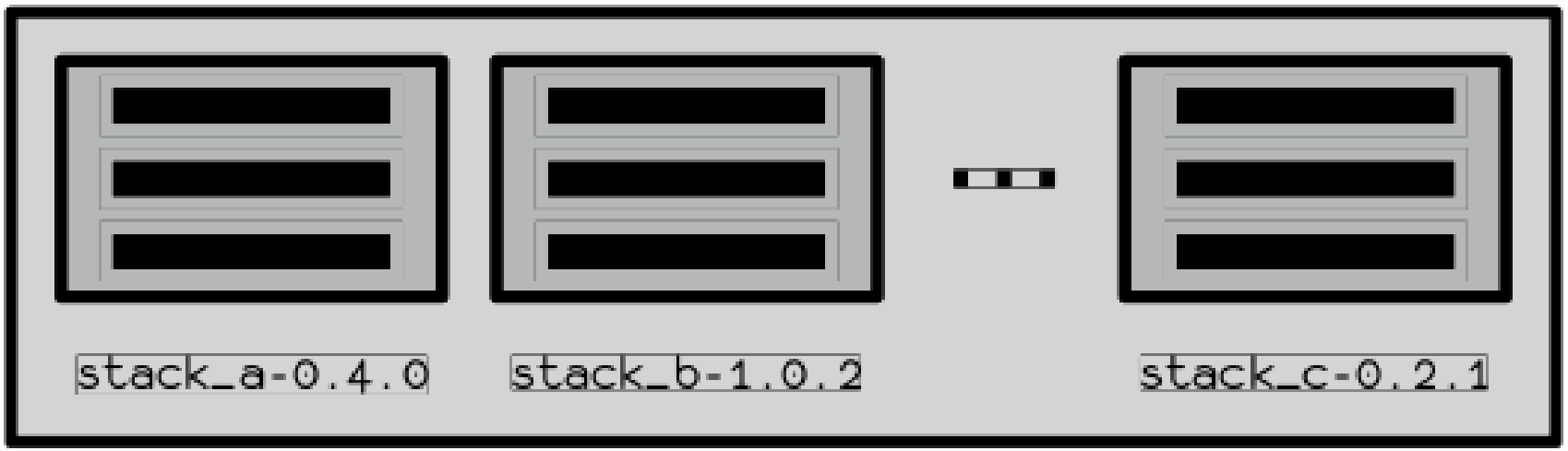
A ROS „stack“ is composed of ROS „packages“.



E.g., the so-called „navigation stack“ includes several packages contributing methods for robot navigation.

Stacks and Distribution

A ROS „distribution“ is composed of ROS „stacks“.



E.g., latest distribution „ROS Noetic Ninjemys“, release date May 2020

10.7 Command-line tools in ROS

- Filesystem Management Tools
  - rospack, roscd, rosls, roscreate-pkg, roscreate-stack, rosmake, rqt_dep, ...
- Start-up and Process Launch Tools
  - roscore, rosrun, roslaunch, ...
- Introspection and Command Tools
  - rosmsg, rosnode, rostopic, rosparam, rosservice, ...
- Logging and Graphical Tools
  - rosbag, rqt_console, rqt_graph, rqt_plot, rqt_bag, ...

Filesystem Management Tools

Overview:

rospack	A tool inspecting packages
roscd	Changes directories to a package
rosls	Lists package information
roscreate-pkg	Creates a new ROS package
roscreate-stack	Creates a new ROS stack
rosmake	Builds a ROS package
rqt_dep	Displays package structure and dependencies
...	...

Start-up and Process Launch Tools

Overview:

roscore	Start the core of ROS
rosrun	Run an executable node
roslaunch	Run a network of nodes
...	...

Introspection and Command Tools

Overview:

rosmsg	Information concerning messages
rosnode	Information and command concerning nodes
rostopic	Information and command concerning topics
rosparam	Information and command concerning parameters
rosservice	Information and call of services
...	...

Logging and Graphical Tools

Overview:

rosbag	Replay of messages from a bag file
rqt_console	Display and filter messages published on rosout.
rqt_graph	Display graph of running nodes and topics.
rqt_plot	Plot data from one or more topic fields.
rqt_bag	Visualize, inspect, and replay histories (.bag files) of messages.
...	...

Literature

- Some slides in this presentation are taken or adapted from few of URLs below.
- URLs:
  - http://wiki.ros.org/ROS/Tutorials
  - http://wiki.ros.org/Courses
  - Rodrigo Ventura, http://mediawiki.isr.ist.utl.pt/images/3/3d/Intro_to_ros.pdf
  - Alec Poitzsch, http://courses.csail.mit.edu/6.141/spring2014/pub/lectures/Lec05-ROS-Lecture.pdf
  - Jonathan Bohren, https://de.scribd.com/document/328846092/ros-cc-1-intro-jrsedit-pdf
  - ROS Cheat Sheet (for Melodic Morenia), https://pk.sedenius.com/wp-content/uploads/2020/08/sedenius_ros_cheatsheet.pdf