Karlsruhe Institute of Technology

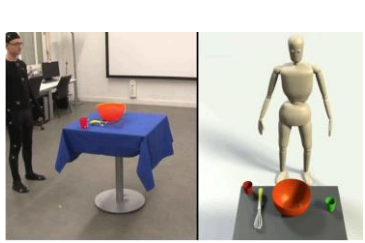
High Performance Humanoid Technologies

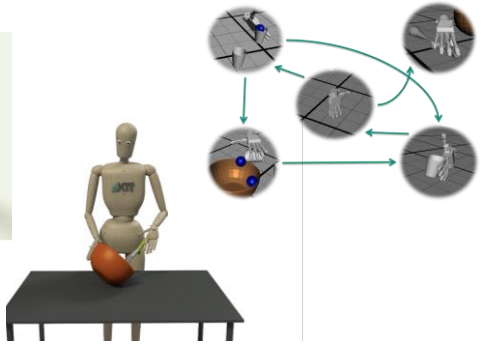
Robotics I: Introduction to Robotics

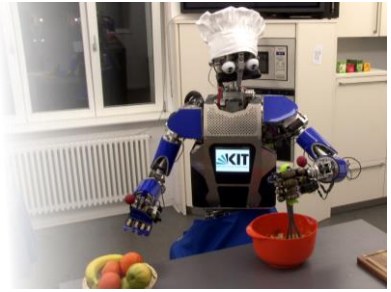
Chapter 10 – Robot Programming and Programming by Demonstration

Tamim Asfour

https://www.humanoids.kit.edu







Contents

- **Motivation**
- Classical robot programming (overview)
- Graphical robot programming (statecharts)
- Programming by Demonstration
- Planning

Motivation – New Requirements in Production

- Small-scale production
- Single-item production (e.g., prototypes)
- Products with …
  - … many variants
  - … high reconfigurability

Flexible production

Motivation – New Requirements in the Service Sector

- Commerce:
  - Picking and palletizing of products
  - Stocking shelves
- Quality assurance
- Care:
  - Support in rehabilitation and care
- Craft industry:
  - Handling in carpentries and metalworking companies

Motivation – Requirements to Humanoid Robots

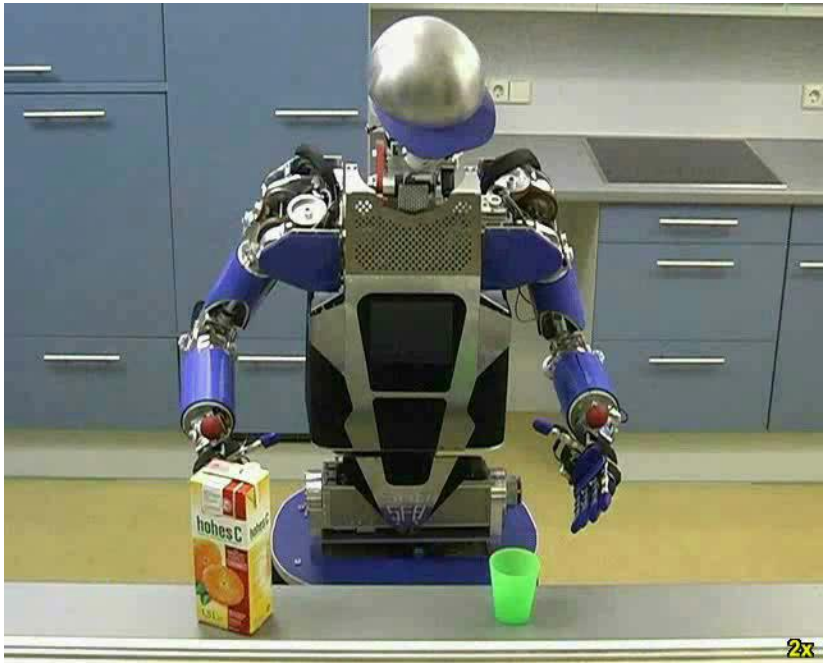
- Manipulation of arbitrary objects
- Independent solving of complex tasks
- Deployment next to humans

Complex environment!

&

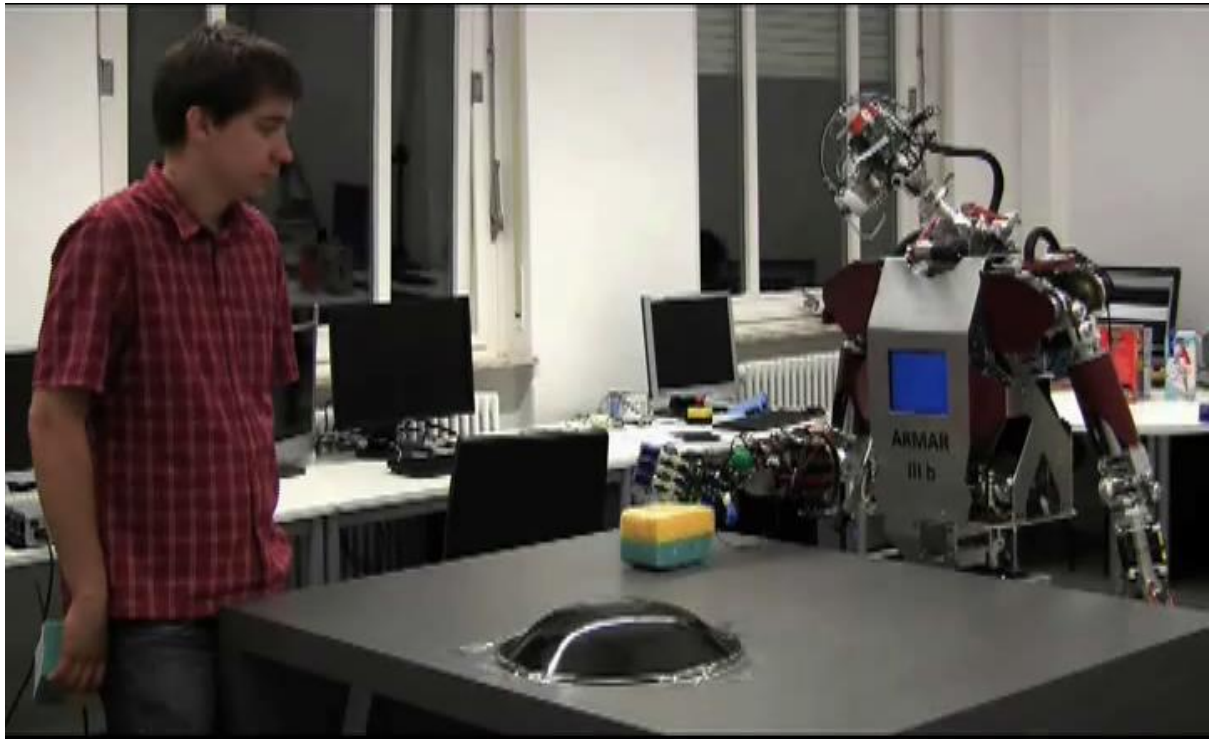
Many degrees of freedom!

How to program robots?



Programming by Demonstration (PbD)

- Observe humans
- Interpret human demonstrations
- Map to robot



Contents

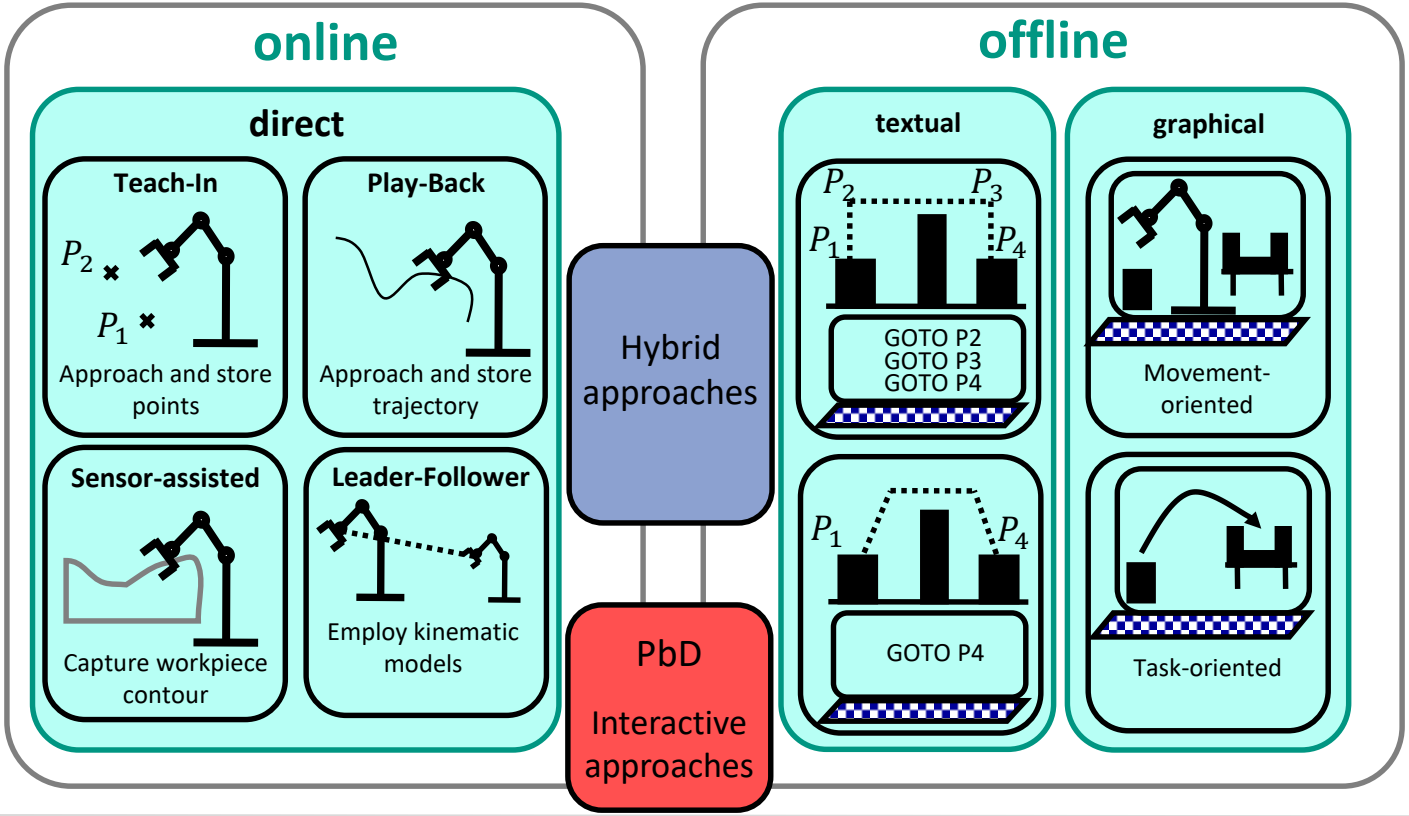
- Motivation
- Classical robot programming (overview)
- Graphical robot programming (statecharts)

Not relevant for the exam

- Programming by Demonstration

- Planning

Robot Programming Approaches



Classifying Robot Programming Approaches

Criteria:

I. **Location of programming**
   - **Online programming**
   - **Offline programming**

II. Degree of abstraction of programming
   - Implicit programming
   - Explicit programming

III. Type of programming
   - Direct programming
   - Textual approaches
   - Graphical approaches
   - Hybrid approaches

Robot Programming Approaches

online

The programming is done **directly on the robot** (at the robot controller).

In the literature, this is also referred to as **direct** programming.

Hybrid approaches

offline

The programming is done **without the robot** using textual, graphical, interactive methods.

In the literature, this is also referred to as **indirect** programming.

Classifying Robot Programming Approaches

Criteria:

I. Location of programming
- Online programming
- Offline programming

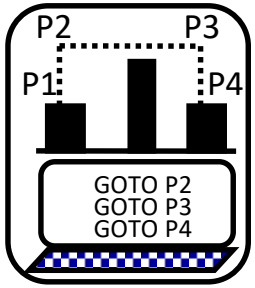
II. Degree of abstraction of programming
- **Explicit programming**
- **Implicit programming**

III. Type of programming
- Direct programming
- Textual approaches
- Graphical approaches
- Hybrid approaches

Degree of Abstraction of Programming

Explicit or robot-oriented programming (imperative)

Movements and gripper commands are directly embedded into a programming language

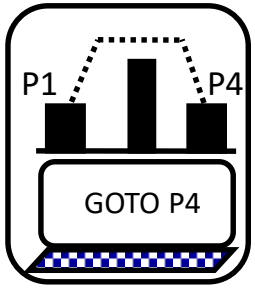


„How is it done?“

Degree of Abstraction of Programming

Implicit or task-oriented programming (declarative)

The task to be executed by the robot is described, e.g. based on states.



„What is to be done?“

- The abstract form of programming is carried out in the following steps:
  1. Modeling the environment
  2. Specifying the task
  3. Generating robot programs
- The execution of the robot program can be verified in simulation.

Classifying Robot Programming Approaches

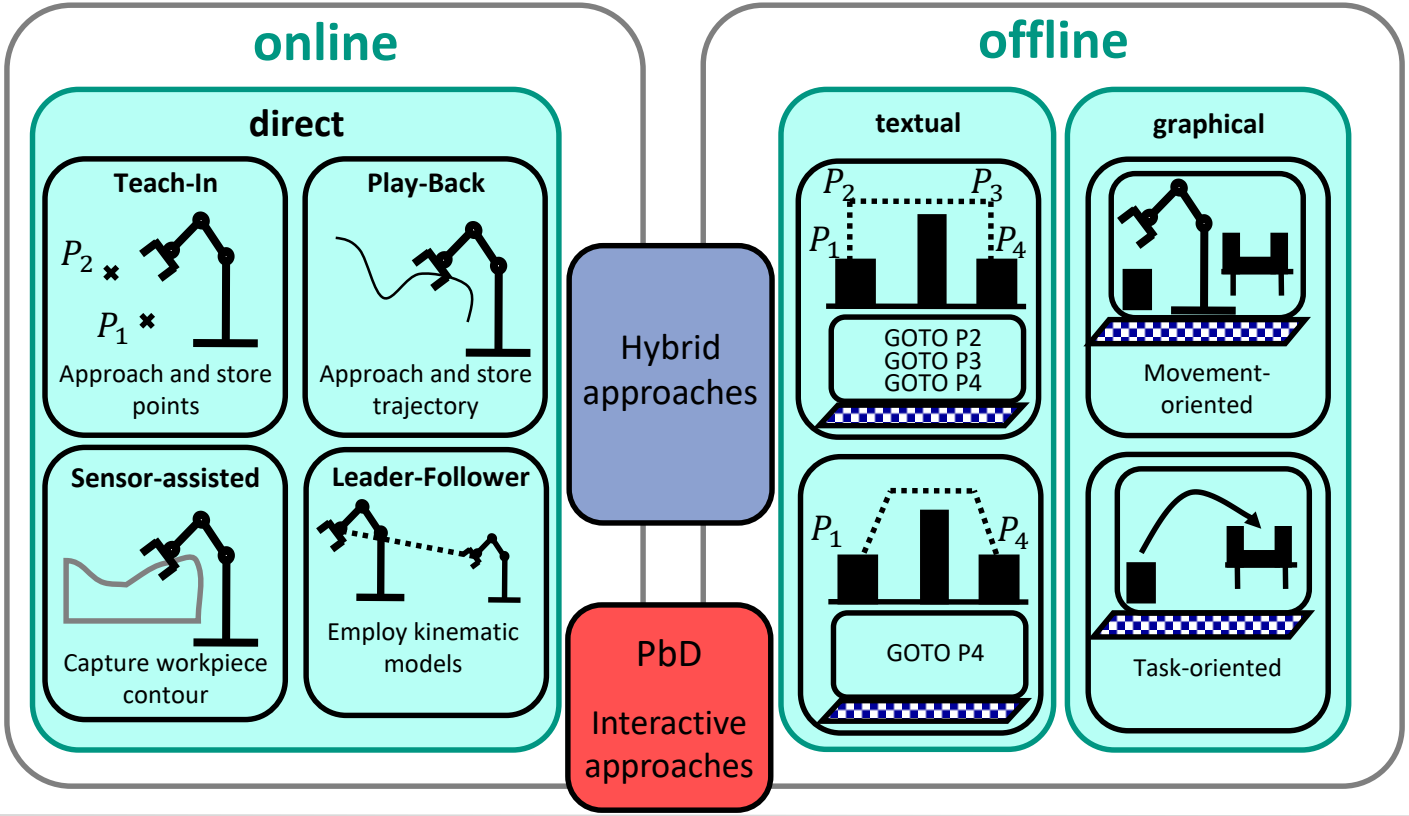
Criteria:

I. Location of programming
- Online programming
- Offline programming

II. Degree of abstraction of programming
- Explicit programming
- Implicit programming

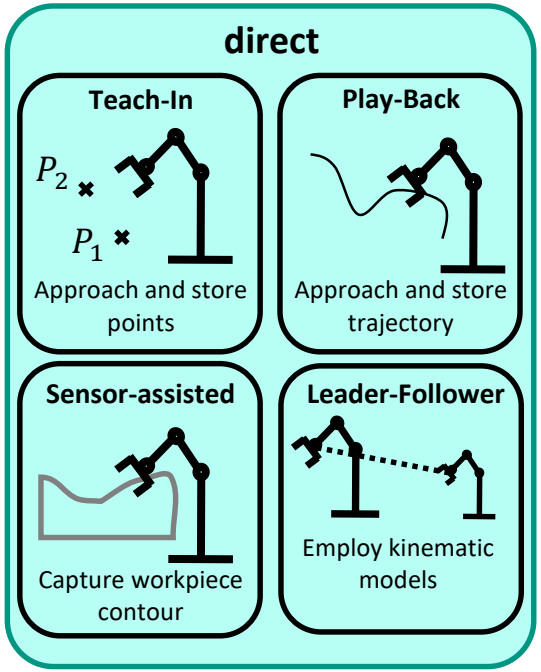
III. Type of programming
- **Direct programming**
- **Textual approaches**
- **Graphical approaches**
- **Hybrid approaches**

Direct Robot Programming Approaches



Direct Programming

- Teach-In programming
- Play-Back programming
- Leader-Follower programming (special case teleoperation)
- Sensor-assisted programming

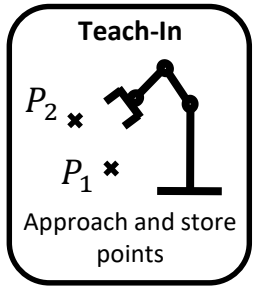


Direct Programming: Teach-In

Teach-In programming

- Approach distinctive points of the trajectory with manual control (Teach Box, Teach Panel)

- Functionality of a Teach Box:
  - Individually control joints
  - Control end effector in Cartesian space
  - Safe and remove points
  - Input of velocities
  - Input of gripper control commands
  - Starting and stopping whole programs



Example: Teach Box of Mitsubishi RM-501

5 DoF + Gripper
1.2 kg payload
Range of 45 cm

Movement:

- Base
- Shoulder
- Elbow
- Wrist
- Gripper rotation
- Gripper

Direct Programming: Teach-In

Approach during Teach-In

- Approach distinctive points of the trajectory
- Trajectory = Sequence of via points
- Store joint angles
- Afterwards add certain parameters to stored joint angles, such as velocity, acceleration, etc.
- Application:
  - In manufacturing (spot-welding, riveting)
  - Manipulation tasks (take parcels from conveyor belt)

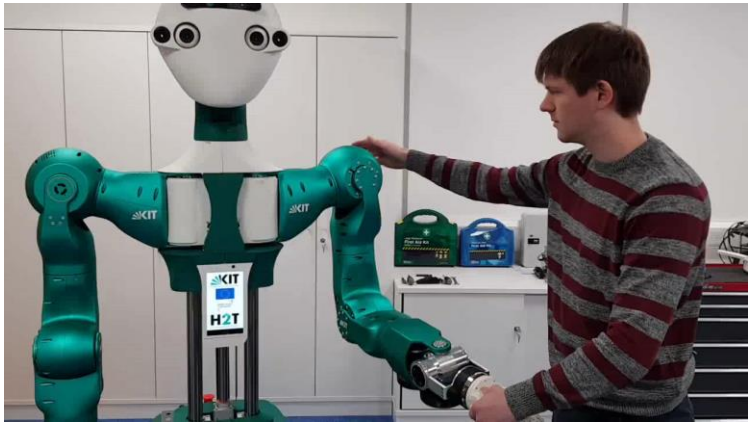
Direct Programming: Play-Back

Play-Back (manual) programming

- Set robot to zero torque control (Robot can be guided by operator)
- Also called **kinesthetic teaching**
- Follow desired trajectory with robot
- Store joint angles:
  - Automatic (pre-defined sampling frequency) or
  - Manual (by key press)
- Application:
  - Mathematically hard to describe movements
  - Integration of domain knowledge of the operator (e.g., craftmanship)
  - Typical areas of application: Lacquering, painting, gluing

Direct Programming: Play-Back

Play-Back (manual) programming



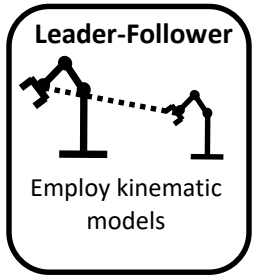
Direct Programming: Play-Back

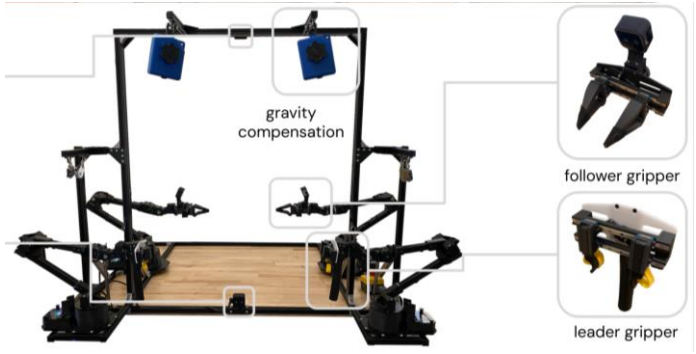
- Disadvantages of Play-Back programming:
  - Direct contact with robot → safety hazard
  - High storage requirements with high sampling frequency (nowadays not an issue anymore)
  - Means of corrections rather limited

Direct Programming: Leader-Follower

Leader-Follower programming:

- Operator guides a another (e.g., smaller and easier-to-move) leader robot, being a kinematic model of the follower robot
  - Movements are transferred to follower robot
  - Movements are executed synchronously
- Applications:
  - Handling of large payloads or large robots
  - Collecting dexterous bimanual demonstrations (e.g., ALOHA)
- Advantages and disadvantages:
  - Enables programming very heavy robots
  - Enables a human to act through a robot
  - Tends to be expensive, two robots are needed





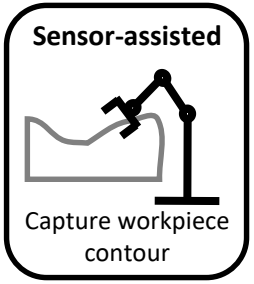
ALOHA 2 Team (2024)

Tony Z. Zhao, Vikash Kumar, Sergey Levine, Chelsea Finn: Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. RSS'23.

Direct Programming: Sensor-Assisted

Sensor-assisted programming

- Manual
  - Operator guides programming pen along the trajectory to be followed
  - Capture the movement with external sensors (e.g., cameras, laser scanners)
  - Calculate inverse kinematics
  - Store trajectory as sequence of joint angle values
- Automatic
  - Specify start and goal points
  - Sensory sampling of expected contour (e.g., via force-torque sensor)
- Disadvantages:
  - Errors during capturing of the trajectory; occlusions of parts of the trajectory
  - No inclusion of the operator's experiences and domain knowledge
- Applications:
  - Grinding, sanding, deburring of work pieces

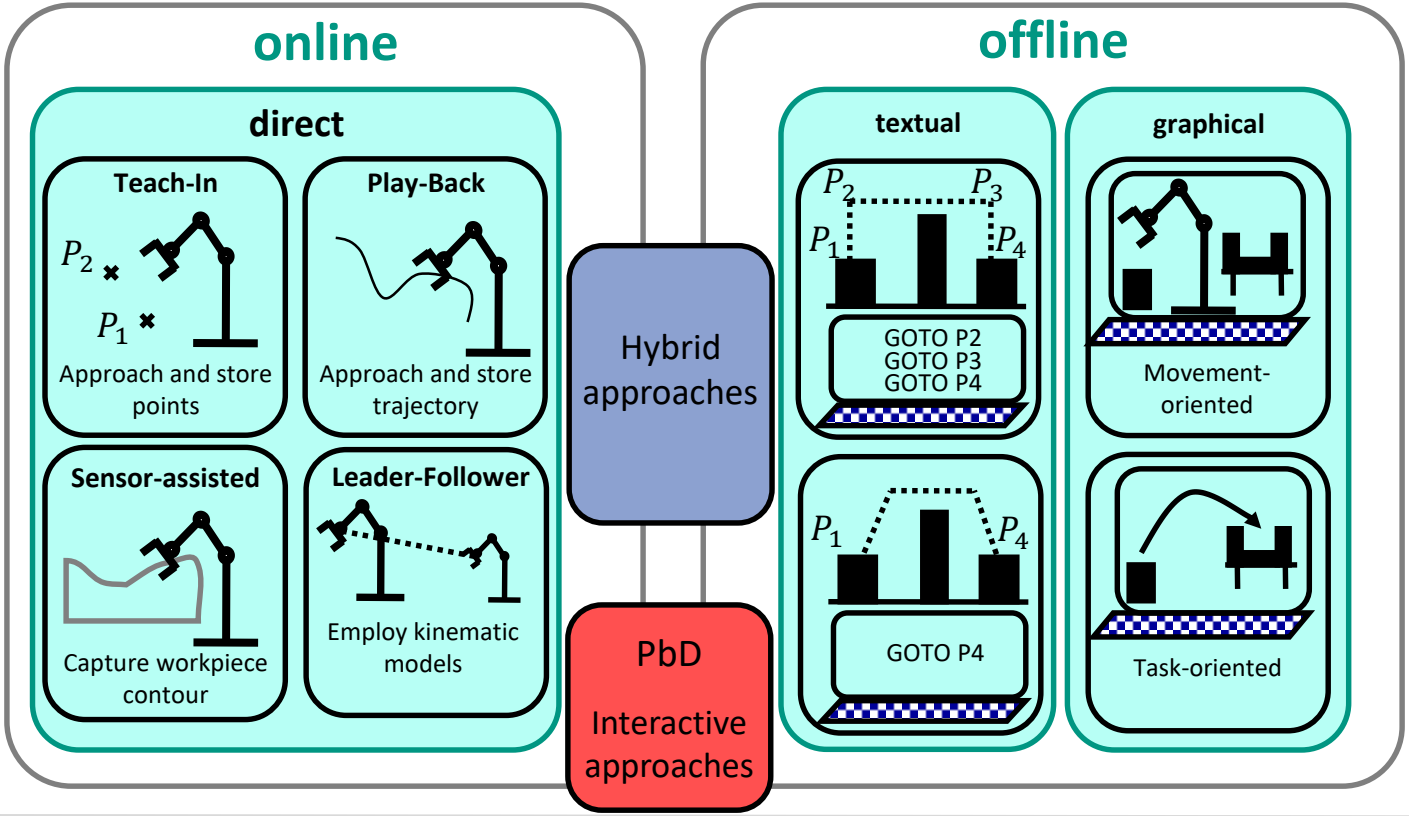


Direct Programming: Summary

- **Advantages:**
  - Fast for simple trajectories
  - Immediately executable
  - Robust to certain errors
  - No programming knowledge of operator required
  - No environment model necessary

- **Disadvantages:**
  - High effort for complex trajectories
  - Only feasible on the robot and with the robot
  - Specific to the respective robot
  - Risk of injury through robot
  - No adaptation to new circumstances

Textual Robot Programming Approaches



Textual Robot Programming Approaches

- Programming using an extended, higher-level programming language such as PASRO, VAL, V+ (Unimation/Stäubli), RAPID (ABB), KRL (KUKA), …

  → **Robot control program**

- Advantages:
  - Programming can be done independent of the robot
  - Structured, clear programming logic
  - Write complex programs
    (Making use of knowledge base, world or environment model, sensors)

- Disadvantages:
  - Operator needs programming knowledge

Textual Robot Programming Approaches: DIN 66025

- Command coding according to **DIN 66025**
- Program = List of numbered sets

  Example:

  `N70 G00 X20 Z12`

  Move tool with rapid traverse (`G00`) to position `X=20` and `Z=12`.
  (`N` = set number)

- Languages:
  - APT (Automatically Programmed Tools) — 1961 MIT
  - EXAPT (Extended Subset of APT) — 1966 TH Aachen

- Example

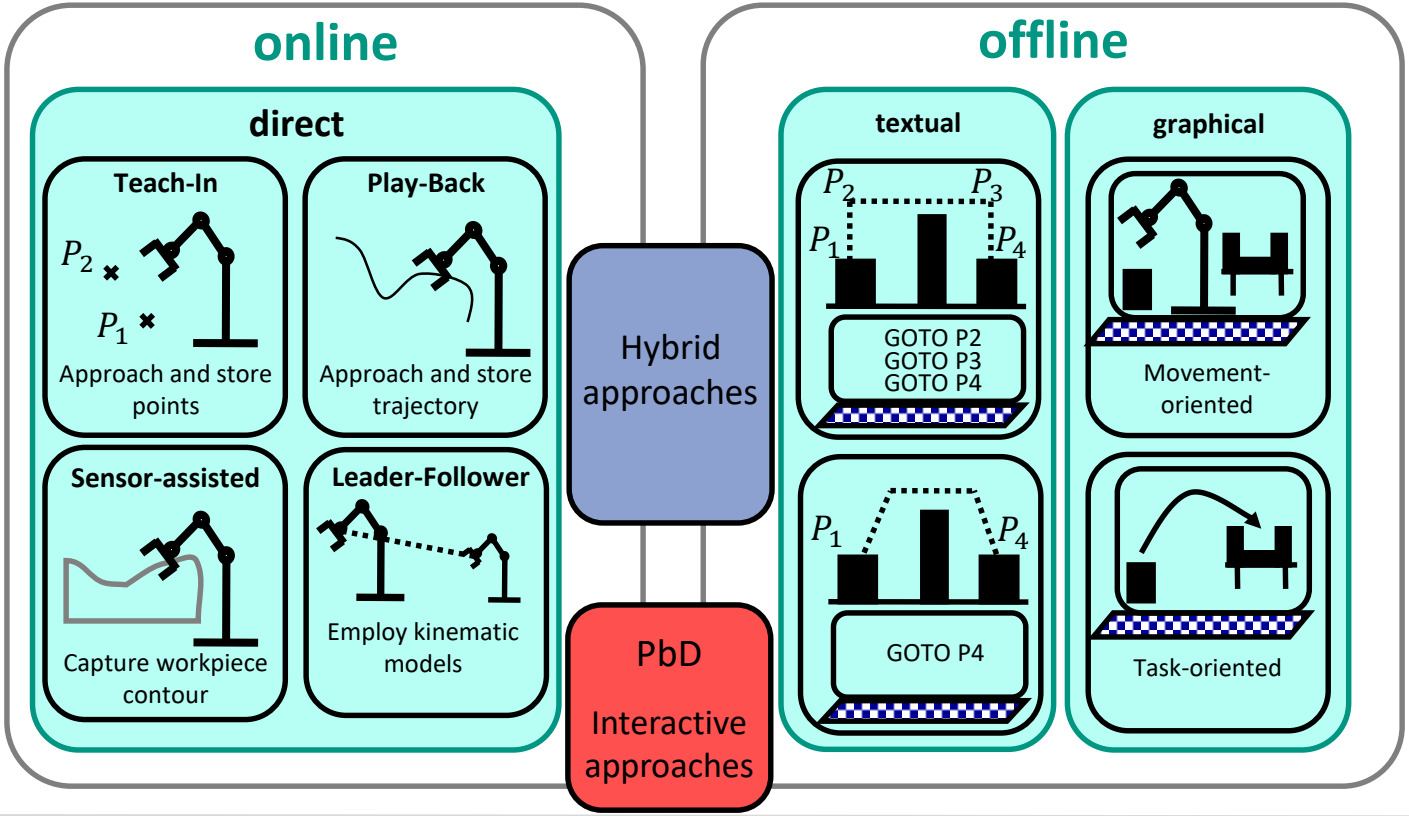
```
P1=POINT/20,12      Variable definition
Rapid               Rapid traverse
GOTO/P1             Positioning at P1
```

Textual Robot Programming Approaches : SPS

- **VPS: Connection-programmed control (German: “Verbindungsprogrammierte Steuerung”, historic)**
  - Control done through hardware
  - „Program change“ = Hardware change

- **PLC: Programmable Logic Controller (German “SPS: Speicherprogrammierbare Steuerung”)**
  - Control procedure is programmed
  - Large flexibility

Hybrid Robot Programming Approaches



Hybrid Approaches

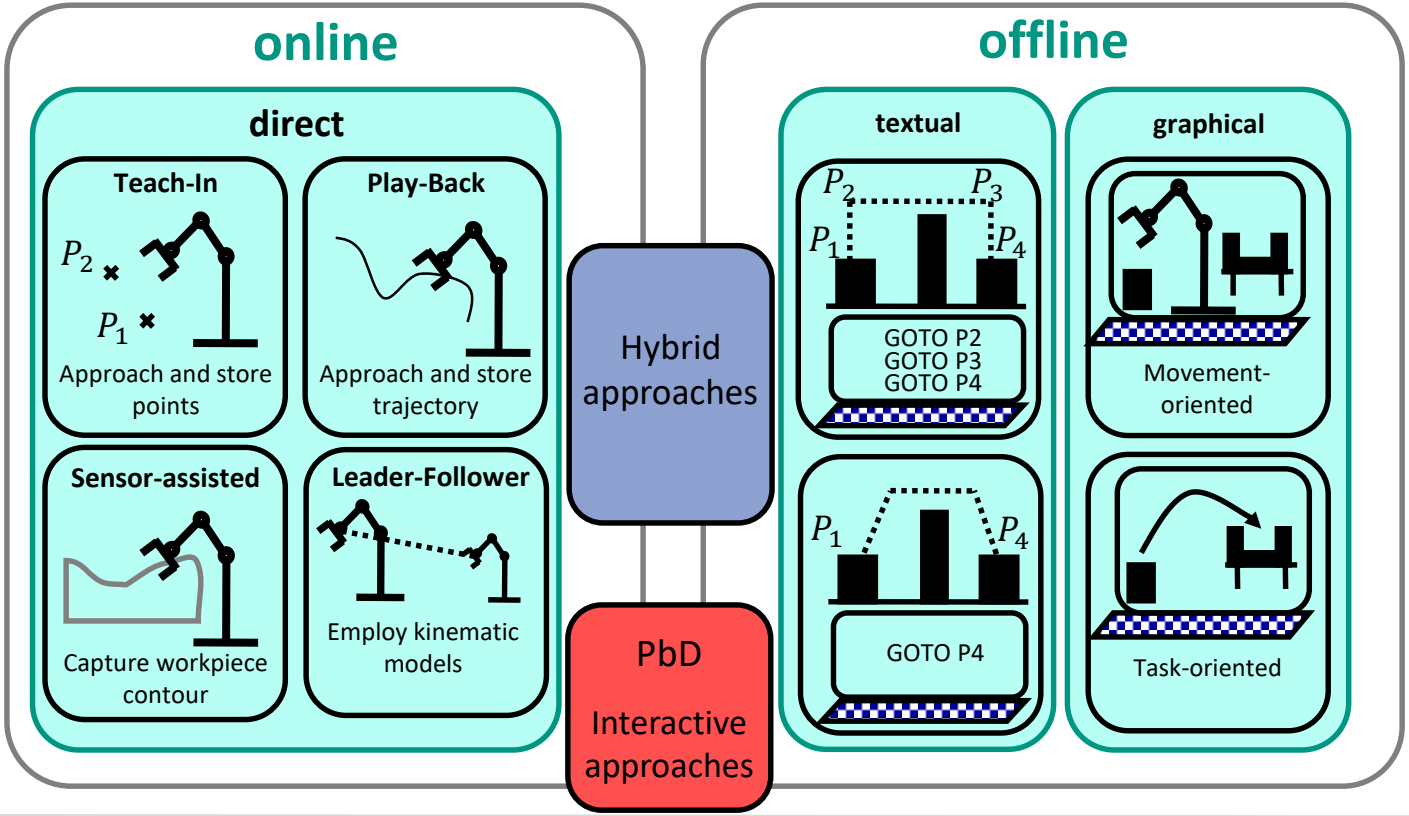
- Graphical programming based on capturing the user demonstration with sensors

  → Simulation of robot programs

- Advantages:
  - Programmer needs less programming knowledge compared to textual programming
  - Easy programming, easy error detection
  - Fast creation of complex programs (rapid prototyping)

- Disadvantages:
  - Sensory capturing still too imprecise (as with all online methods)
  - Capable hardware for signal analysis, modelling, … needed
  - Complex models needed

Graphical Robot Programming Approaches

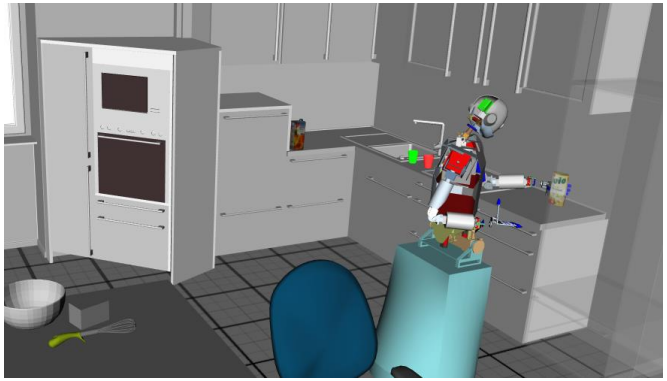


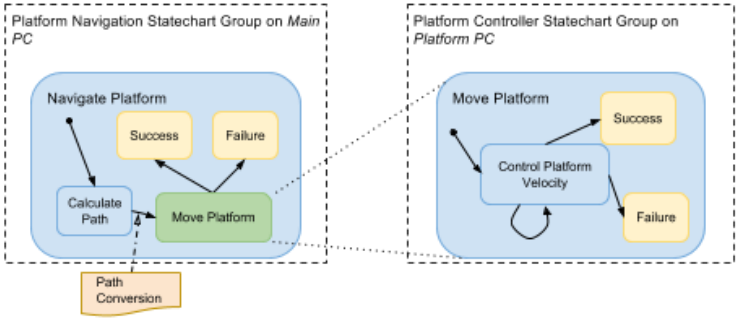
Contents

- Motivation
- Classical robot programming (overview)
- **Graphical robot programming (statecharts)**
- Programming by Demonstration
- Planning

Graphical Robot Programming Approaches

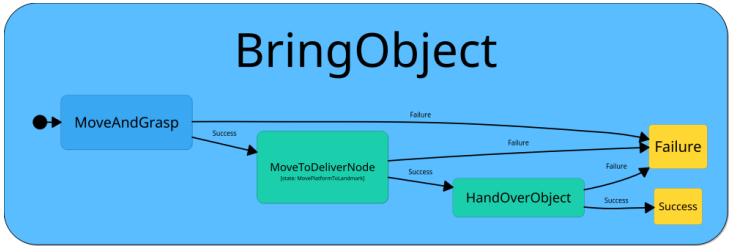
- Two fundamentally different variants
  - **Virtual Teach-In**
    - Manipulation of robot and environment in 3D visualization
    - Store movements
    - Needs accurate 3D model of robot and environment
  - **Graphical modelling formalisms**
    - Finite state machines
    - Petri nets
    - Statecharts





Representation of Robot Actions Using Statecharts

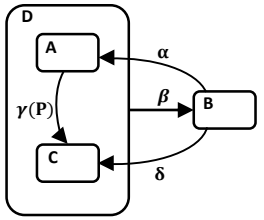
- Why Statecharts to program robots?
- **Learn** actions from observation of humans is one of the most difficult problems
  - Perception
  - Embodiment
  - Uncertainties during execution
- **Textual action programming** is hard
  - System complexity
  - Robot skills are heavily state-dependent
  - Skills usually consist of subskills
  - Textual programming can be unclear



→Graphical programming of robot actions: **Statecharts**

Graphical Modeling Formalisms

- Harel Statechart formalism (Harel, 1987)
  - **Graphical Formalism** to design complex systems
  - Key features:
    - Hierarchical
    - Interlevel transitions
    - Orthogonality
    - State Actions: Hooks “Entry”, “Exit”, “Throughout”
  - **Limitation:** No data flow specification

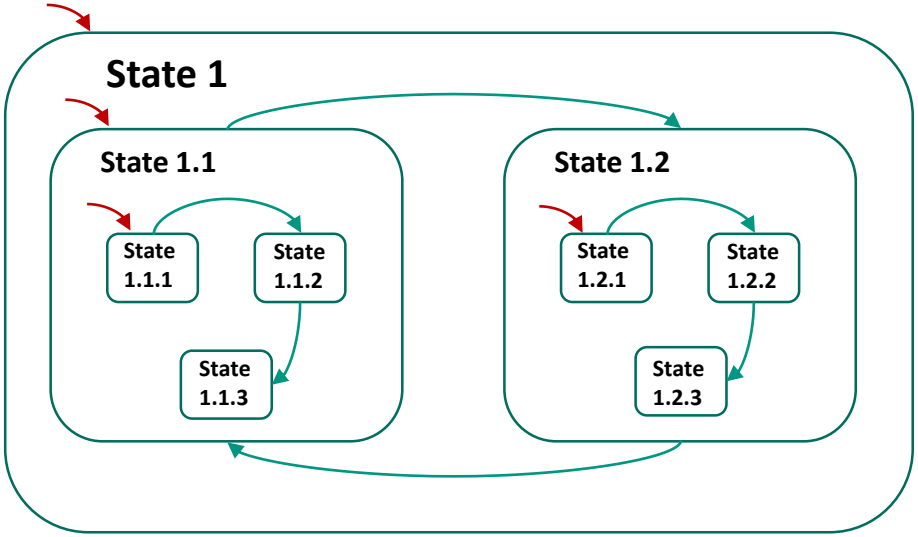


Harel,1987

- A, B, C und D are states
- Letters on the edges indicate events; conditions are specified in parantheses

D. Harel, *Statecharts: A visual formalism for complex systems*, Science of computer programming 8.3, pp. 231-274, 1987

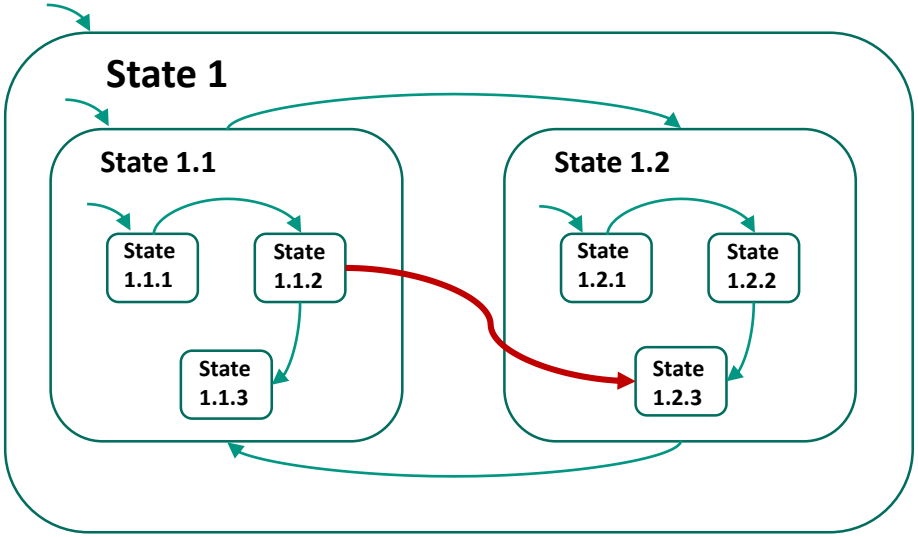
Statecharts (1)



Key Feature “Hierarchical”

- State 1 is top-level state
- State 1.1 and state 1.2 are substates of state 1
- State 1.1 and state 1.2 contain additional substates
- On entering a state, its **initial substate** will be entered

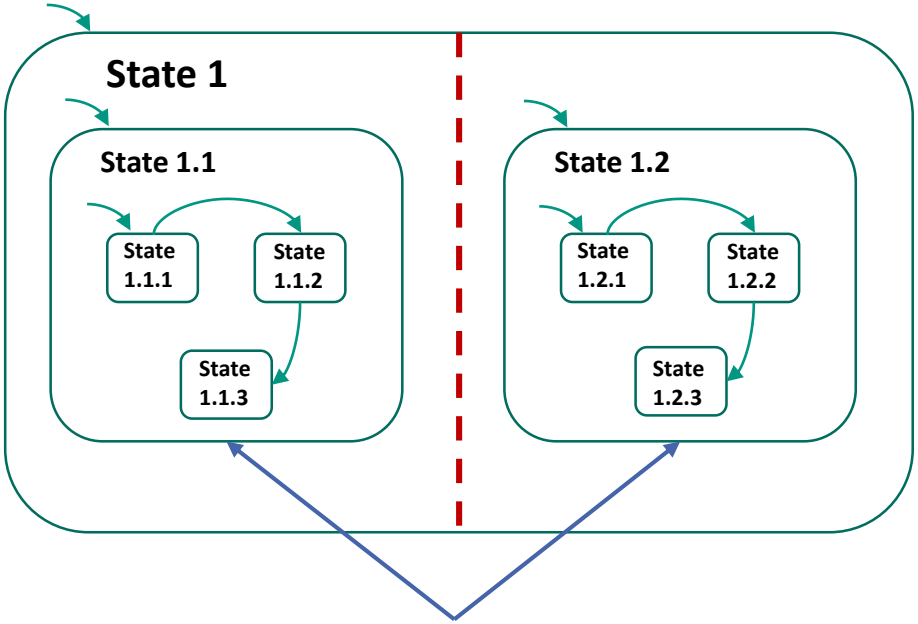
Statecharts (2)



Key Feature “Interlevel Transitions”

- Transitions can occur between hierarchy layers

Statecharts (3)

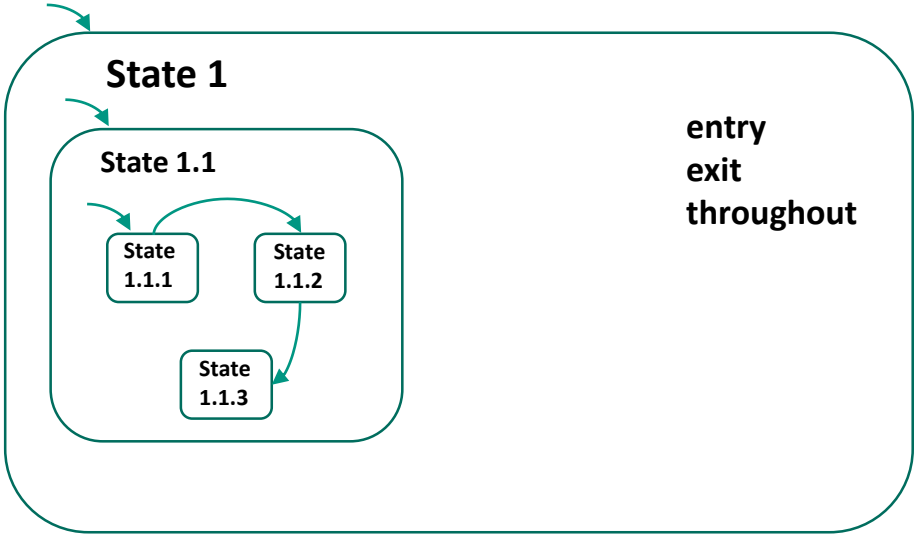


Parallel execution of both states

Key Feature “Orthogonality”

- A **dashed line** marks the **parallel execution** of states 1.1 and 1.2

Statecharts (4)



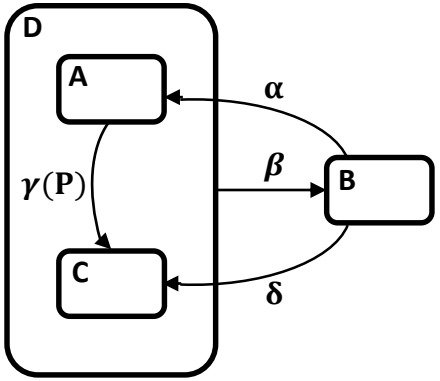
Key Feature “State Actions”

- On **entering** state 1, the **action *entry*** will be executed before entering state 1.1
- **While** state 1.1 is being executed, the **action *throughout*** will be executed
- After exiting state 1.1 and **before leaving** state 1, the **action *exit*** will be executed

Limitations of the Harel Statechart Formalism

- No **data flow specification**
  - Data flow is important for robots: results of states are needed in follow-up states
  - Example: Object localization is needed for visual servoing or inverse kinematics

- Reusability requires **adaptation of parameters**
  - Control parameters
  - Kinematics parameters
  - Object parameters
  - ….

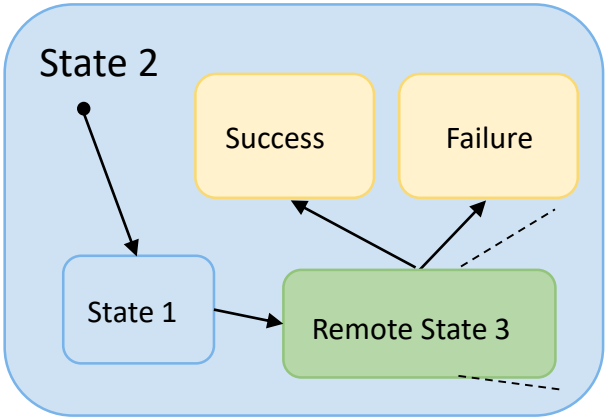


Harel,1987

Extension of the Harel Statechart Formalism at H²T

ArmarX Statechart extension: https://armarx.humanoids.kit.edu

- **Data flow specification**: Transition-based data flow
- **No Inter-level transitions**, since they hinder state reusability
- **Success and Failure states** in each state
- Dynamic structure
- Distributed over several hosts
- Connection of graphical control and data flow specification with C++ user code

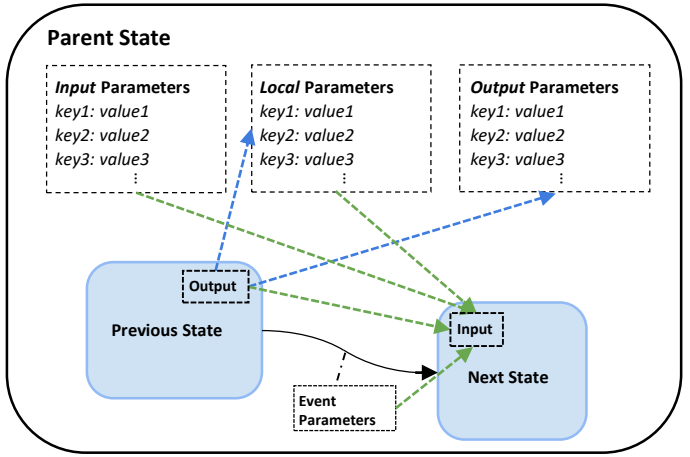


M. Wächter, S. Ottenhaus, M. Kröhnert, N. Vahrenkamp and T. Asfour, *The ArmarX Statechart Concept: Graphical Programming of Robot Behaviour*, Frontiers - Software Architectures for Humanoid Robotics, 2016

Statechart Extension

Transition-based data flow

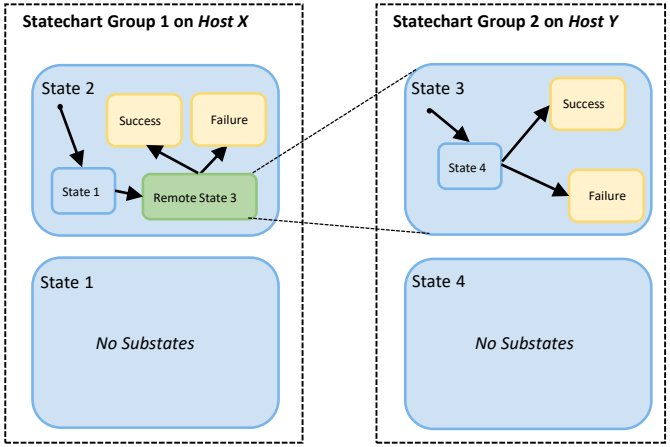
- Arbitrary data types
  - Elemental data types: int, float, string, …
  - Complex data types: Positions, Poses, Matrices, Lists, …
- Three parameter sets per state
  - **Input**, **Local**, **Output**
- Parameter mappings per transition
  - Of output parameters of the **source state** …
  - … to input parameters of the **target state**
- Specified data flow
  → No side effects through global variables



Parameter mappings

Statechart Extension (1)

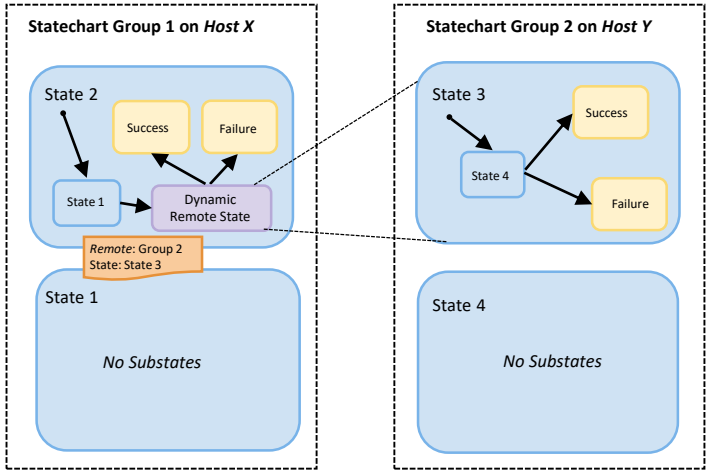
- **Distribution across several hosts**
  - Transparent distribution of Statecharts
    - Network middleware *ZeroC Ice (https://zeroc.com)*
  - Substates can be pointers to remote states (green states)
  - **Benefits:**
    - Load balancing
    - Increased robustness and error tolerance
    - Closer to the employed hardware (sensors or actuators)



Distributed Statecharts

Statechart Extension (2)

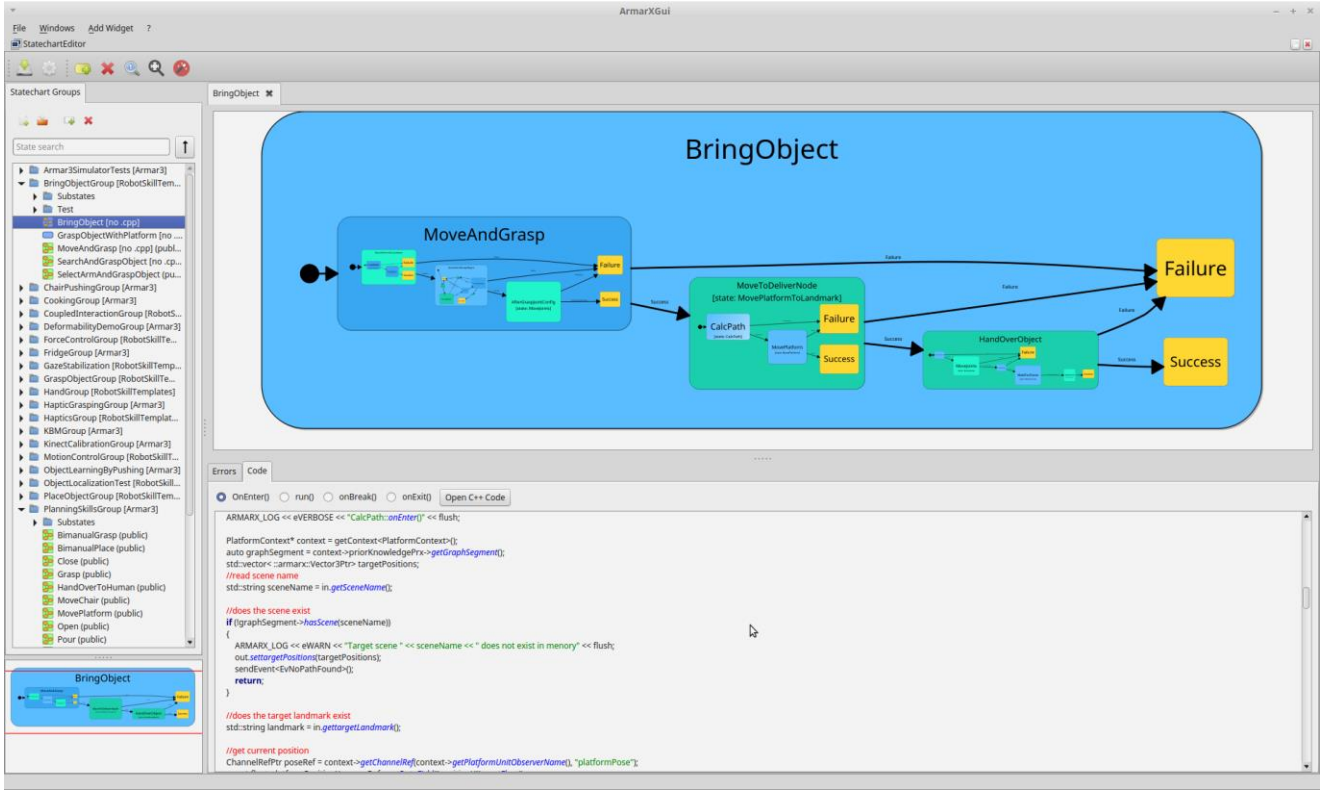
- **Dynamic Structure**
  - Exchange of substates during runtime
  - Transparent connection to arbitrary remote state
  - Substates specified through parameter mappings
  - Use-case: Execution of generated plans
- **Fully integrated into the robot development framework *ArmarX***
  - Graphical editor
  - Connection to all robot components
  - Online inspection of the current execution



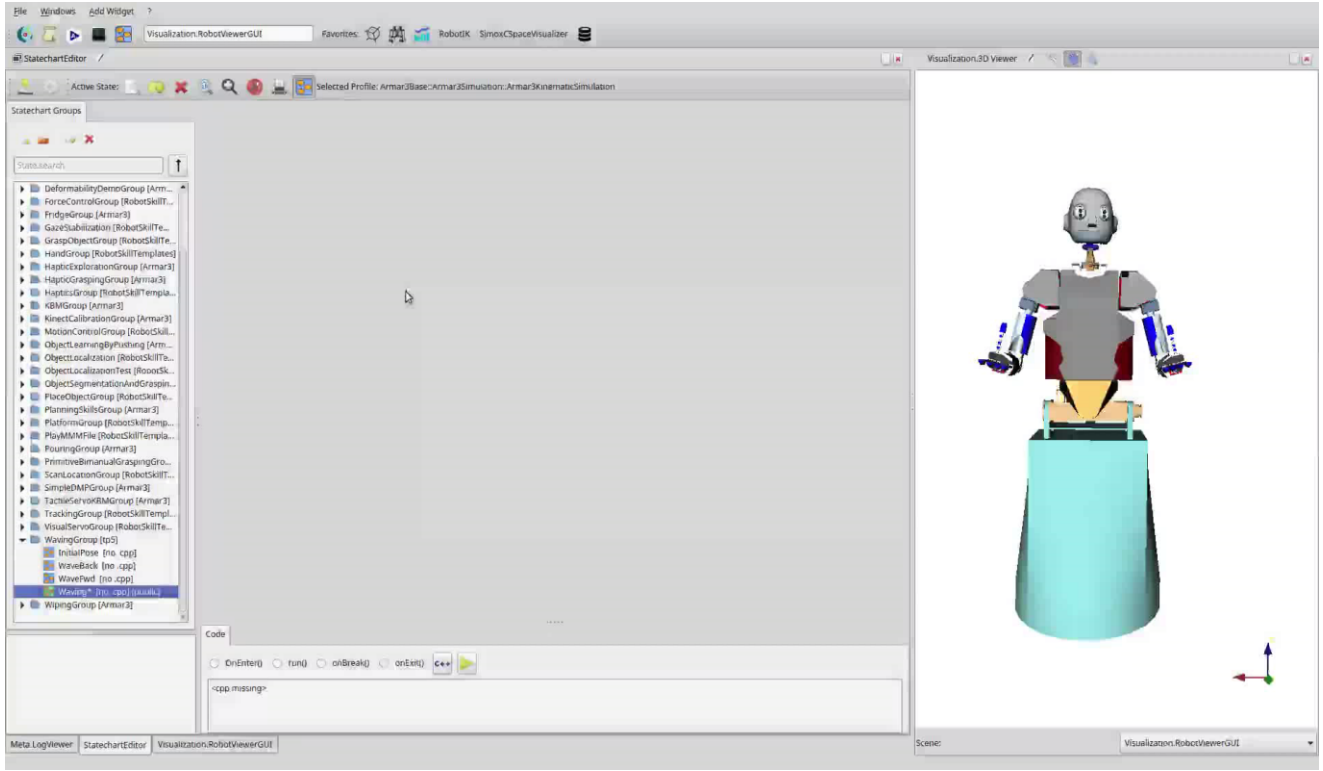
Graphical Statechart Editor in ArmarX

Tool for graphical specification of:

- Control flow
- Data flow
- Independent of existing robot components
- Directly linked with C++ code

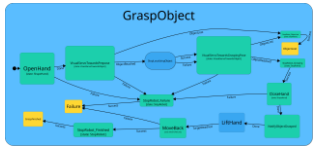


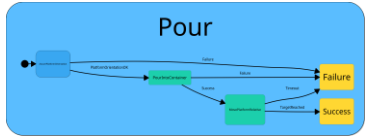
Graphical Statechart Editor in ArmarX: Example

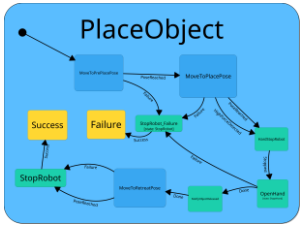


Manipulation Skills Via Statecharts

- ARMAR-III: Approx. 330 Statecharts implemented







Contents

- Motivation
- Classical robot programming (overview)
- Graphical robot programming (statecharts)
- **Programming by Demonstration**
  - Key questions of programming by demonstration
  - Capturing human demonstrations
  - Learning task models
  - Execution on a robot
- Planning

Literature

- More details in the lecture **Robotics II**
- Here: Basics

- Literature:

A. Billard, S. Calinon, R. Dillmann and S. Schaal, Robot Programming by Demonstration, In *Handbook of Robotics*, Springer, pp. 1371-1394, 2008.

B. Argall, S. Chernova, M. Veloso and B. Browning, *A survey of robot learning from demonstration*, Robotics and Autonomous Systems, 2009.

Basic Idea

Learning from **observation** of human

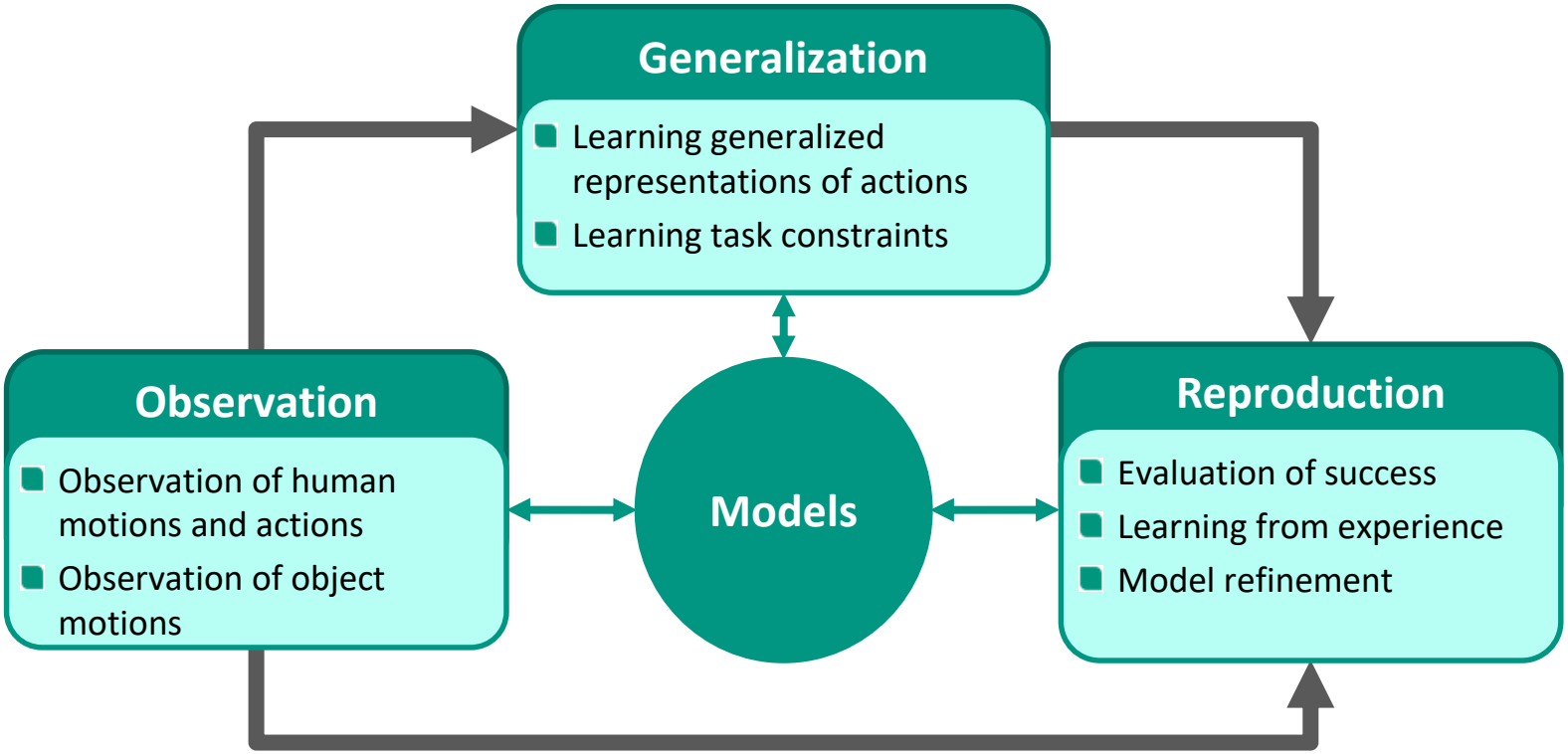
Programming by Demonstration (PbD)

- Main goal of programming by demonstration
  - **Intuitive** robot programming **without expert knowledge**
  - Learning of task models from human demonstrations

- PbD implies learning and generalization. Therefore, it is no playback approach.

- The research area is also known as
  - Learning from/by demonstration (LfD)
  - Learning from human demonstration (LfD)
  - Apprenticeship learning
  - Imitation learning (IL)
  - *Programmieren durch Vormachen* (*PdV*, German)

Learning From Human Observation



Three Reasons for PbD / Imitation Learning

- Powerful mechanism for **complexity reduction** of the search during learning (in contrast to brute force)
  - Good demonstrations can be chosen as starting point to learn a skill
  - Bad demonstrations can either be removed from the search space or taken as a negative example
- Implicit mechanism to **train a robot** that reduces or even completely eliminates explicit and tedious manual programming
- Understand coupling and learn relevant relations between **perception and action**

Challenges in PbD / Imitation Learning

1. Whom to imitate?

- Choosing a demonstrator whose behavior can benefit the imitator (**teacher selection**)
- This problem is usually avoided in most methods by defining the teacher

Challenges in PbD / Imitation Learning

2. When to imitate?

- The imitator has to **segment** and identify the beginning and end of a shown behavior
- The imitator has to decide if the observed behavior is **appropriate in the current context**, and also how many times this behavior should be repeated
- This problem is usually avoided in most methods by explicitly marking the starts and ends of demonstrations.

Challenges in PbD / Imitation Learning

3. What to imitate?

- How to determine **what aspects of a demonstration are of interest**? For some tasks it is about the movements (trajectories) of the human. In others, the results and effects of actions are important.
- Some observable **properties are irrelevant** and can be **ignored**. Examples: is it important that the teacher always approaches the table from the north?
- For continuous control tasks, this corresponds to determining the **feature space** for learning, as well as **constraints** and the **cost function**.
- For discrete control tasks, such as those treated by reinforcement learning and symbolic reasoning, this corresponds to **defining** the **state** and **action space**, and the automatic learning of **pre and post conditions (constraints)**.

Challenges in PbD / Imitation Learning

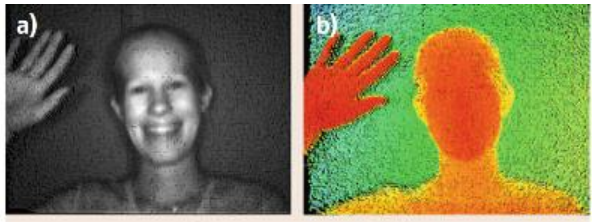
4. How to imitate?

- Determine how the robot will actually perform the learned behaviors to maximize the metric found when solving the “what to imitate” problem
  - A robot cannot act exactly in the same way as a human, due to differences in physical embodiment
  - Example: if the demonstrator uses a foot to move an object, is it acceptable for a wheeled robot to bump it, or should it use a gripper instead?
- The robot has to learn how to imitate **by mapping perception into a sequence of motion actions related to its own body**
  - Embodiment of the robot and its body constraints determine how observed action can be imitated (**correspondence problem**)

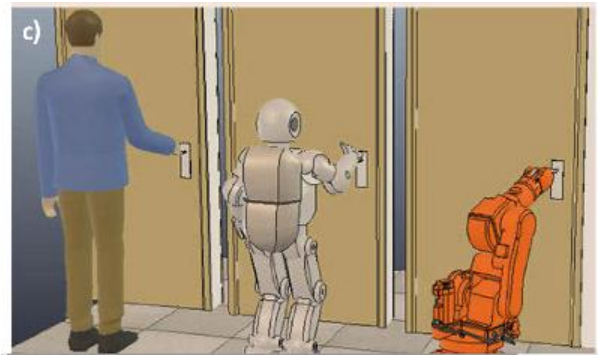
Perceptual and Physical (In-)Equivalence

Two different ways to determine, whether the state of a teacher (human) and the learner (robot) correlate.

- **Perceptual Equivalence:** Due to differences in the human's and the robot's **sensor capabilities**, the scene may appear entirely different. For example, a human perceives other humans through color and intensity, while a robot may use **depth information** in addition.



- **Physical Equivalence:** Due to different **embodiments**, humans and robots execute actions differently (or different actions) to achieve the same physical effects.



[Billard et al. 2008]

Challenges in PbD / Imitation Learning

- Who should be imitated?
- When should be imitated?

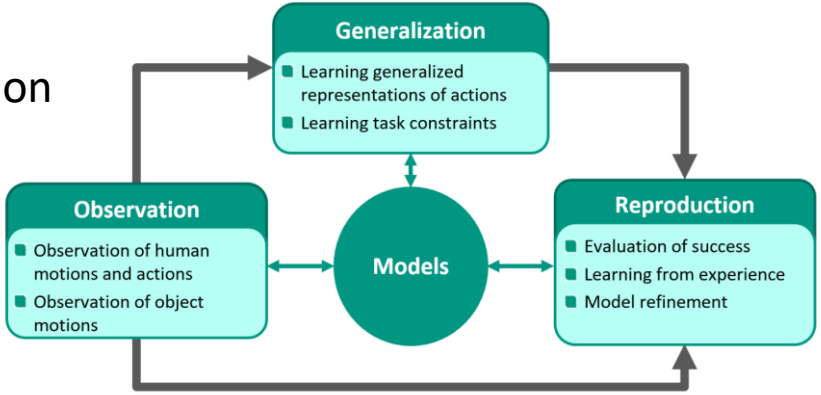
Mostly unresearched !

- What should be imitated?
- How should be imitated?

Learning of a skill or a task!

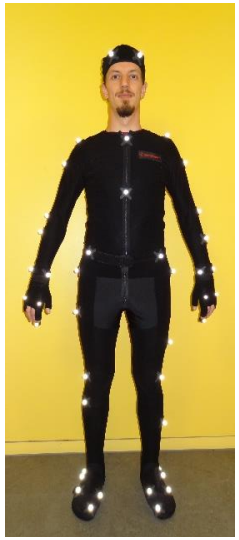
Contents

- Motivation
- Classical robot programming (overview)
- Graphical robot programming (statecharts)
- **Programming by Demonstration**
  - Key questions of programming by demonstration
  - Capturing human demonstrations
  - Learning task models
  - Execution on a robot
- Planning



Capturing Human Movements

Capturing human movements is the starting point for programming by demonstration



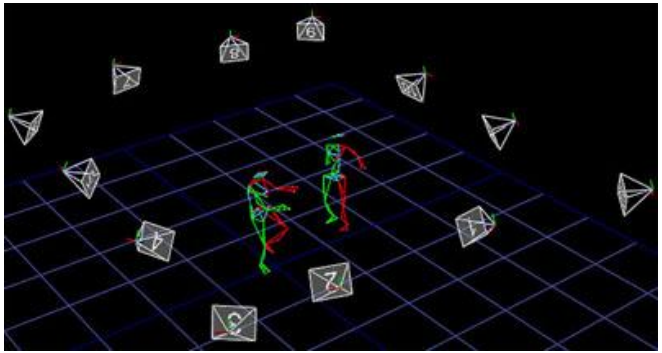
Two Approaches

- **Marker-based Approaches:**
  - Marker are attached to predefined anatomic landmarks on the human body
  - Marker may be **active** (e.g., coding via LED) or **passive** (e.g., reflecting)
  - Subject must be prepared for capturing
- **Markerless Approaches:**
  - Direct reconstruction of the human pose from camera data (RGB / depth data)
  - No markers on the human required

Marker-based Passive Optical Motion Capturing

- Localization of infrared-reflecting passive markers using a **multi-camera system**
- Widely used solution, various commercial providers (e.g., **VICON** and **OptiTrack**)

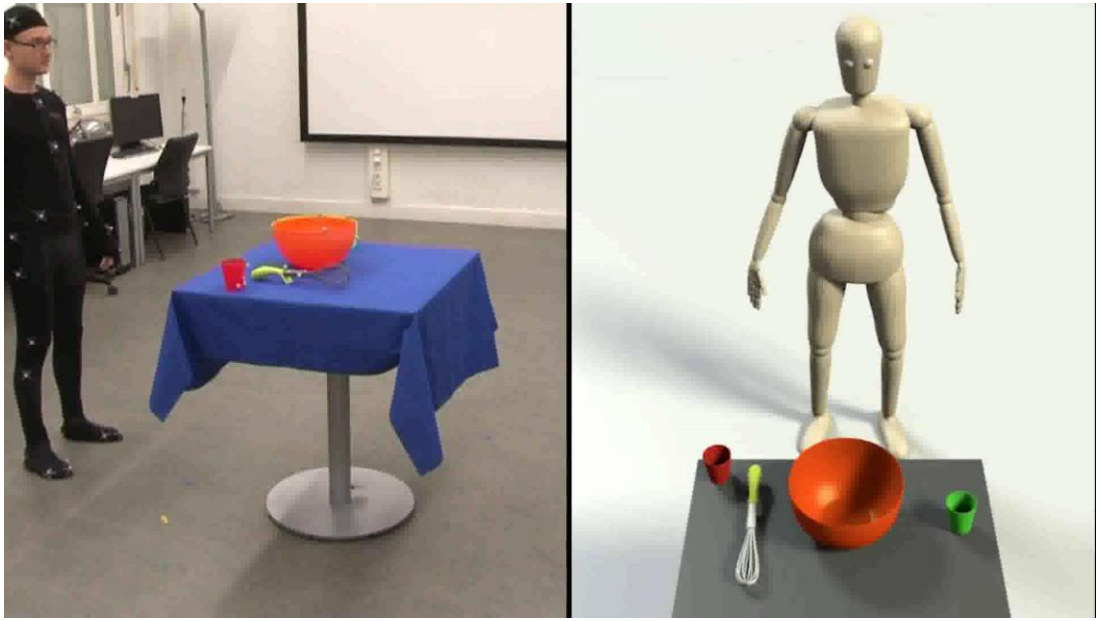
- **Advantages**:
  - High spatial resolution (submillimeter range)
  - High temporal resolution (1-2 kHz possible)



- **Disadvantages**:
  - Expensive and large technical effort
  - Efforts for required post-processing because of occlusions and labelling markers
  - Restrictions to the capturing area (e.g., only feasible indoors)

Example

- KIT Whole-Body Human Motion Database



https://motion-database.humanoids.kit.edu

Marker-based Active Optical Motion Capturing

- Marker transmits a coded identifier, e.g., via infrared LED

- **Advantages/Disadvantages** similar to optic-passive systems, but:
  - Easier handling of marker labeling
  - However: Batteries/cables required to power each marker

© Qualisys

IMU-based Motion Capture

- **Inertia Measurement Units (IMUs)** attached to anatomically defined landmarks on human subjects
  (e.g., 17 sensors with Xsens MVN)

- **Advantages**:
  - Less constrainted environment (e.g., enabling outdoor use)
  - Accurate capturing of the human pose

- **Disadvantages**:
  - Exact placement of IMUs required
  - No precise localization of human in the global reference frame (IMU drift)

© Xsens

Mechanical Motion Capturing

- Direct measurement of joint angles (e.g., potentiometers in exoskeletons)

- **Advantages**:
  - Low requirements for the recording environment (e.g., outdoors)

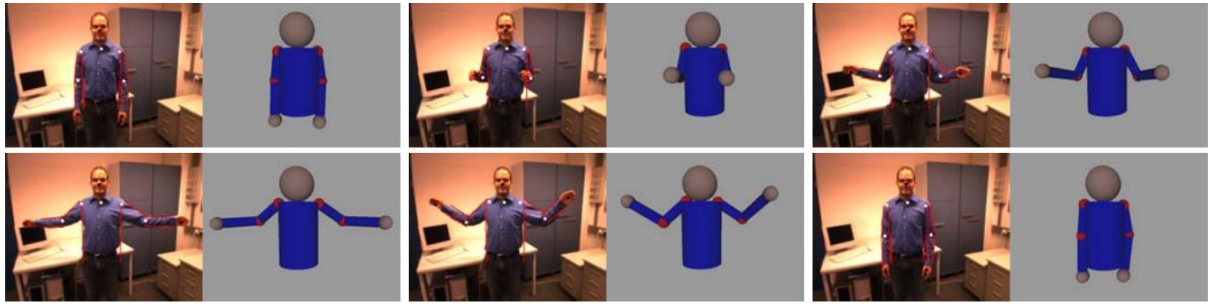
- **Disadvantages**:
  - Exoskeleton constraints human movements → See lecture “Wearable Robotic Technologies“ in summer term
  - High technical efforts
  - No localization of human in global reference frame possible

Alexander Gmiterko, Tomáš Lipták, Motion Capture of Human for Interaction with Service Robot, *American Journal of Mechanical Engineering*. 2013, 1(7), 212-216 doi:10.12691/ajme-1-7-12
http://pubs.sciepub.com/ajme/1/7/12/index.html

https://metamotion.com

Marker-less Optic Motion Capturing

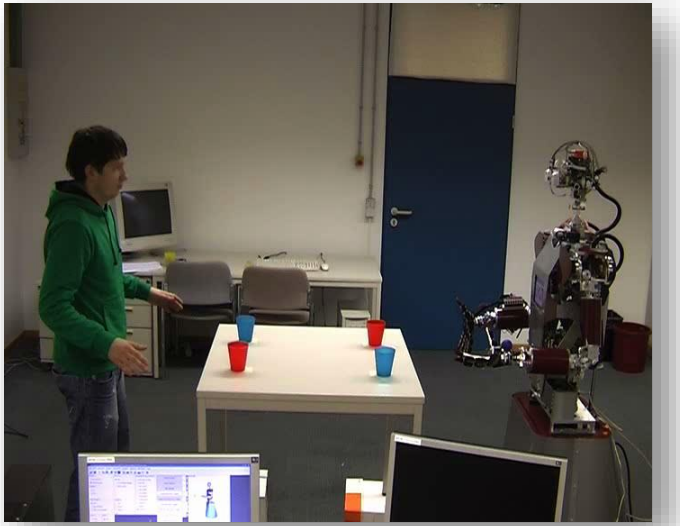
- Reconstruction of human pose from **RGB- and/or depth data**
- **Advantages**:
  - Very cheap, low hardware requirements
  - Low requirements for capturing area
- **Disadvantages**:
  - Complex algorithms and high error rate (active area of research)
  - Low temporal resolution, especially online



Azad et al. (2008)

Marker-less Optic Motion Capturing

- Stereo vision



Azad et al. (2008)

Marker-less Optic Motion Capturing

- Skeleton tracking with depth cameras
- Skeleton data are calculated from single frames
- Accuracy between RGB tracking and marker-based tracking

Kinect (RGB-D)

Captured depth image with background removal

Inferred body parts per pixel

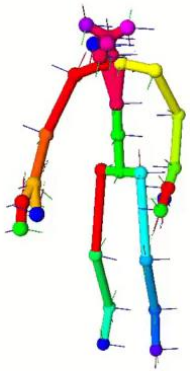
Estimated body joint positions

Fitted skeleton model

J. Shotton et al., *Real-Time Human Pose Recognition in Parts from a Single Depth Image, CVPR, 2011*

Deep-Learning-based Approaches

- Azure Kinect body tracking (in real-time) https://learn.microsoft.com/en-us/azure/kinect-dk/body-sdk-download
  - Deep Neural Network for human pose estimation
  - Nonlinear optimization for motion retargeting to MMM model



RGB | Depth | Human pose | Motion retargeting

Motion Capture Studio at H2T

14 Vicon Motion Capture Cameras

3 Blue Trident IMU Sensors

Force Torque Sensors

Sensorized Exoskeleton

3 Azure Kinect RGB-D Cameras

3 Kinect V2 RGB-D Cameras

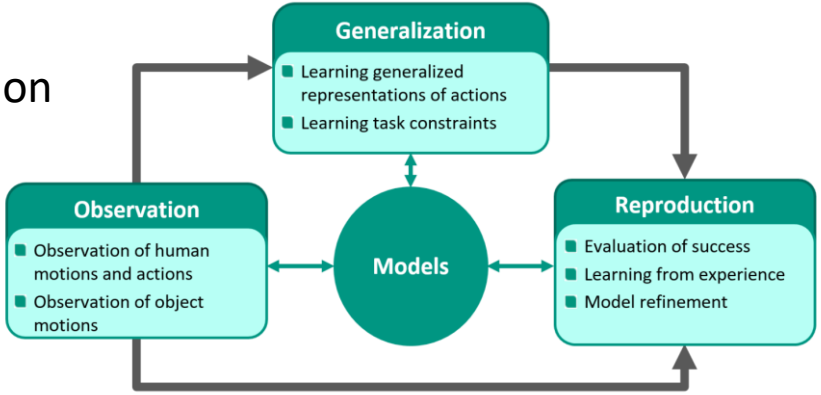
Microphone Array

GoPro Hero 8

Pair of CyberGlove II

Contents

- Motivation
- Classical robot programming (overview)
- Graphical robot programming (statecharts)
- **Programming by Demonstration**
  - Key questions of programming by demonstration
  - Capturing human demonstrations
  - Learning task models
  - Execution on a robot
- Planning

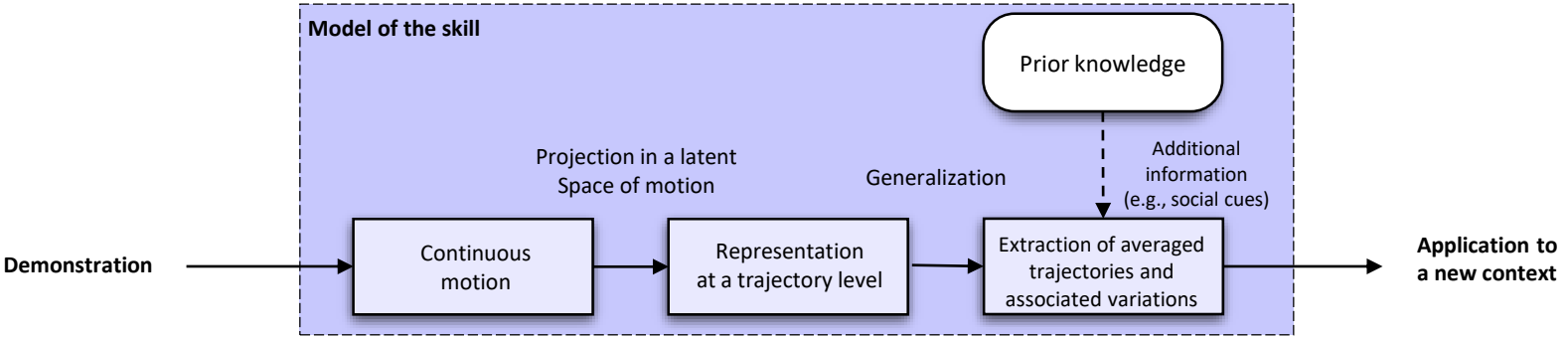


Important Questions

- **Learning on two levels**
  - **Subsymbolic or sensomotoric** (trajectory level)
    → Learning of a skill
  - **Symbolic or semantic** (symbolic level)
    → Learning of task models

PbD – Learning a Skill

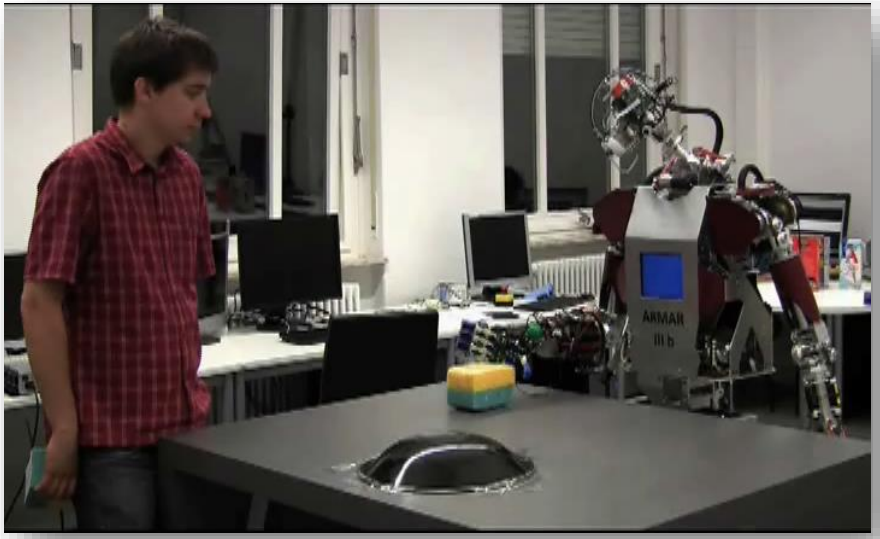
- **Subsymbolic or sensomotoric** (trajectory level)
  Learning a non-linear mapping between sensor and motor information. This is a representation on the lowest level.
- **Generalization on trajectory level**

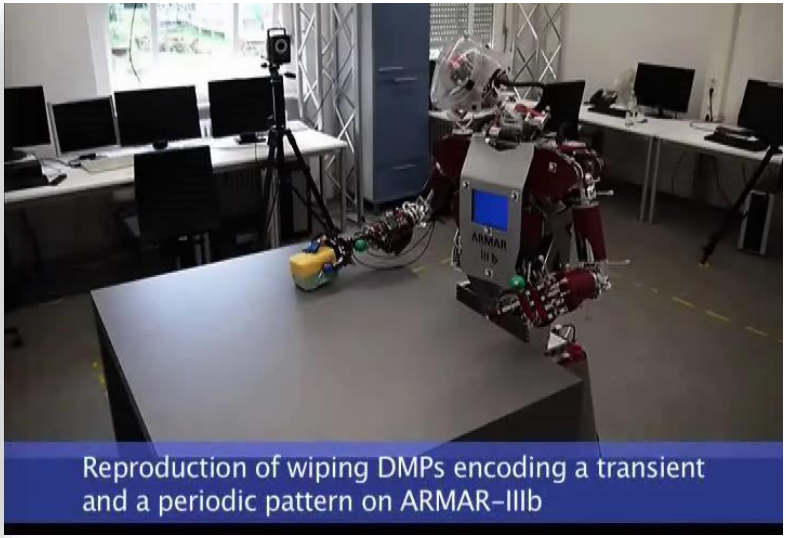


Billard, A., Calinon, S. and Dillmann, R. (2016). Learning From Humans. Siciliano, B. and Khatib, O. (eds.). Handbook of Robotics, 2nd Edition, Chapter 74, pp. 1995-2014. Springer

Example: Trajectory Level

- Learning a wiping motion



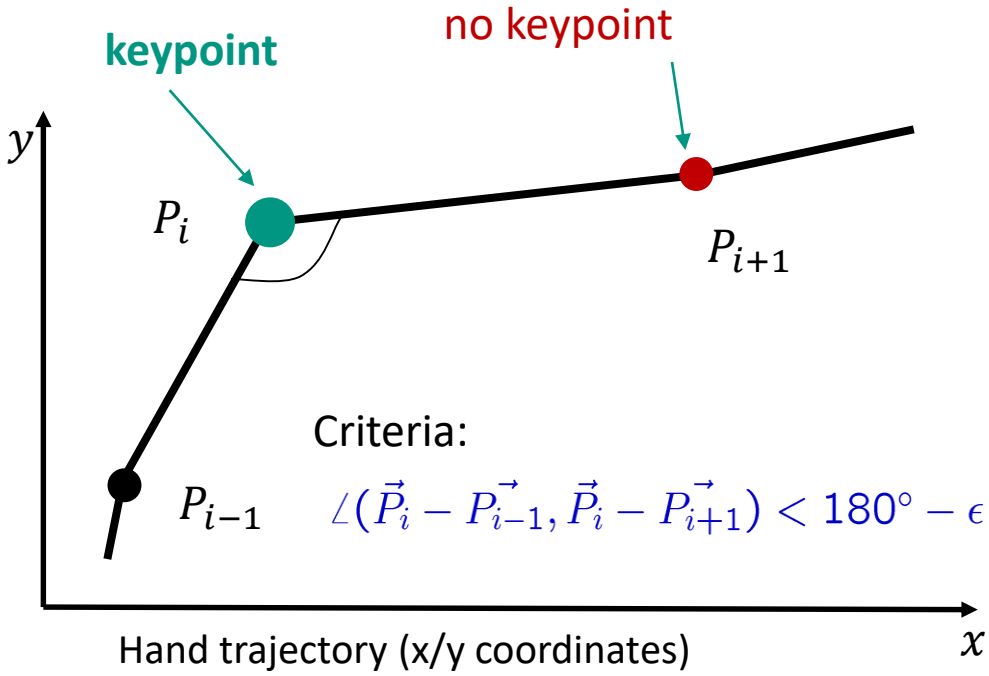


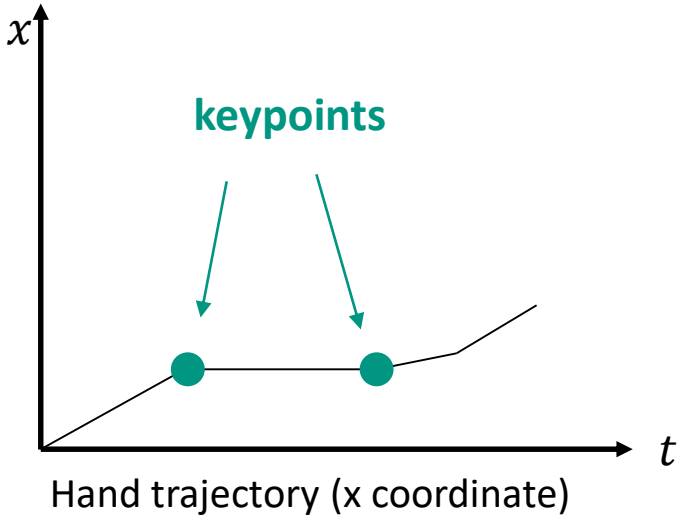
Segmenting a Demonstration

- **Motion segmentation:** Partition of movement trajectories into simple representable parts
- For this, key points of a demonstration need to be identified
- Criteria for segmentation required
  - In the configuration space and in task space
  - Different criteria: Changes in movement, search for specific patterns

Criteria for Segmentation

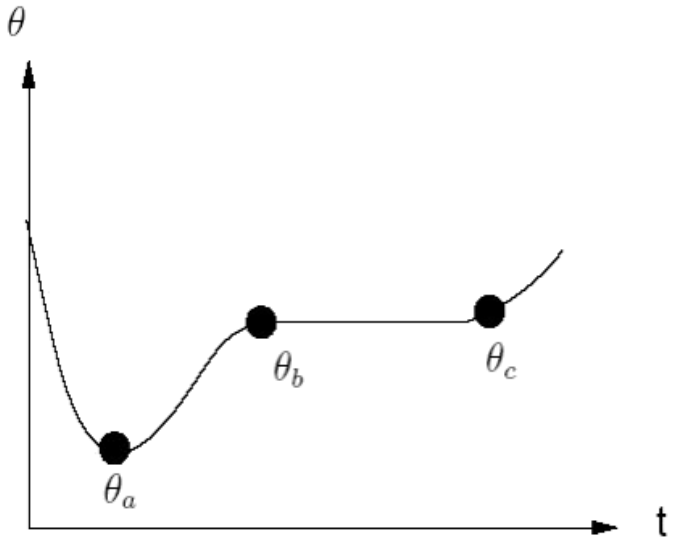
- Local minima and maxima and pauses in task space





Criteria for Segmentation

- Local minima and maxima and pauses in configuration space



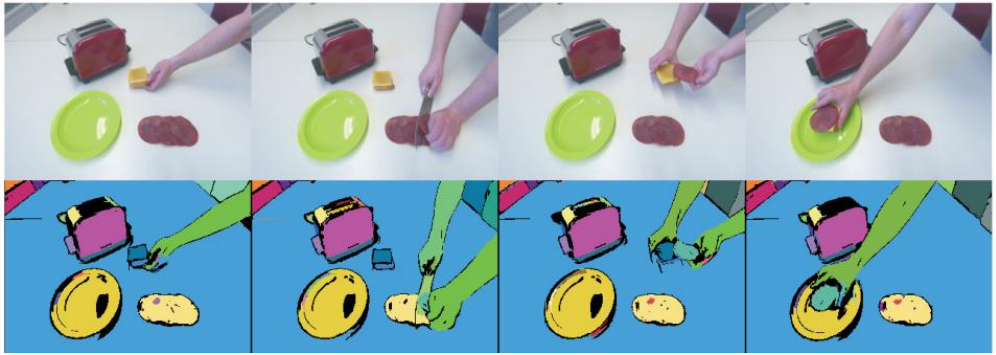
Joint trajectory

Example: Learning a Trajectory from RGB-D Data

- Tracking of teacher and objects via color and depth regions
- Optical flow (movement patterns) are used, to track the movement between 2 frames
- Segment centers per frame represent the trajectory

Camera images

Tracked
Image regions



A. Abramov, E.E. Aksoy, J. Dörr, K. Pauwels, F. Wörgötter, *Real-Time Human Pose Recognition in Parts from a Single Depth Image, 3DPVT, 2010*

PbD – Learning a Skill

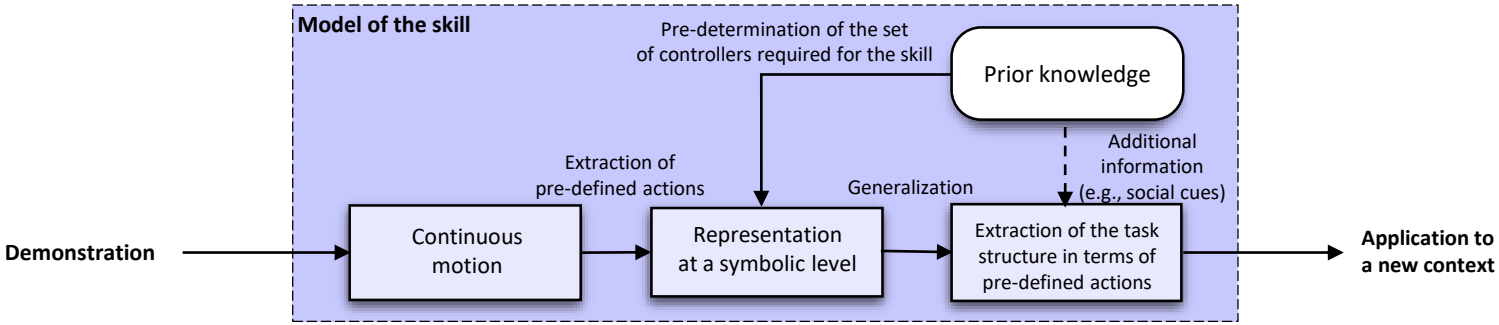
- Learning of action and representation often happens jointly
  - Representation is strongly coupled to the learning approach
- Algorithms for motion segmentation: HMM, PCA, Clustering, Template Matching, Classification
- Methods to learn a skill
  - Hidden Markov Models (HMM)
  - Dynamic Movement Primitives (DMP)
  - Gaussian Mixture Models (GMM)
- Refining learned trajectories
  - Reinforcement Learning
- **More in the lecture “Robotics II – Humanoid Robotics” in summer term**

PbD – Learning a Skill

- **Subsymbolic:**
  - Segmenting a motion trajectory, i.e., identifying keypoints of the demonstration
  - Describes the resulting segments in a generalized form (function approximation)
- **Advantage:** Efficient learning of a motion
- **Disadvantage:** Semantic information not considered, e.g., pre and postconditions

PbD – Learning a Skill

- **Semantic or symbolic:**
  Learning a sequence of actions that represents the demonstration.
  This is the representation on the highest level.
- **Generalization on symbolic (task) level**



Billard, A., Calinon, S. and Dillmann, R. (2016). Learning From Humans. Siciliano, B. and Khatib, O. (eds.). Handbook of Robotics, 2nd Edition, Chapter 74, pp. 1995-2014. Springer

Summary – Learning on Two Levels

Representation	Generalization	Advantage	Disadvantage
Trajectory	Generalization of motions	General representation of motions that allow encoding different kinds of signals or functions	Complex skills cannot be reproduced
Symbolic	Organization of predefined motion elements	Enables learning of hierarchical task descriptions	Needs a set of predefined controllers for reproduction

Example: Task Level

- Learning a sequence of actions, i.e., a plan (prepare dough)

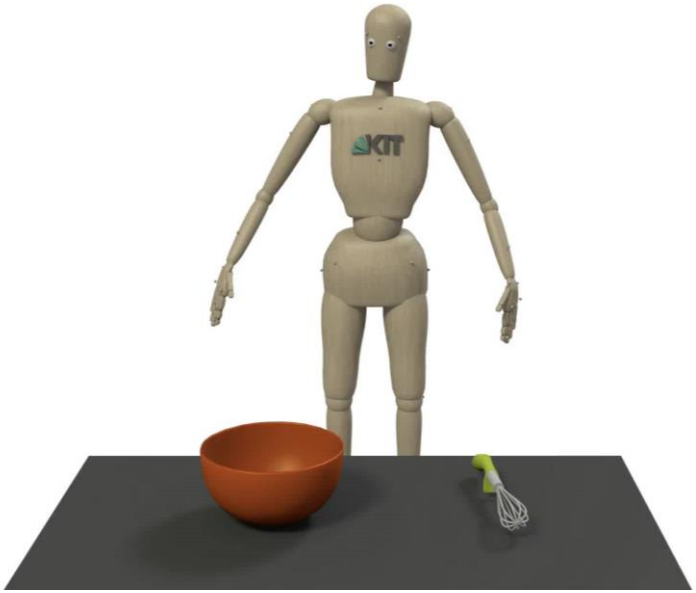
PbD – Learning Task Models – Approach

- Observing the teacher in the environment
- Extracting semantic features: States, object-object relations, object-hand relations, rules
- The robot needs to estimate a suitable action based on the **current world state**
- The world state must be determined from **observations of the robot**
- **Goal:** A **mapping** between world state and action → Strategy for action selection

Task Segmentation (1)

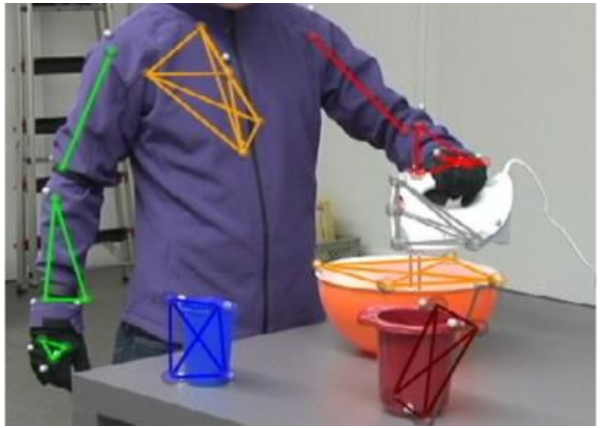
- Demonstrations mostly consist of action sequences
- **Understanding** demonstrations is **easier**, if they are separated into **smaller segments** or even **known actions**

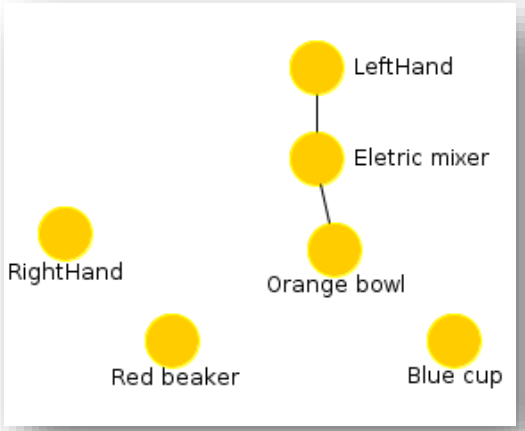
- But:
  - Which actions occur and how many actions are unknown?
  - Start and end of actions is unknown
  - Actions blend into each other seamlessly



Task Segmentation (2)

- **Contact relations between objects** to represent world state
- **Keyframes** at **changes** of the contact relations
- Each segment represents a **manipulation action**
  - *Mixing dough* for example contains *grasping*, *mixing*, *placing*, …





Task Segmentation (3)

- **Object relations** enable the extraction of preconditions and effects (postconditions) of actions for **symbolic planners**

- **Example:**
  - State at segment start: *LeftHand touches nothing*
  - State at segment end: *LeftHand touches RedCup*

  → *Precondition:* *empty(LeftHand)*

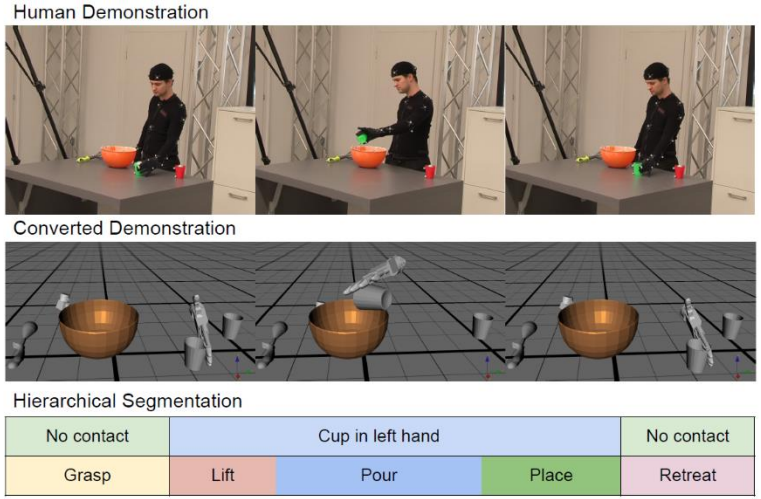
  → *Postcondition:* *in(RedCup,LeftHand) ∧ ¬empty(LeftHand)*

Hierarchical Task Segmentation

- **Hierarchical Task Segmentation** considering movements and relevant objects

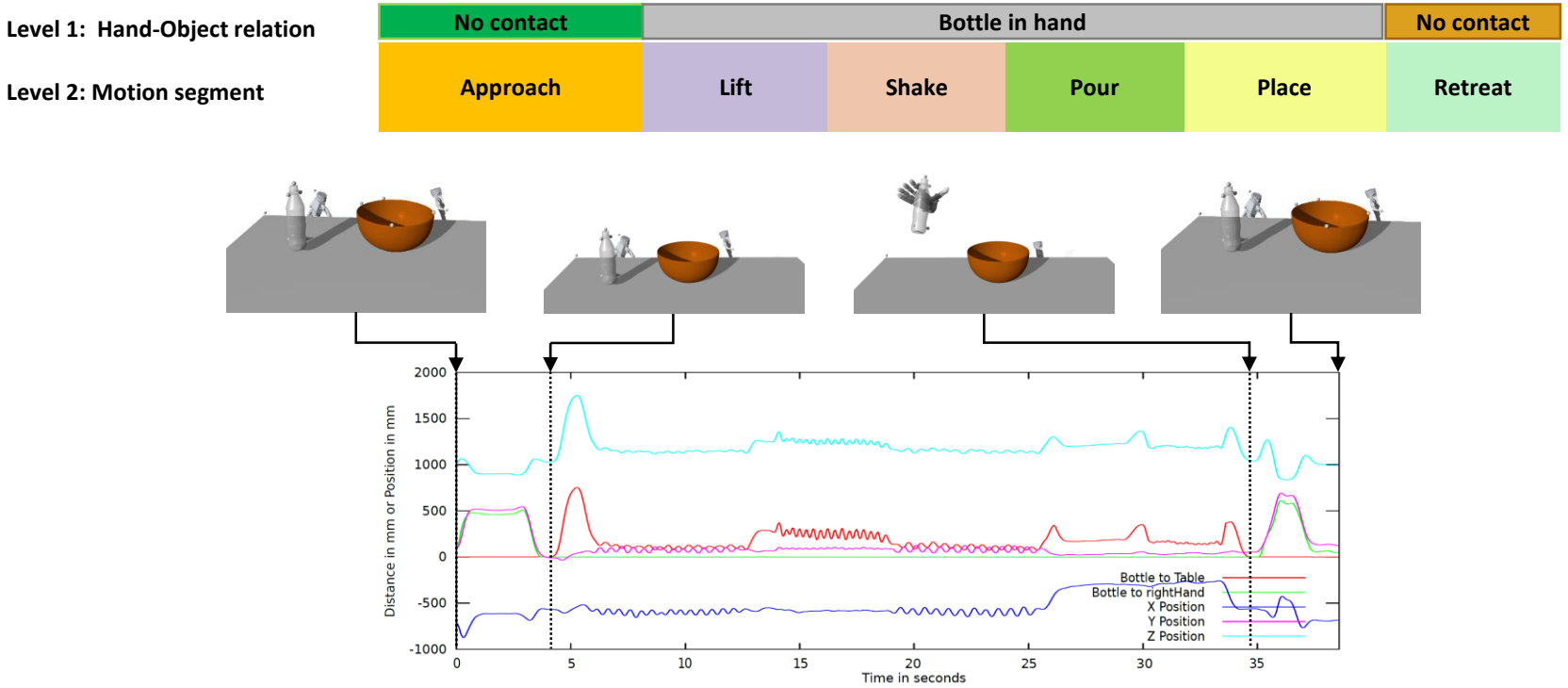
  Level 1: Semantic segmentation based on **contact relations between objects**

  Level 2: Motion segmentation based on **characteristics** of trajectories (motion dynamics)

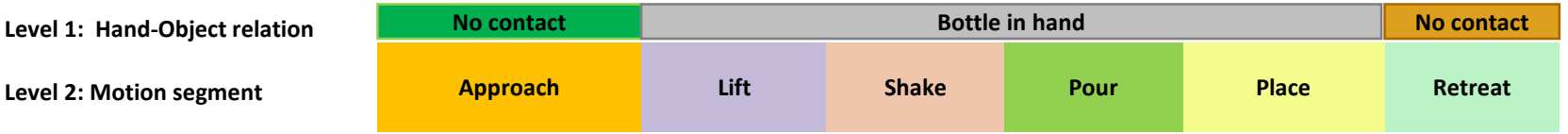


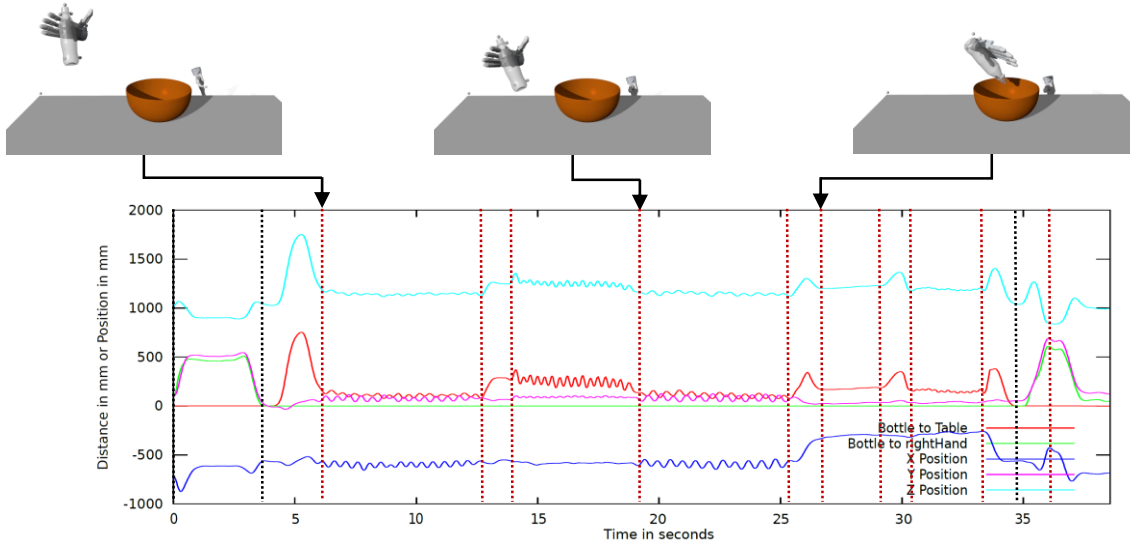
M. Wächter and T. Asfour, Hierarchical Segmentation of Manipulation Actions based on Object Relations and Motion Characteristics, International Conference on Advanced Robotics (ICAR), July, 2015

Level 1: Semantic Segmentation

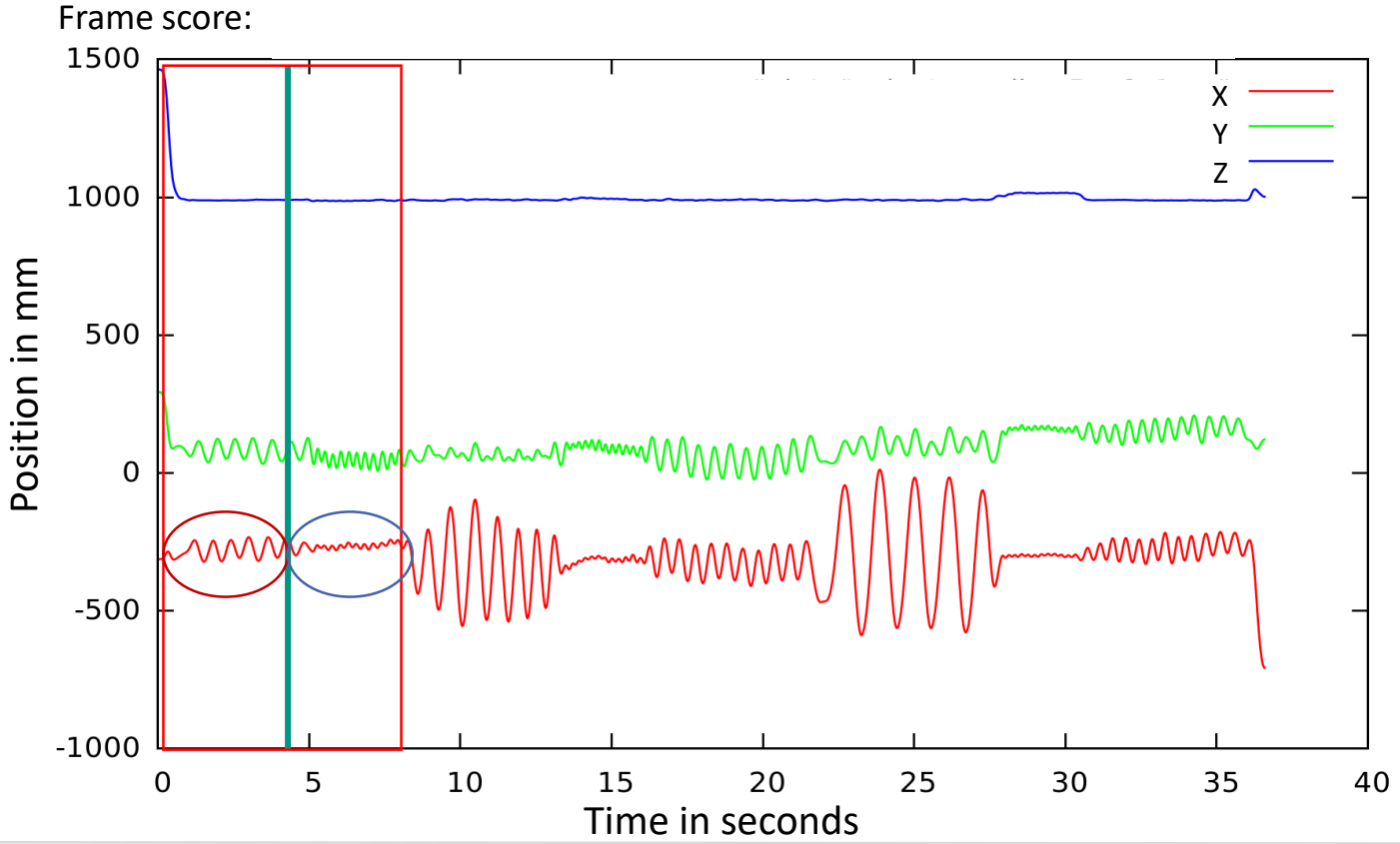


Level 2: Motion Segmentation

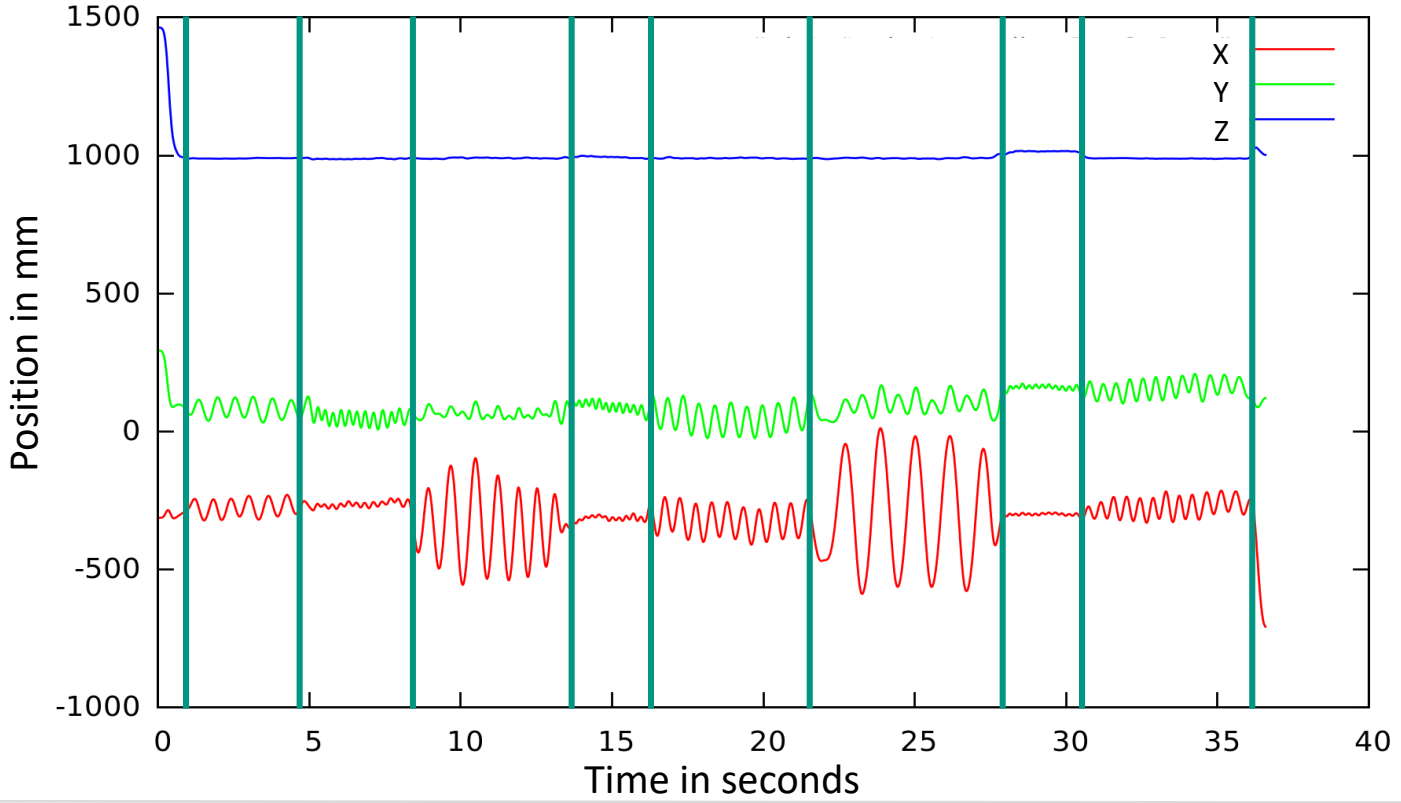




Motion Characteristic Heuristic: Sliding Window



Motion Characteristic Heuristic: Recursive Search

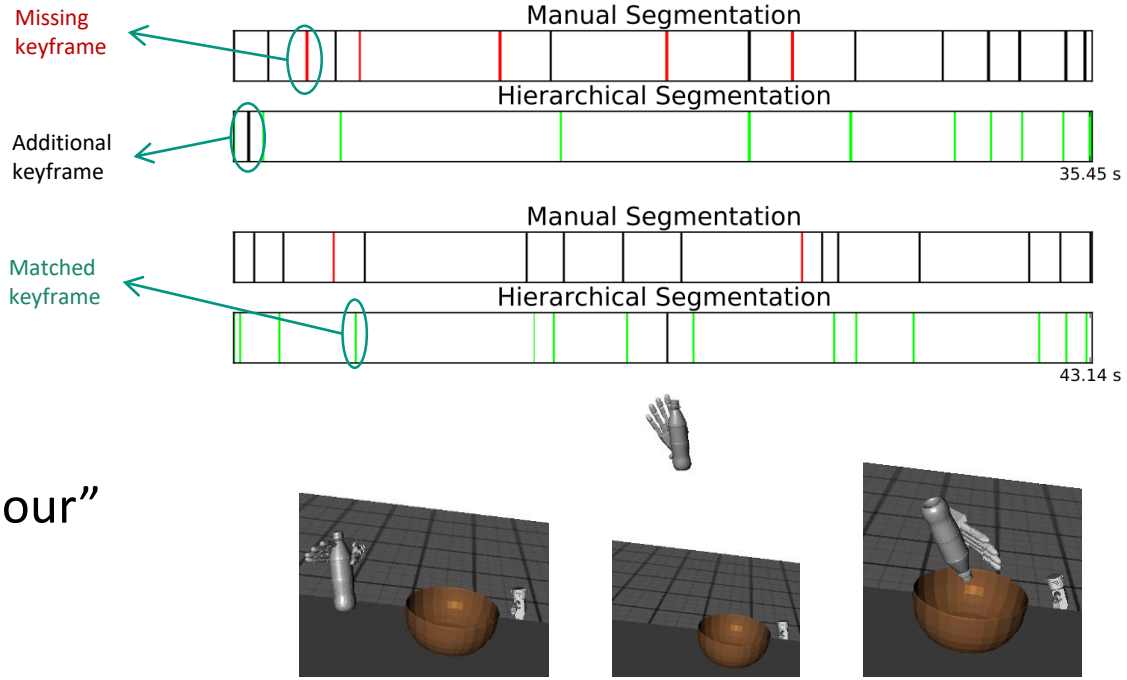


Segmentation Quality Evaluation

- Metric for segmentation results: **Penalize** missing or additional keyframes

$$e = \underbrace{(m + f) * p}_{\text{Penalty for missing and additional keyframes}} + \underbrace{\sum_{i} \min_{j}(k_{r,i} - k_{f,j})^2}_{\text{Mean squared error to the next keyframe}}$$

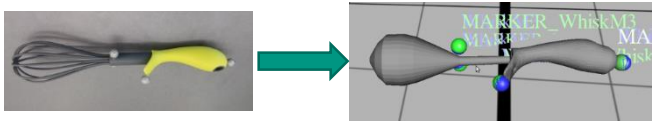
Evaluation: Comparison to Reference Segmentation



2 repetitions of “shake and pour”

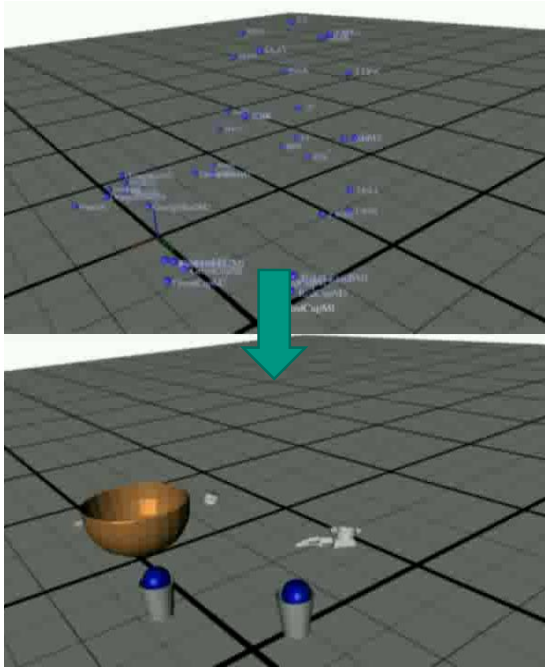
Marker-based Capturing of Demonstrations

- Marker-based capturing of demonstrations from
  - Human (teacher)
  - Objects
  - Environment



- Transformation to trajectories of 6D poses using 3D models with virtual markers

- Reference model of the human body using the Master Motor Map (MMM) data format[1]



[1] https://mmm.humanoids.kit.edu/

Contents

- Motivation
- Classical robot programming (overview)
- Graphical robot programming (statecharts)
- **Programming by Demonstration**
  - Key questions of programming by demonstration
  - Capturing human demonstrations
  - Learning task models
  - Execution on a robot
- Planning

Execution on a Robot

- Mapping the learned knowledge (actions, action sequences, …) to a robot
- **Solving the correspondence problem**: Humans and robots have different kinematics and dynamics
- Resulting motions serve as setpoints for controllers
- Execution with online adaptation and dynamic environments

Summary

- PbD allows learning skills from human teachers and mapping the skills to arbitrary robots
- PbD can facilitate the learning process (compared to Reinforcement Learning or trial-and-error methods)
- The robot needs to provide a library of generic scills and should be able to adapt them for different contexts

Research Questions

- **Learning type**
  - **Batch learning:** The action is learned after all demonstrations/examples have been captured
  - **Incremental learning:** The action representation is continuously updated/learned after each new demonstration

- **Number und type of demonstration/example**
  - Learning from **many** demonstrations
  - Learning from **few** demonstrations or a **single** demonstration
  - Learning from positive or negative demonstrations/examples

- **Interaction with the human during learning**
  - **Natural language:** Verbal commands (*here*, *there*, *now*, *faster*, ..), clarifying dialogs to complete knowledge over the task, feedback, …
  - Gaze direction and pointing gestures of the teacher
  - Incorporating additional modalities, such as haptics, audio, …

More on the topic *Programming by Demonstration* in the **Lecture “Robotics II – Humanoid Robotics“** in the summer term

Contents

- Motivation
- Classical robot programming (overview)
- Graphical robot programming (statecharts)
- Programming by Demonstration
- **Planning**

Symbolic Planning

- A problem defines a **state space Θ**:
  - S, finite set of states (consisting of symbols)
  - A, finite set of actions (consisting of name, preconditions, effects)
  - $c: A \rightarrow \mathbb{R}_0^+$, cost function
  - $I \in S$, initial state
  - S^G set of goal states

- Classical assumptions:
  - Finite number of states (discrete)
  - Single agent (no other agents or opponents)
  - Fully observable (agent knows and perceives everything)
  - Deterministic (every action has exactly one following state)
  - Static (no dynamic state changes)

Symbolic Planning – Solving a Problem

- Definition of „Plan“
  - Oxford Leaner’s Dictionaries: plan, *noun*
    - *Intention: something that you intend to do or achieve*
    - *Arrangement: a set of things to do in order to achieve something, especially one that has been considered in detail in advance*
    - *Further meanings: map, drawing, a way to invest money*

- Technical view:
  - **Sequence** of **parametrized actions** to achieve a **predefined goal**
  - The shortest plan is called **optimal**

- Planning:
  - Finding an (optimal) plan for a problem

https://www.oxfordlearnersdictionaries.com/definition/english/plan_1, accessed February 05, 2025

STRIPS

- **„ST**anford **R**esearch **I**nstitute **P**roblem **S**olver“ (Fikes, 1971)
- One of the first, and most widely known, languages to describe planning problems. Developed in 1971 by Fikes and Nilson to control the robot „Shakey“

- Very simple, thus also limited

- Derived from STRIPS:
  - *Action Description Language* (ADL)
  - *Planning Domain Definition Language* (PDDL)

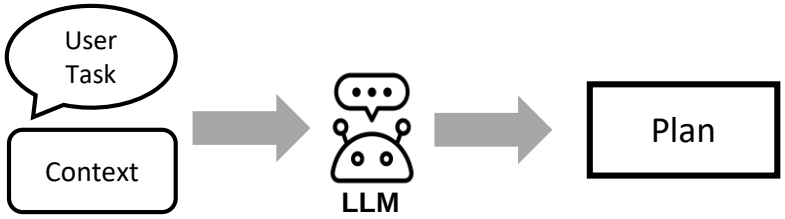
- **Planning domain** in STRIPS:
  - States
  - Actions, all having the same costs: $c(a) = 1, \forall\, a \in A$
  - Goals

R. Fikes and N. Nilsson (1971). STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2:189-208.

Large Language Models for Task Planning

LLM As Planner

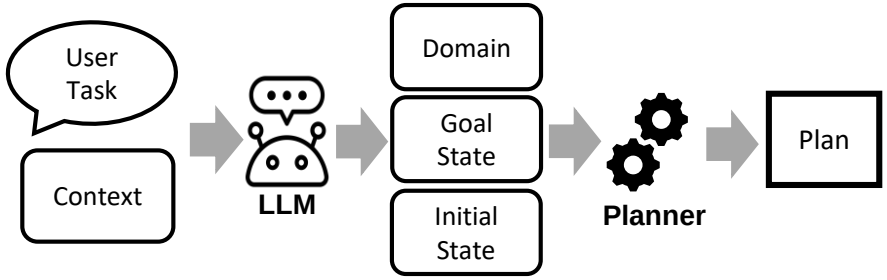
(SayCan, Code As Policies, PaLM-E, Progprompt)



+ Dynamic handling of changing environment and reactions to errors
− Plans are not always optimal
− The LLM can cause the robot to perform unsafe actions

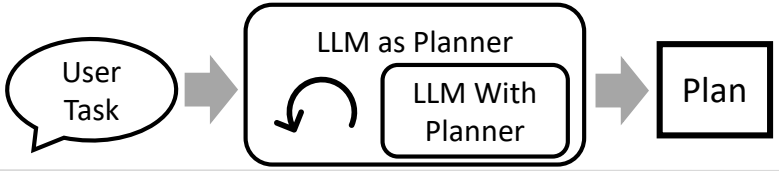
LLM With Planner

(LLM+P, AutoTAMP, Delta, COWP)

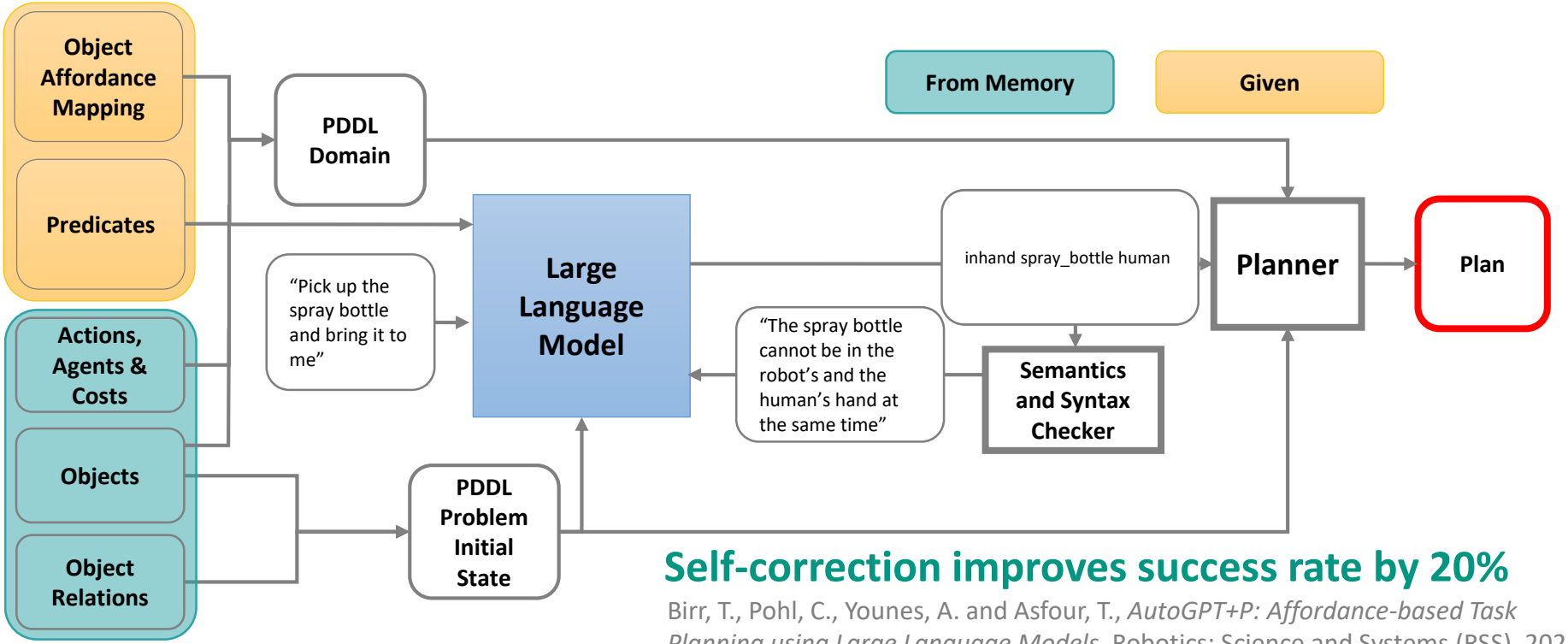


− Fails if robot has not detected all objects
+ Plans are always optimal
+ Definition and strict following of rules increases safety

AutoGPT+P



AutoGPT+P: LLM+P with Self-Correction



Self-correction improves success rate by 20%

Birr, T., Pohl, C., Younes, A. and Asfour, T., *AutoGPT+P: Affordance-based Task Planning using Large Language Models*, Robotics: Science and Systems (RSS), 2024