

# Constraint-based robot programming for advanced sensor-based applications and human-robot interaction

COBAROP – Bringing constraint-based robot programming to real-world applications

Workshop at IEEE/RSJ International conference on Robot and Systems (IROS 2020)

dr. ir. Erwin Aertbeliën,

Research Expert, Dep. of Mech. Eng., KU Leuven, core Lab of Flanders Make

Erwin.Aertbelien@kuleuven.be



# Motivation

# Introduction

## From cages to human environments



KUKA, <https://www.kuka.com/en-ch/industries/automotive>

Traditional view on robotics:

- Robots are locked up in cages
- No humans around
- Highly conditioned environment

# Introduction

## From cages to human environments

New application areas where the traditional view isn't valid any more:

- Robots working in a human-like environment or with natural products:
  - **variability** and **uncertainty** in the environment or products
- **Human collaborators** close by
- **Humans physically interacting** with robots (jointly performing tasks)



# Introduction

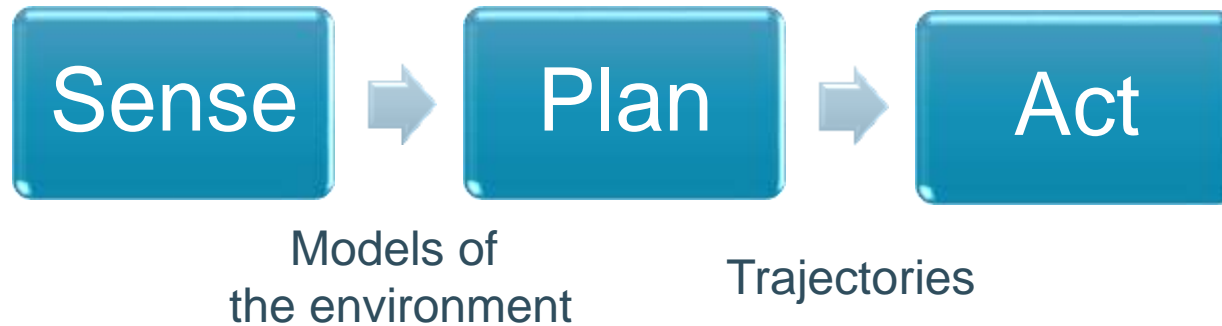
## Seemingly conflicting goals when creating “robot apps”

1. Dealing with variations in the process
  - Production line is less conditioned
  - Product variations (natural, processes such as molding)
  - Human interference/interaction
  - ➔ More complex and involved robot programming
2. Decreased development cost needed
  - Smaller production series
  - Rapid deployment

# Approach

# Approach

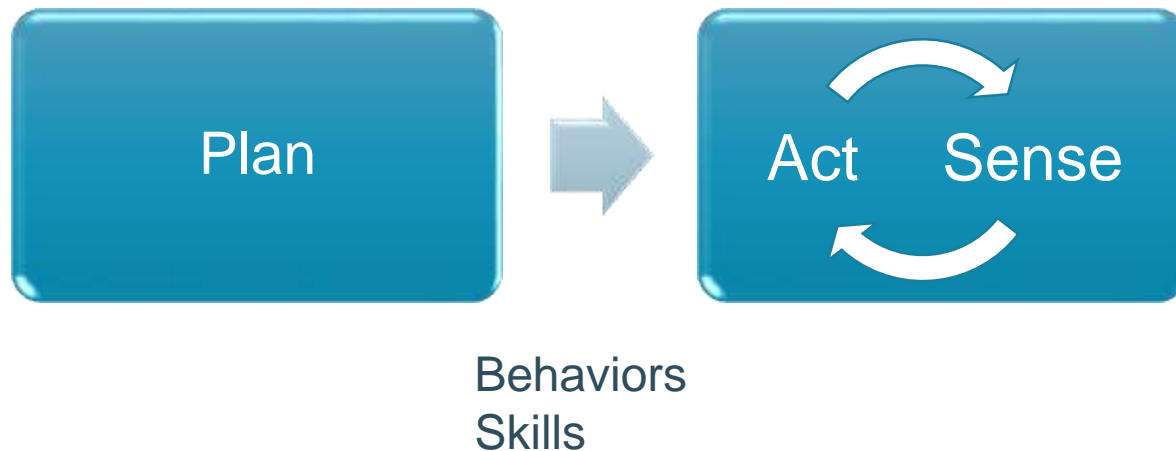
## Traditional : Sense-Plan-Act



- Requires extensive calibration
- Once planned, there is no flexibility during execution.  
(or at least, much coarser)

# Approach

## Skill-based

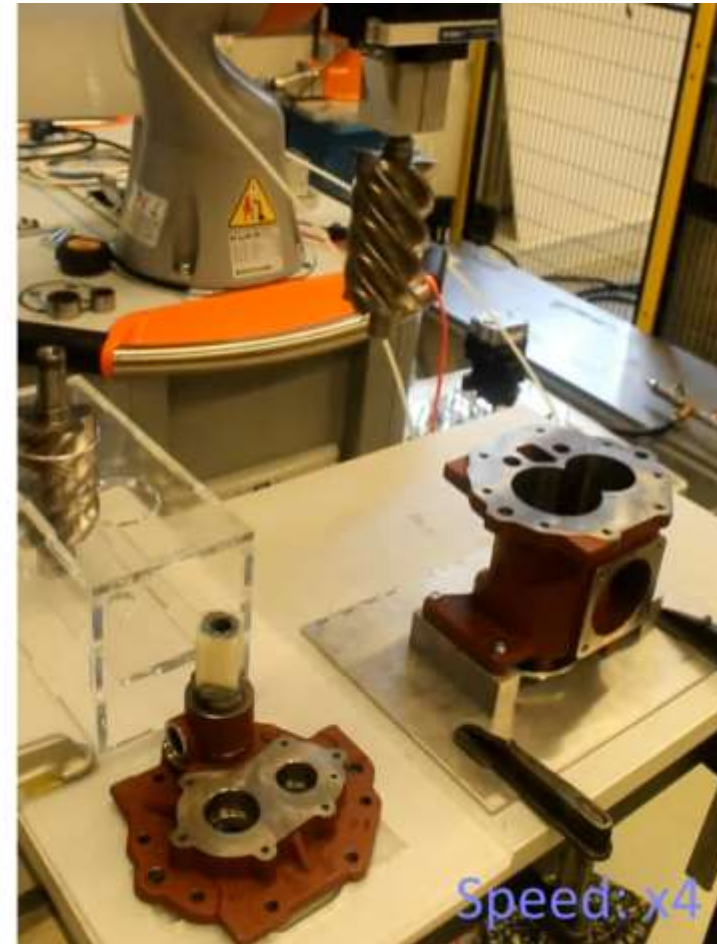
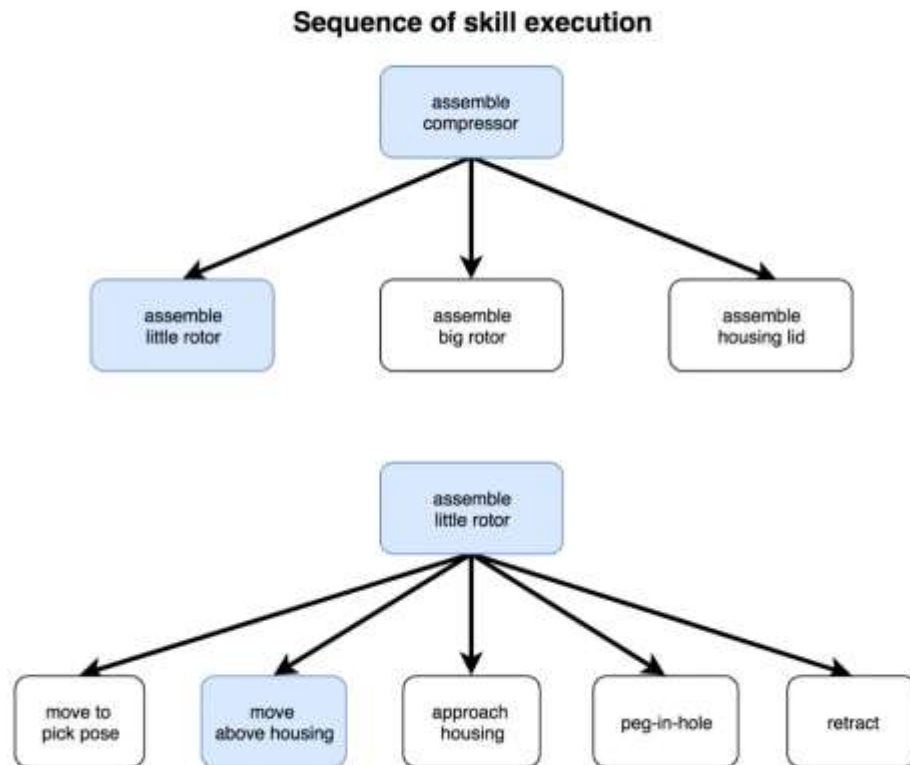


- Plan in terms of “**skills**” or behaviors
- **Reactive**
- Avoid calibration issues:
  - **Local** sensor measurements instead of **global**
  - Often more robust
  - Variable environments with human intervention/interaction



# Example of skill-based approach

## Force/Torque-based Assembly

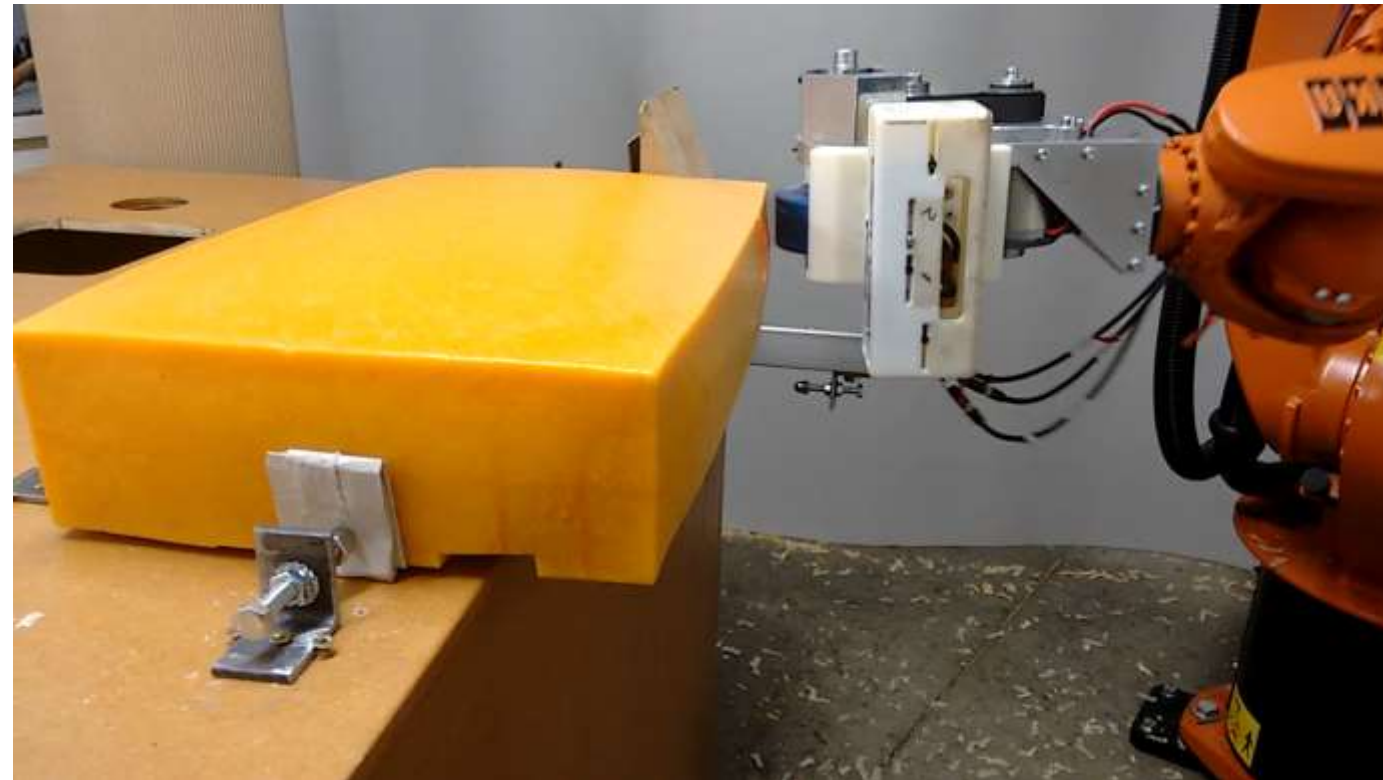


# Example of skill-based approach

## Cheese decrusting



- Cheese **decrusting** application
- A local measurement of 3 distances :  
(using laser distance sens.)
- High speed – high accuracy –  
avoiding calibration errors



Echord++ 3DSSC : KULeuven / FRS-Flexible Robotic Systems

# A constraint-based approach

Task function  $\underline{e}(q, t) \rightarrow 0$

First order  $\frac{de}{dt} = -k e$

Second order  $\frac{d^2e}{dt^2} = -2\zeta\omega_0 \frac{de}{dt} - \omega_0^2 e$

# A constraint-based approach

First order

$$\frac{de}{dt} = -k e$$

$$\frac{\partial e}{\partial q} \dot{q} + \frac{\partial e}{\partial t} = -k e$$

$$J\dot{q} = -k e - \frac{\partial e}{\partial t}$$

Jacobian

Feedback term

Feedforward term

# A constraint-based approach

First order

$$\frac{de}{dt} \leq -k e$$

$$\frac{\partial e}{\partial q} \dot{q} + \frac{\partial e}{\partial t} \leq -k e$$

$$J\dot{q} \leq -k e - \frac{\partial e}{\partial t}$$

Jacobian

Feedback term

Feedforward term

# A constraint-based approach

$$\underset{\dot{q}}{\text{minimum}} \sum_i w_i \varepsilon_i^2 + \lambda \sum_l v_l \dot{q}_l^2$$

Subject to:

$$\begin{array}{c} \vdots \\ J_i \dot{q} = -k e_i - \frac{\partial e_i}{\partial t} + \varepsilon_i \\ \vdots \end{array}$$

# eTaSL



# eTaSL

## expression graph based **T**ask **S**pecification **L**anguage

Constraint-based task specification and control framework to describe these **reactive** skills (behaviors)

- at each sample time (100 Hz→250 Hz), it **optimizes** the **control velocity** of each robot joint **subject to** a number of **constraints**
- **instantaneous optimization**: we do not (yet) look ahead in time
- only considers **kinematical model** of the robot  
→ we can still achieve high performance!

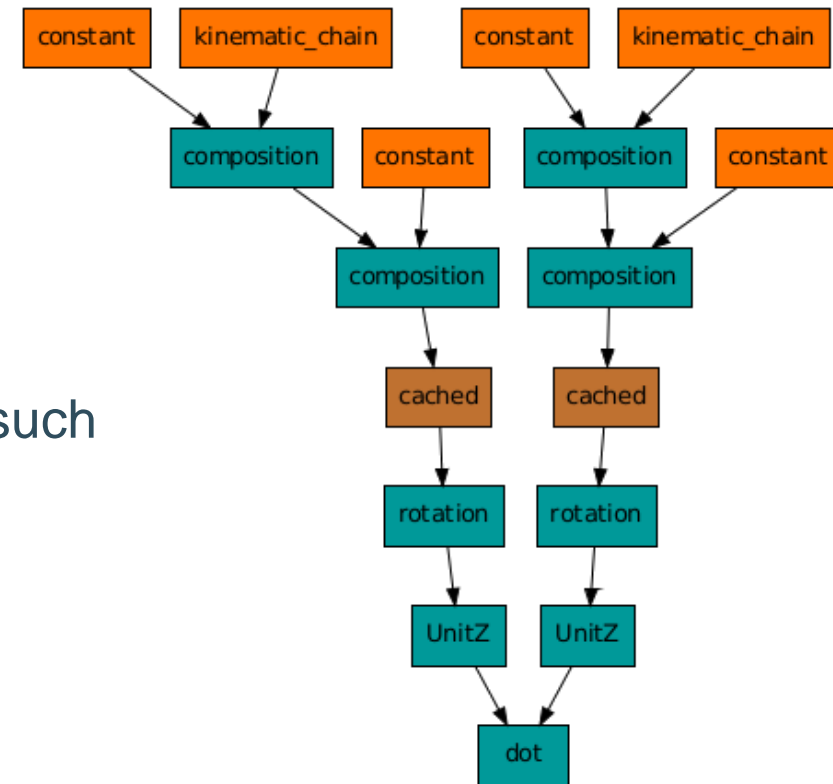
More information: <https://etasl.pages.mech.kuleuven.be/> and E. Aertbeliën and J. De Schutter, **Etasl/eTC: A Constraint-Based Task Specification Language and Robot Controller Using Expression Graphs**, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014



# eTaSL

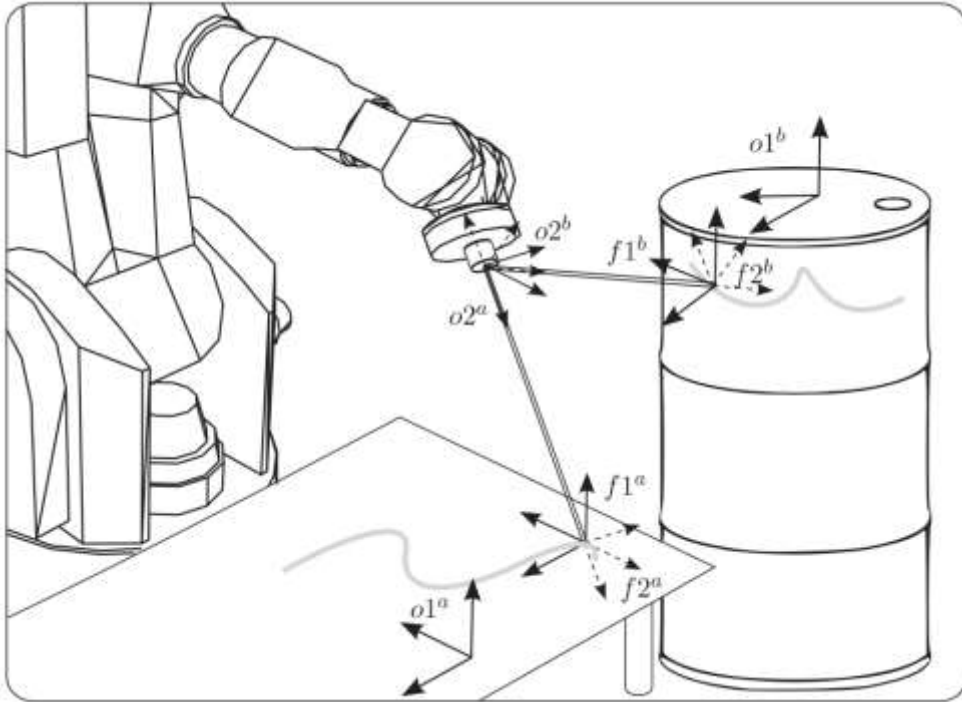
## expression graph based Task Specification Language

- Constraints are described using **expression graphs**:
  - Symbolic, graph-structure
  - **Not only scalar** expressions
  - **Robot** is described as a function of its joint variables
  - **Trajectory** as a function of time
  - Expressions can relate to **sensor-input**
  - **Simple language** (LUA-based) where you can write down such expressions
- Controller is automatically generated:
  - Evaluation
  - Introspection
  - Automatic differentiation for Jacobians (avoiding representational singularities)



# Variables & Feature variables

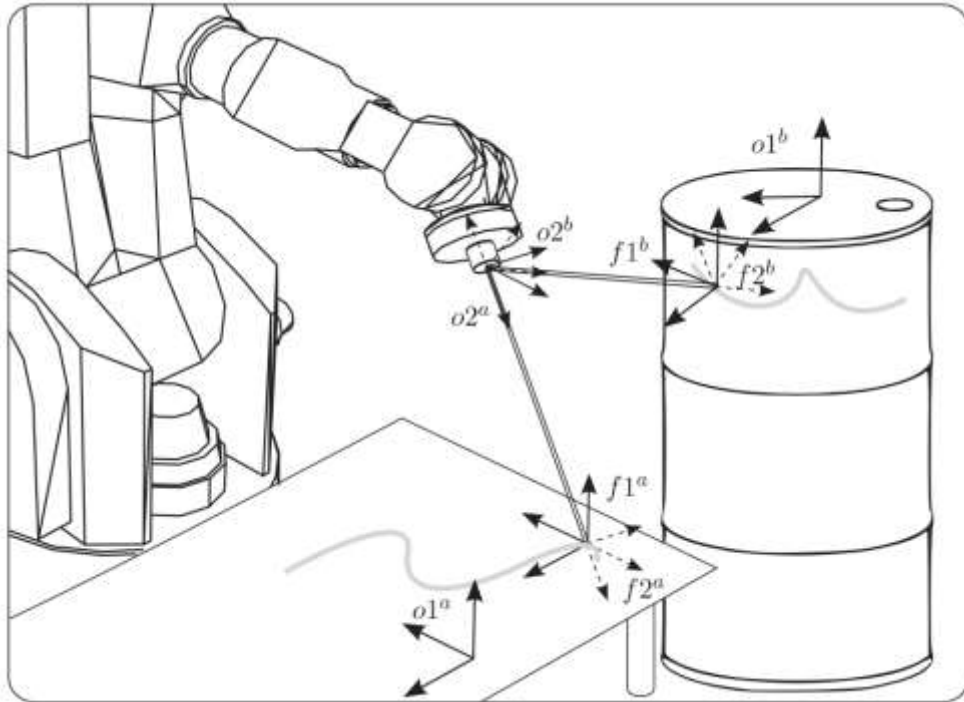
iTaSC (\*)



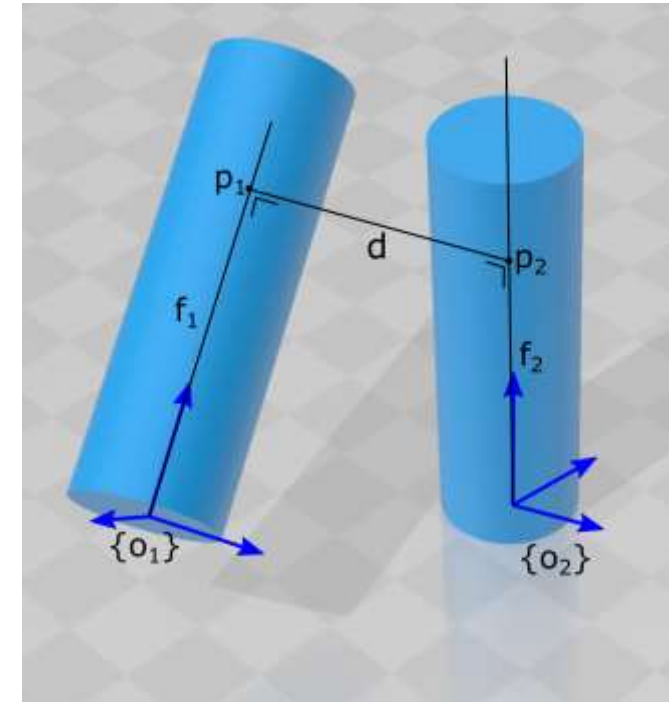
(\*) J. De Schutter et al., “Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty,” *The Int. Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007.

# Variables & Feature variables

iTaSC (\*)



eTaSL



(\*) J. De Schutter et al., "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *The Int. Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007.

# eTaSL

## expression graph based Task Specification Language

Constraints can be related to:

- the **task**: desired trajectory, speed, contact force, distance, etc.

- the **robot platform** and its limitations in terms of reachable/allowable joint positions velocities

(specified using URDF)

- the **environment**: e.g. avoiding collisions and self collisions
- interaction with **humans** (physical or cognitive)



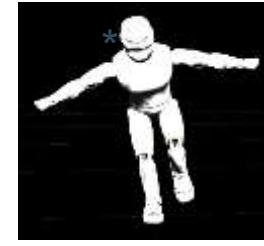
7 dof



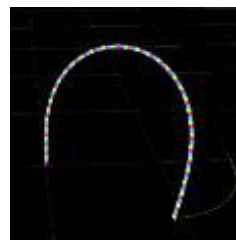
8 dof



20 dof



33 dof



100 dof

# eTaSL

## expression graph based Task Specification Language

- constraints can be **conflicting**:
  - priorities
  - soft constraints & weights
- and can be **equality** or **inequality** constraints (e.g. collision constraint)
- **explicit time**
  - task function expressions within the constraints:
  - a trajectory can be specified a mix of trajectories with time-varying weights
  - eTaSL perfectly deals with tracking errors  
(**automatically generates feedforward control term!**)

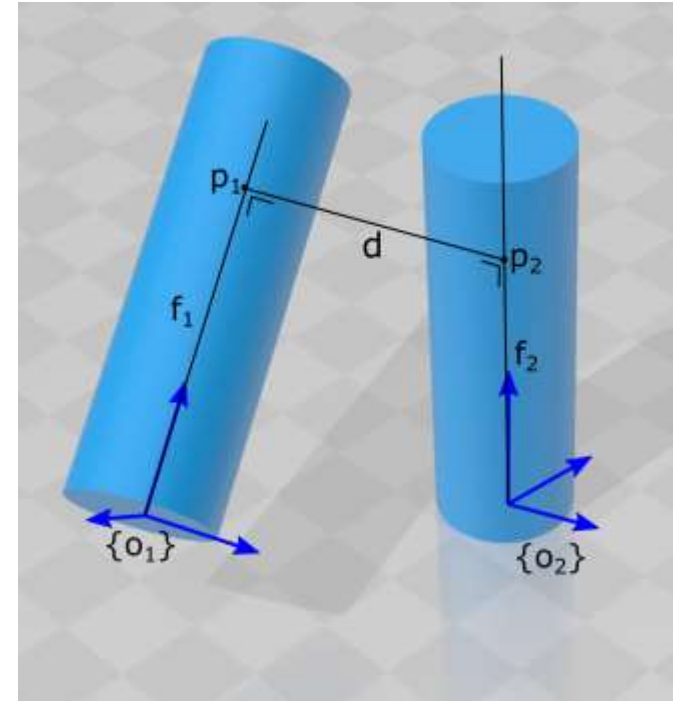
# Geometric constraints



# Geometric constraints

Tools to model geometric constraints:

- Facilitated by the use of feature variables
- Distances between convex objects using the GJK-algorithm (\*)
- Library for typical geometrical distances and angles.

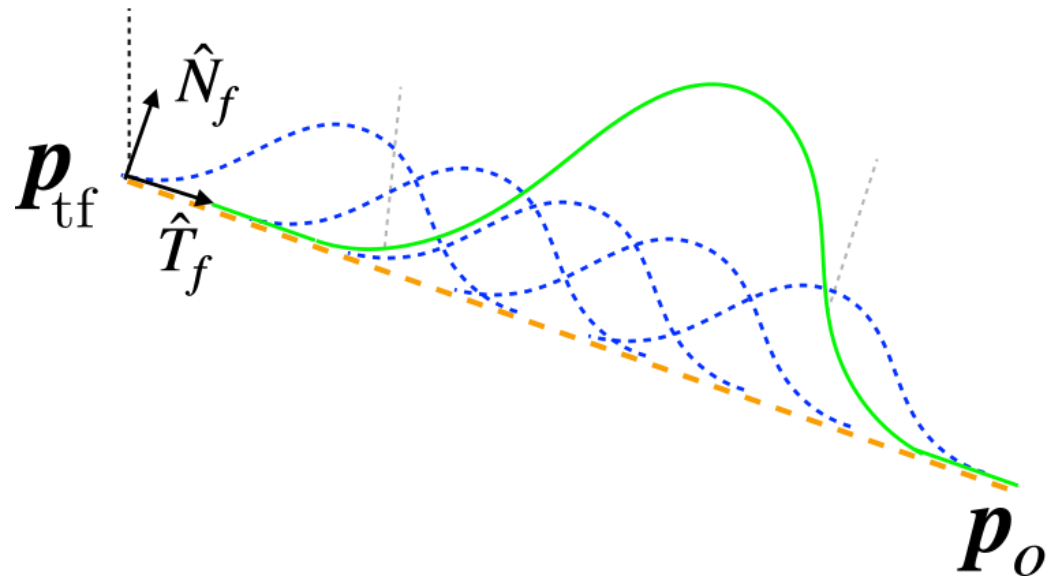


(\*) Gilbert, E. G., Johnson, D. W., & Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2), 193-203.





# Flexible trajectories

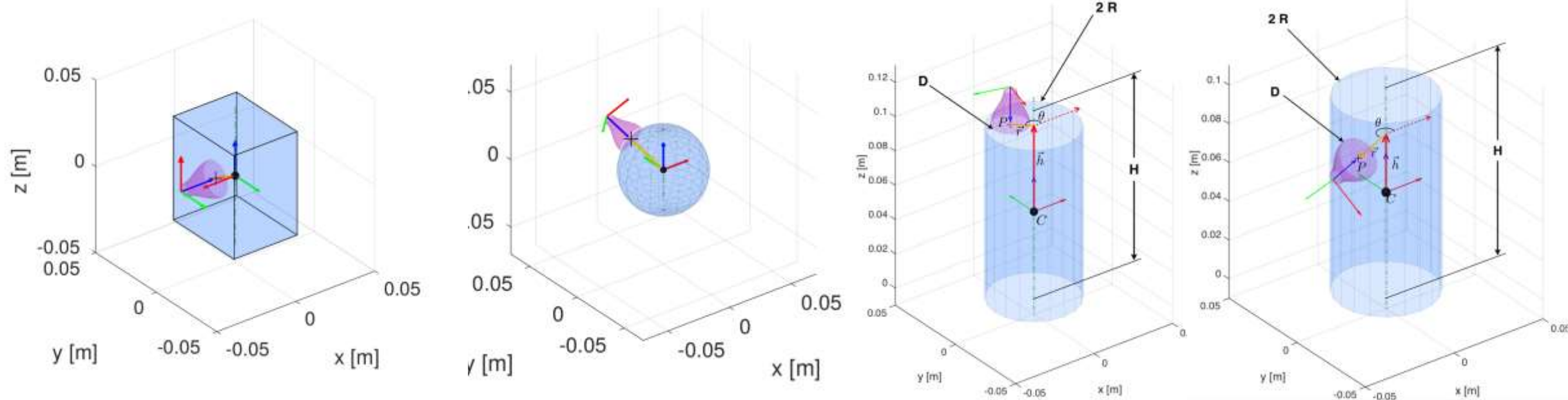


- Linear combination of basis functions  $B_i$  (in function of feature variables  $f_i$  and progress  $s$ )

$$T(s, f_1, \dots, f_n) = T_{mean}(s) + \sum_i f_i B_i(s)$$

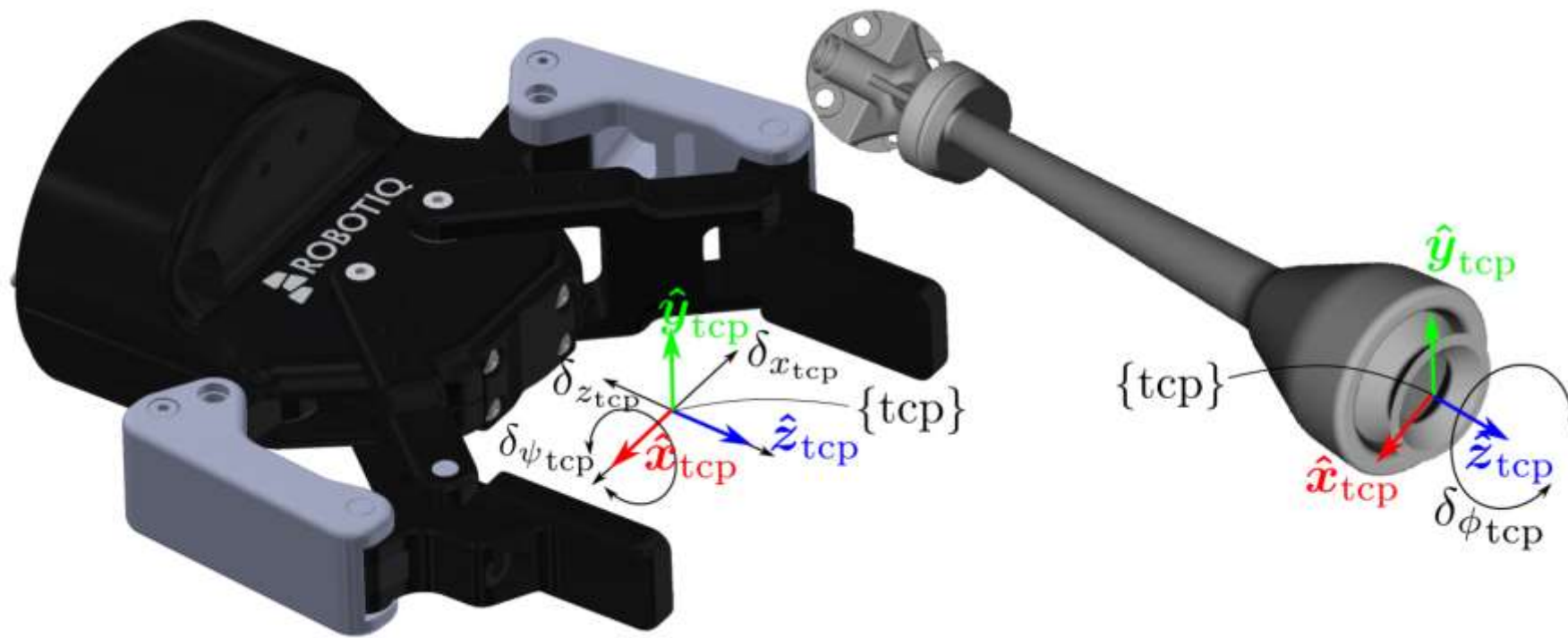
- The progress  $s$  is modeled separately and is related to time via a soft position or velocity constraint

# Reactively modeling grasp and contact



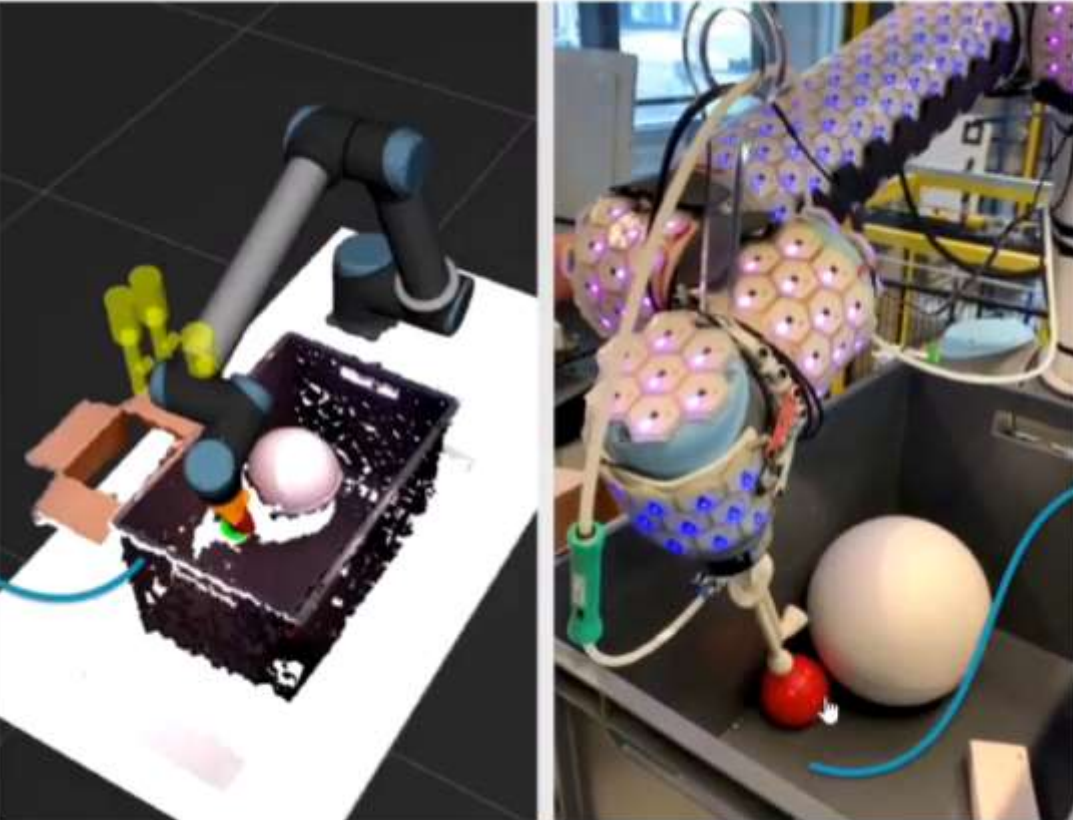
C. Vergara, S. Iregui et al. Generating Reactive Approach Motions Towards Allowable Manifolds using Generalized Trajectories from Demonstrations, IROS 2020

# Reactively modeling grasp and contact



# Flexible trajectories & grasp modeling

- Impose constraints “in the future”
- (local) collisions can be implemented by adding virtual “tools”
- reactive



Surface model  
parameters acquired  
with vision

# Sensor related constraints



# Sensor-related constraints

## Admittance constraints

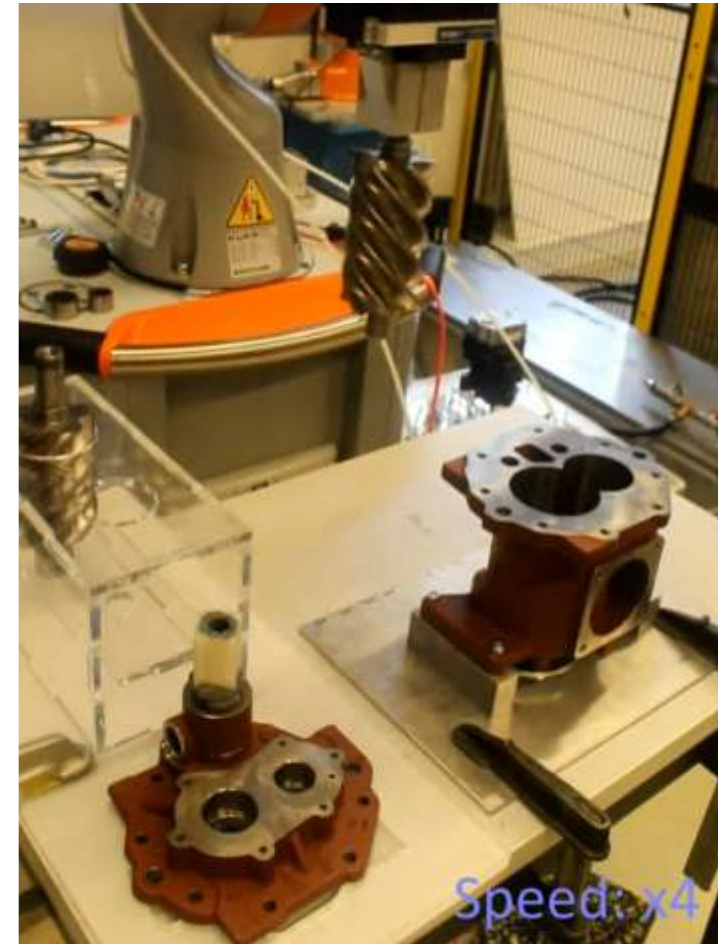
A generalized way to include sensors using an admittance control strategy:

- **model** of the sensor measurement
- sensor **measurement**
- **target** value

$$\underbrace{\mathbf{J}(\tilde{\mathbf{q}})}_{\frac{\partial}{\partial \tilde{\mathbf{q}}} \text{model}} \dot{\tilde{\mathbf{q}}} = -K \underbrace{(e(\tilde{\mathbf{q}}))}_{\text{meas} - \text{target}} - \underbrace{\frac{\partial e}{\partial t}(\tilde{\mathbf{q}})}_{\frac{\partial e}{\partial t} \text{model}} + \epsilon$$

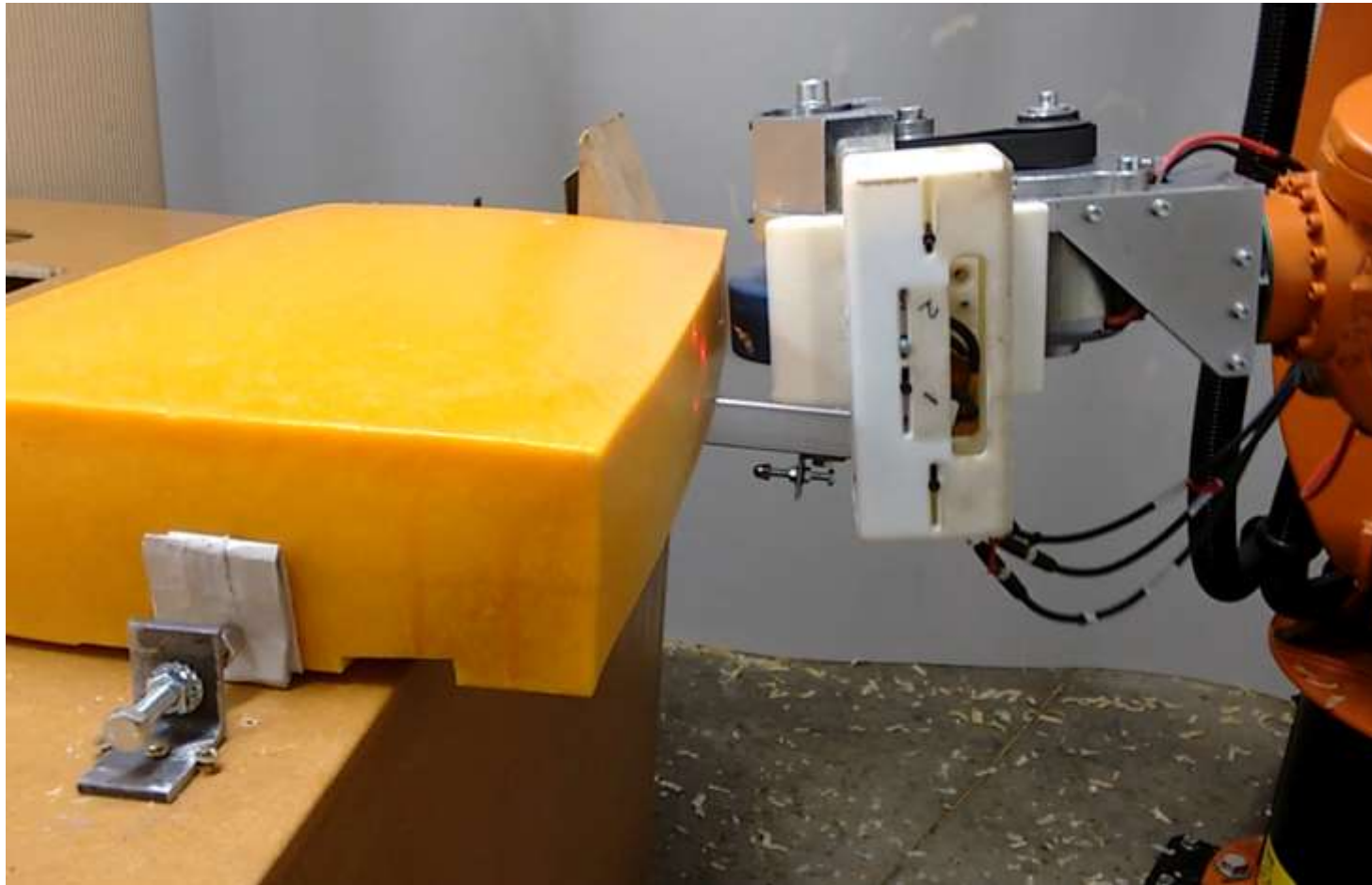
# Sensor-related constraints

## Force/Torque for assembly



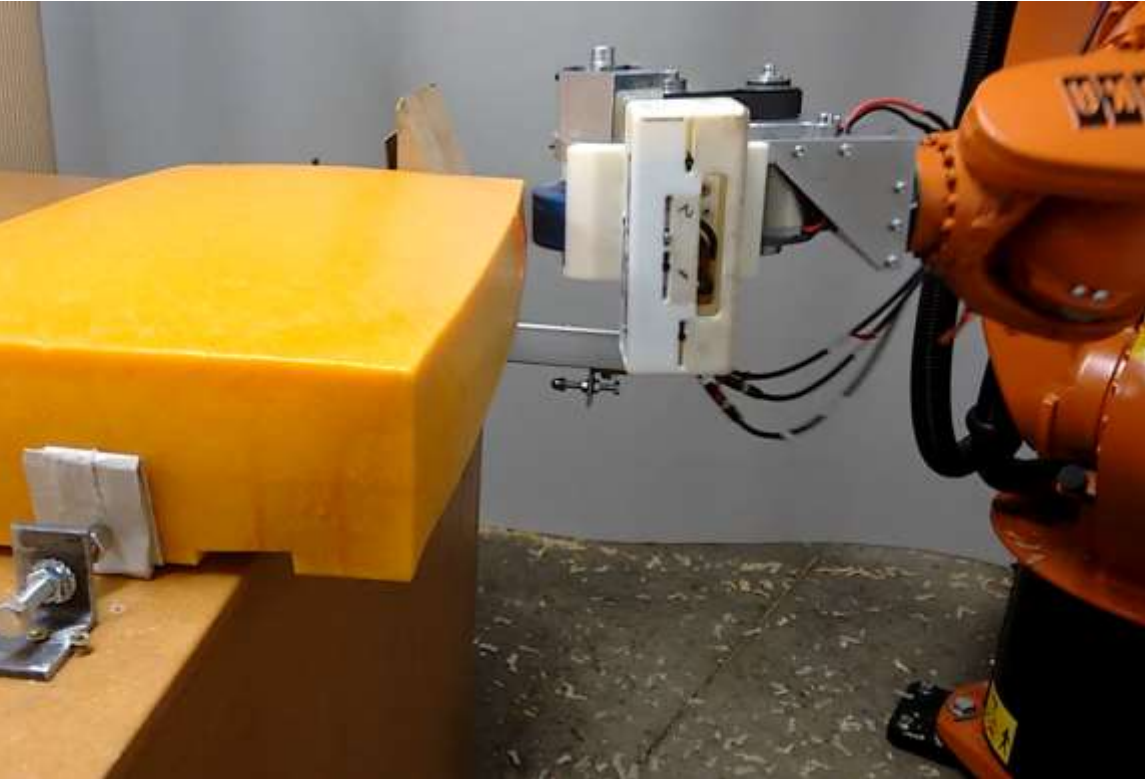
# Sensor-related constraints

## distance sensors





# Combining constraints



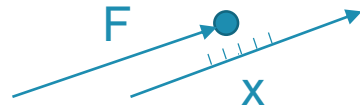
Added value of eTaSL:

- generating robot trajectory based on model built from sensor readings
- trajectory control (feedback + feedforward)
- automatic pitch control to keep laser sensor measurements within range
- compensation of time delays in control loop
- automatic speed reduction to keep joint velocities within limits
- smooth and fast transition between approach/retract and cutting trajectory

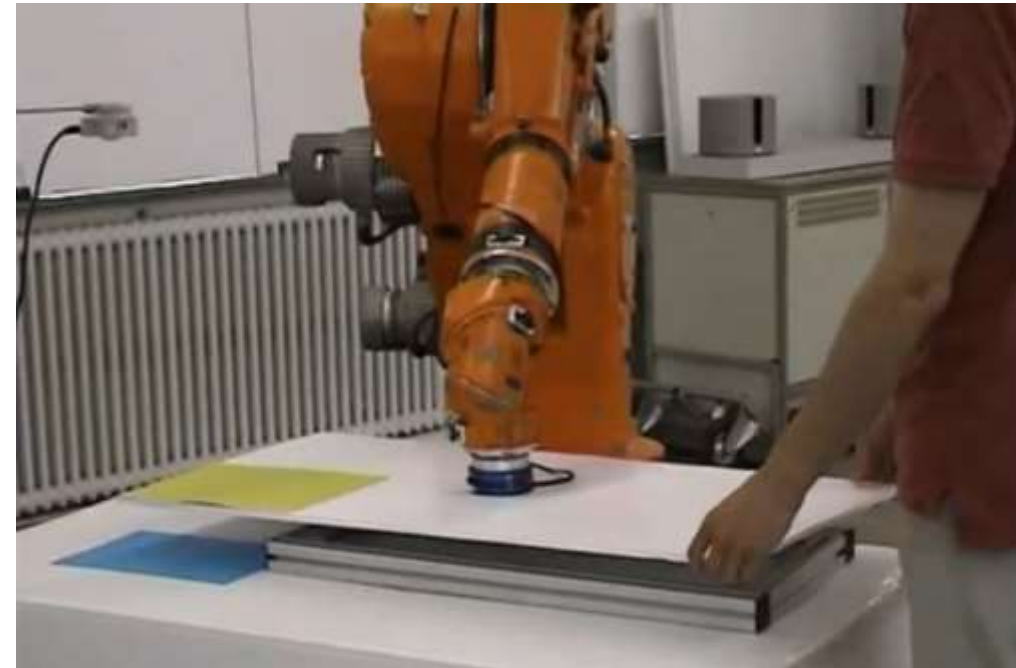
Echord++ 3DSSC : KULeuven / FRS-Flexible Robotic Systems

# Sensor-related constraints

## Conflicting admittance and position constraints



- Impedance = conflicting **position** and **force** constraint
- \* Need **not** necessarily the same **reference** point!
- \* Often used in **shared** control

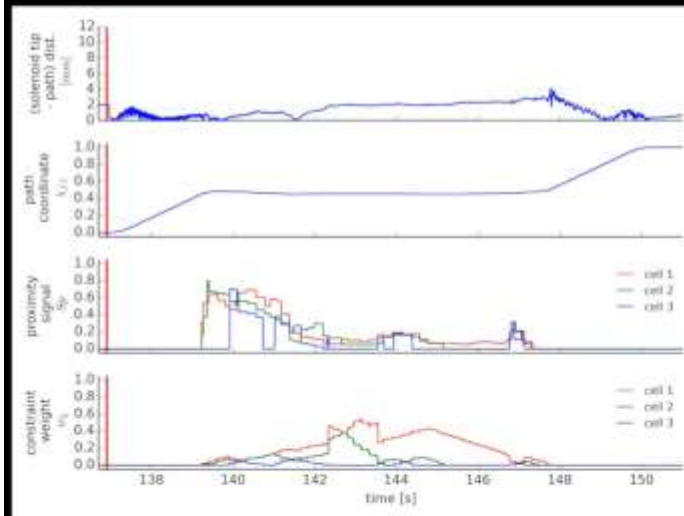


# Sensor-related constraints

## Skin

The operator place a screw while the robot moves to insert the next solenoid.

- Skin with 400 cells that measure both distance and force.
- Defined behavior along a trajectory and away from that trajectory.
- Trajectory adapts itself to the information from vision.



# Sensor-related constraints

## Force/Torque for contour following



Movie by Flexible Robotic systems (FRS, <https://www.frsrobotics.com/>)

No pre-programmed positions, the robot is automatically adapting to the contour.

A constraint formulation similar to the task frame formalism (\*)

Can still be combined with other Non-task frame related constraints

H. Bruyninckx and J. De Schutter, "Specification of force-controlled actions in the "task frame formalism"-a synthesis," in IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 581-589, 1996

# Modeling human motion



# Modeling human motion

Why?

- For use in **programming-by-demonstration**:  
⇒ For rapid deployment
- To **anticipate** and **predict** human motion in the neighborhood of the robot
- For **shared control**



a modeling approach for **reactive** control and **constraints**.

# Demonstration of tasks



Demonstrate task segments

while recording:

- poses
- wrenches

## Kinesthetic teaching

- ensure feasibility
- less calibration efforts needed
- use previous demonstrations to facilitate
- more disturbance forces



## Passive observation:

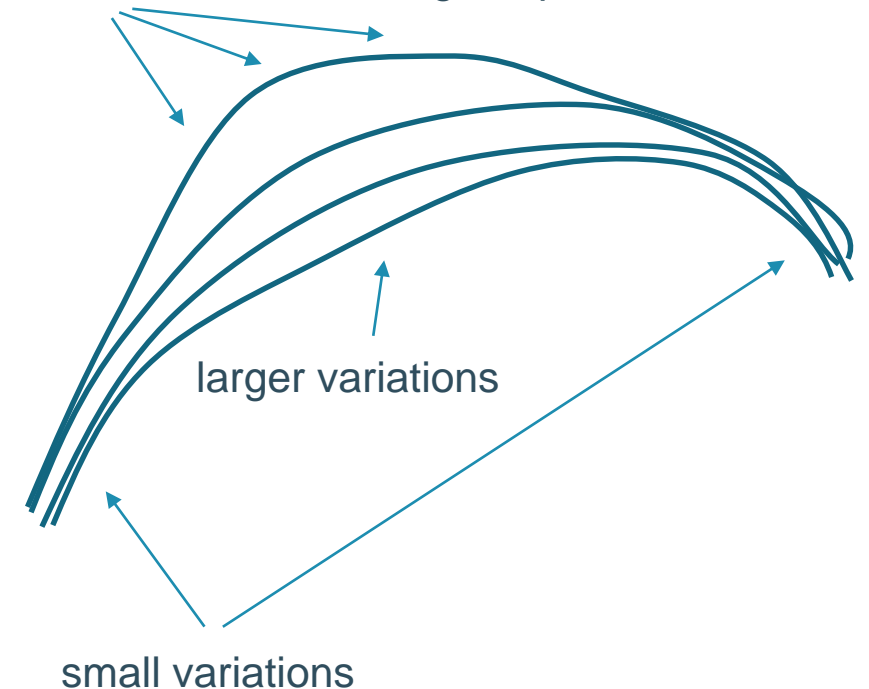
- feasibility is not guaranteed
- less disturbed demonstrations

# Programming by demonstration

## and combining this with constraint-based task specification

- we learn a **trajectory** and its **allowable variations** from demonstrations using a generative probabilistic approach:  
**Probabilistic Principal Component Analysis (PPCA)**
- combine with (model-based) constraint-based task specification
  - to support the **demonstration**
  - to add constraints for the **task execution**

correlation of the variations along the path





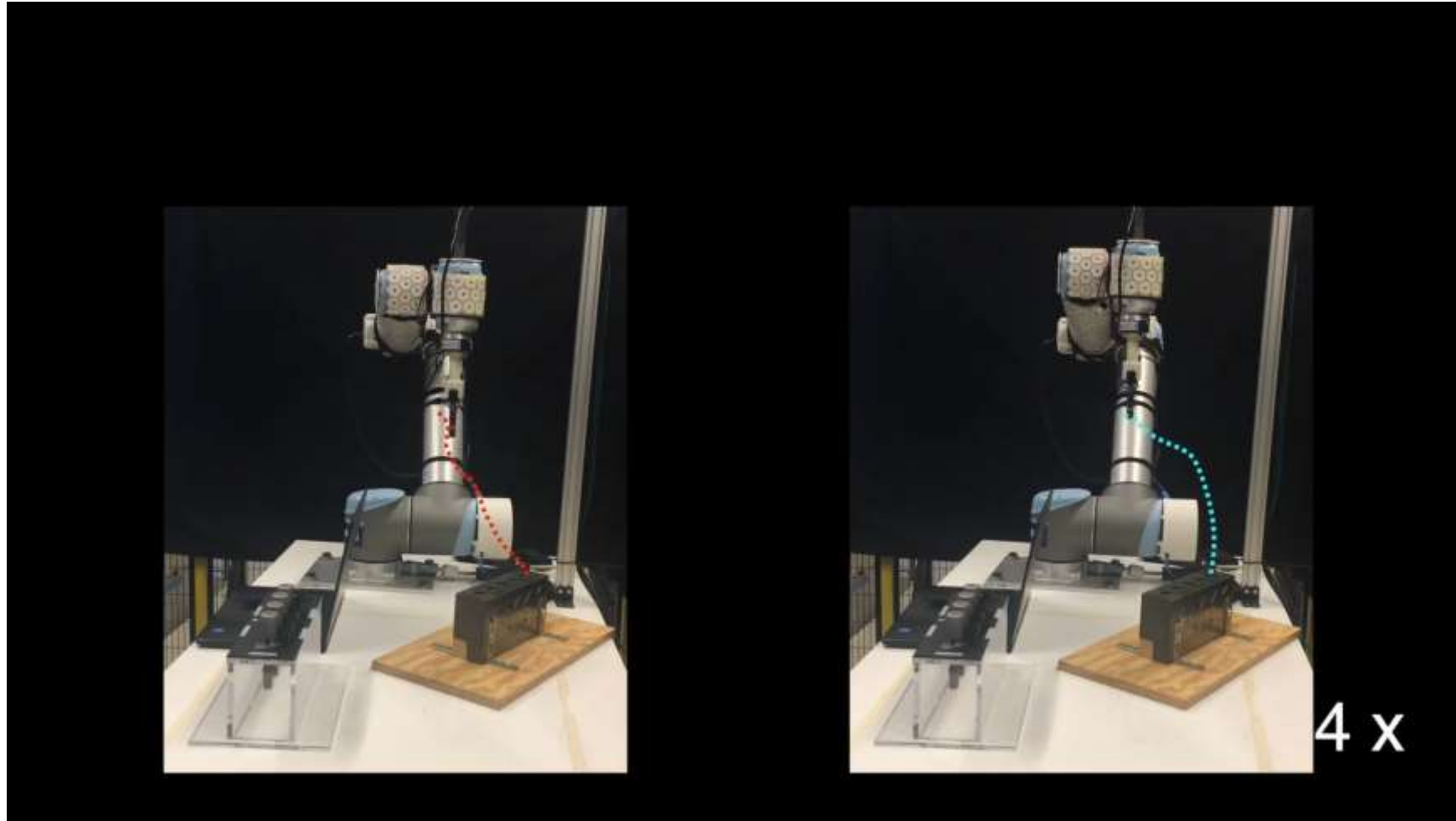
# PbD for the assembly of solenoids: demonstrations



# PbD for the assembly of solenoids: guided demonstrations



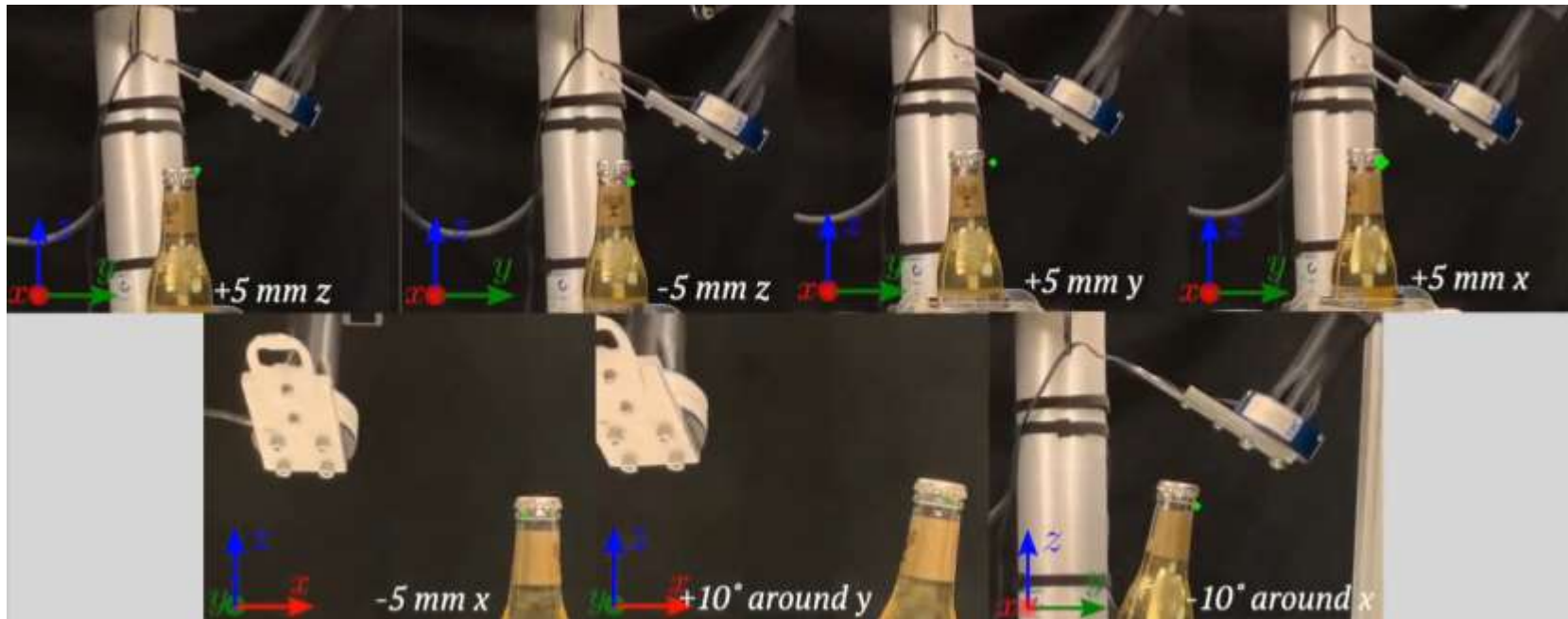
# PbD for the assembly of solenoids : execution



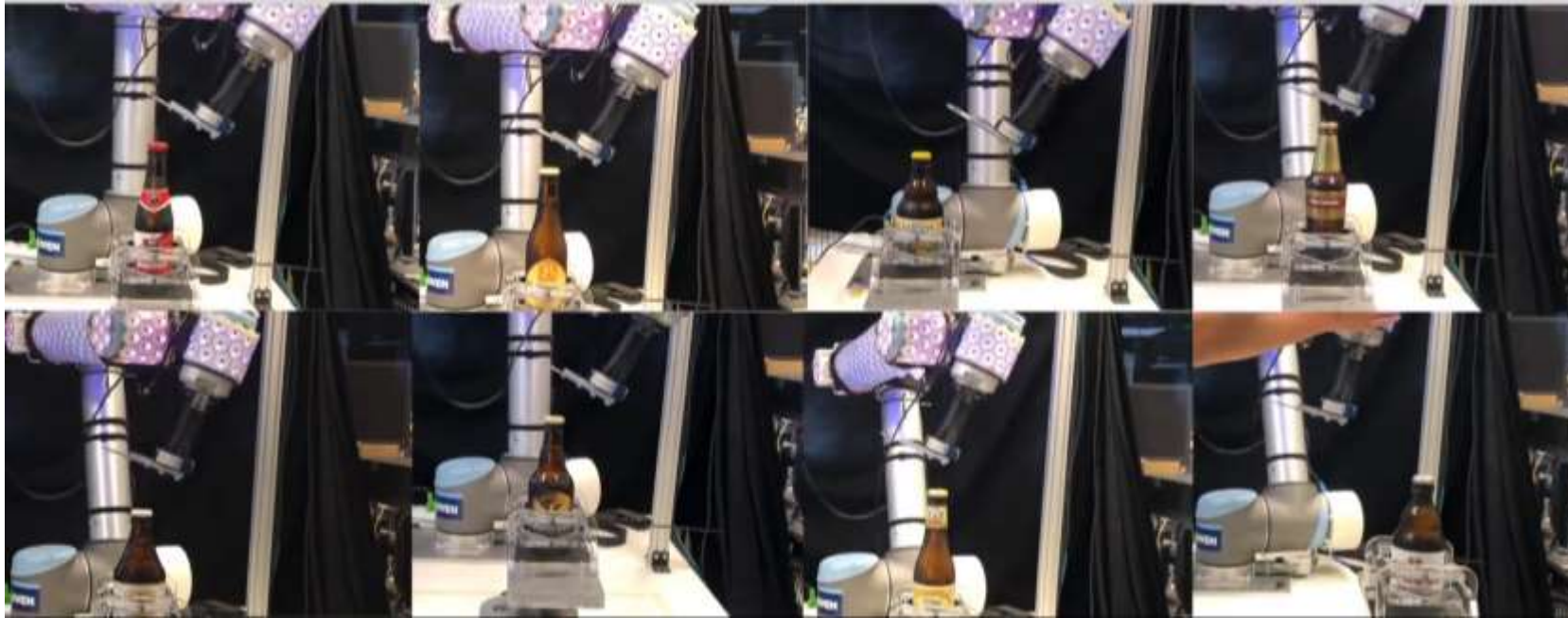
# Tasks with an approach & contact-phase: bier opening



# Tasks with an approach & contact phase: contour following



# Tasks with an approach & contact phase: contour following



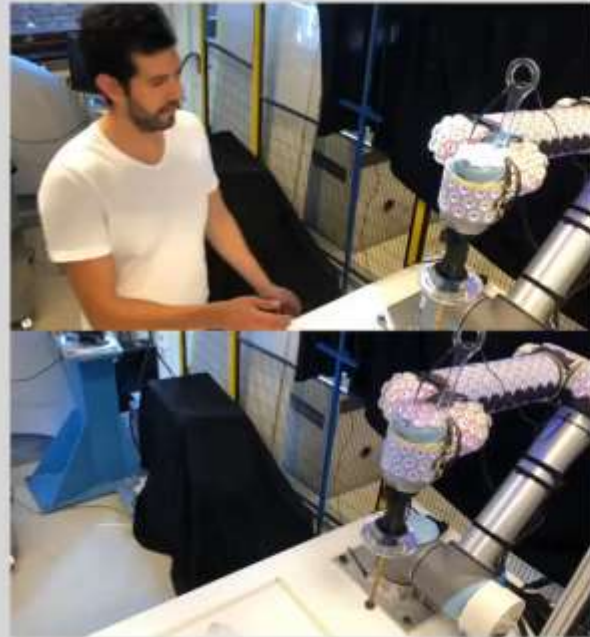
# Tasks with an approach & contact phase: contour following

Similarly, in another *use case*, a user demonstrates how to *approach* to perform a *contour following* task



# Tasks with an approach & contact phase: contour following

*Same pose, wrench and evolution constraints as in the bottle-opening case are used to perform this task*





# Conclusion



# Conclusion

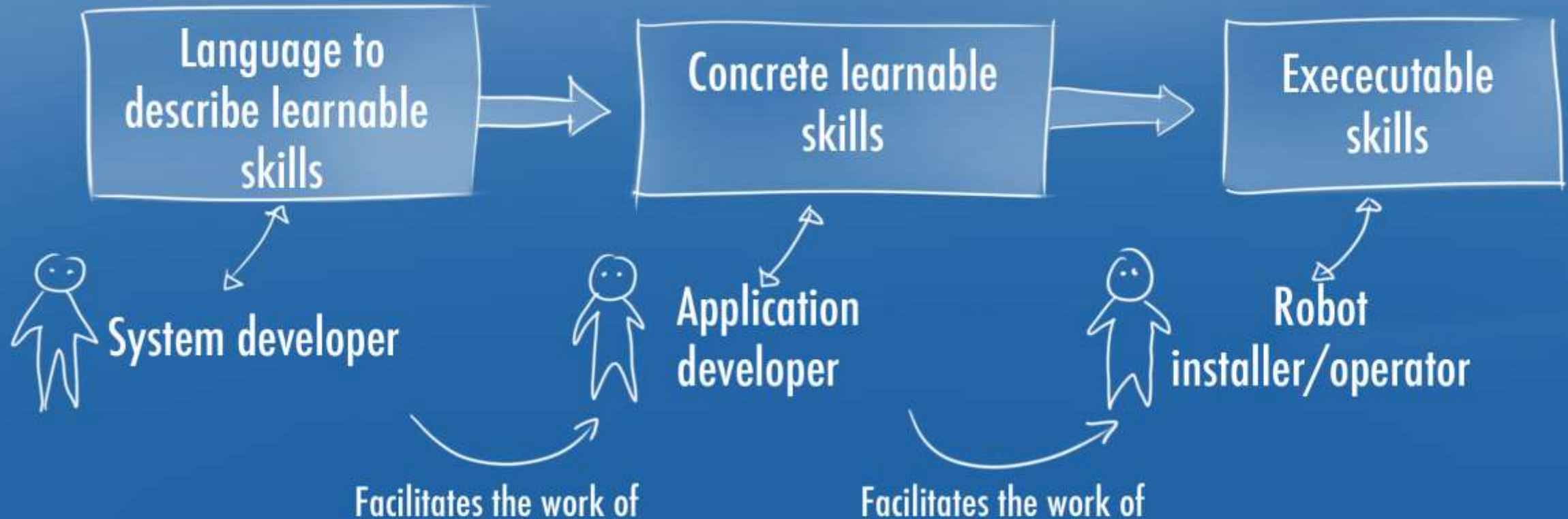
## Seemingly conflicting goals when creating “robot apps”

1. Dealing with variations in the process
  - Production line is less conditioned
  - Product variations (natural, processes such as molding)
  - Human interference/interaction
  - More complex and involved robot programming
2. Decreased development cost needed
  - Smaller production series
  - Rapid deployment

- Almost independent of app domain
- Larger effort
- Larger development time

- Application domain specific
- Expert in the app. Domain

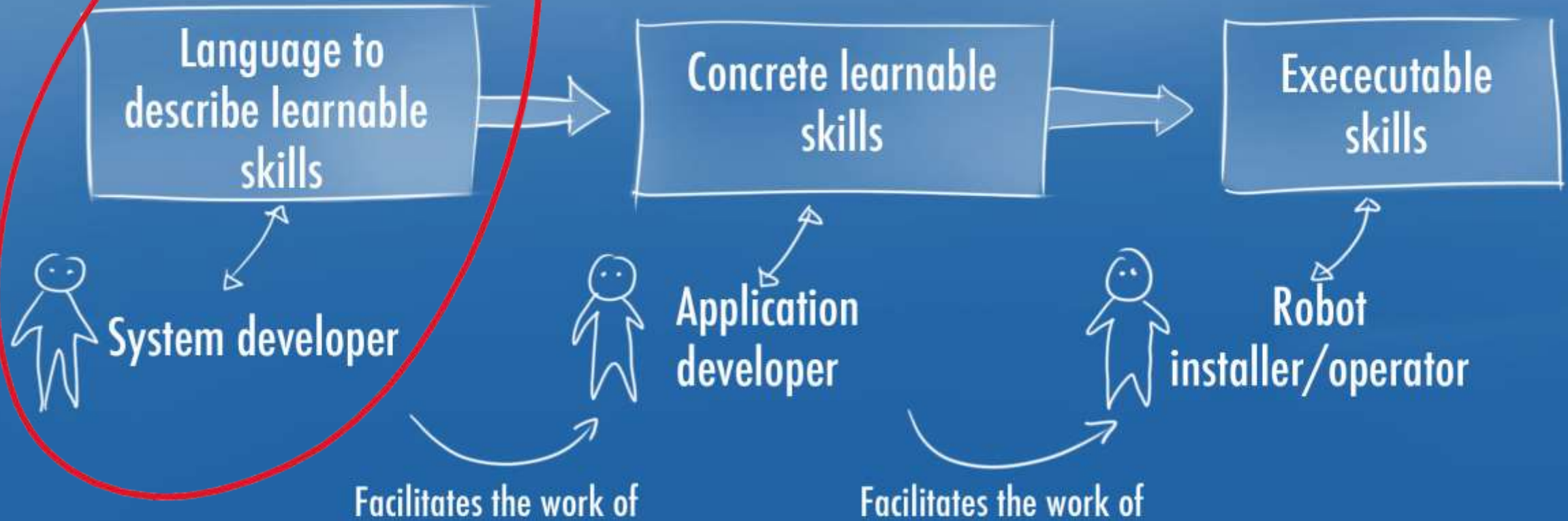
- For a given product line
- Small effort
- Quick deployment



- Almost independent of app domain
- Larger effort
- Larger development time

- Application domain specific
- Expert in the app. Domain

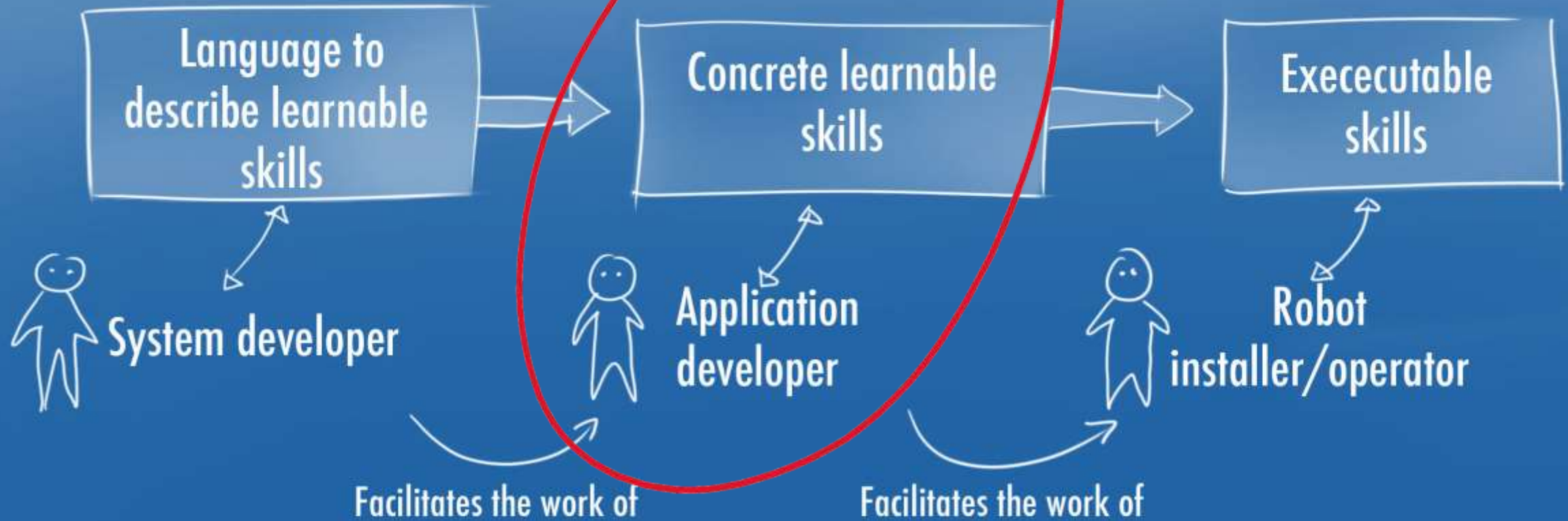
- For a given product line
- Small effort
- Quick deployment



- Almost independent of app domain
- Larger effort
- Larger development time

- Application domain specific
- Expert in the app. Domain

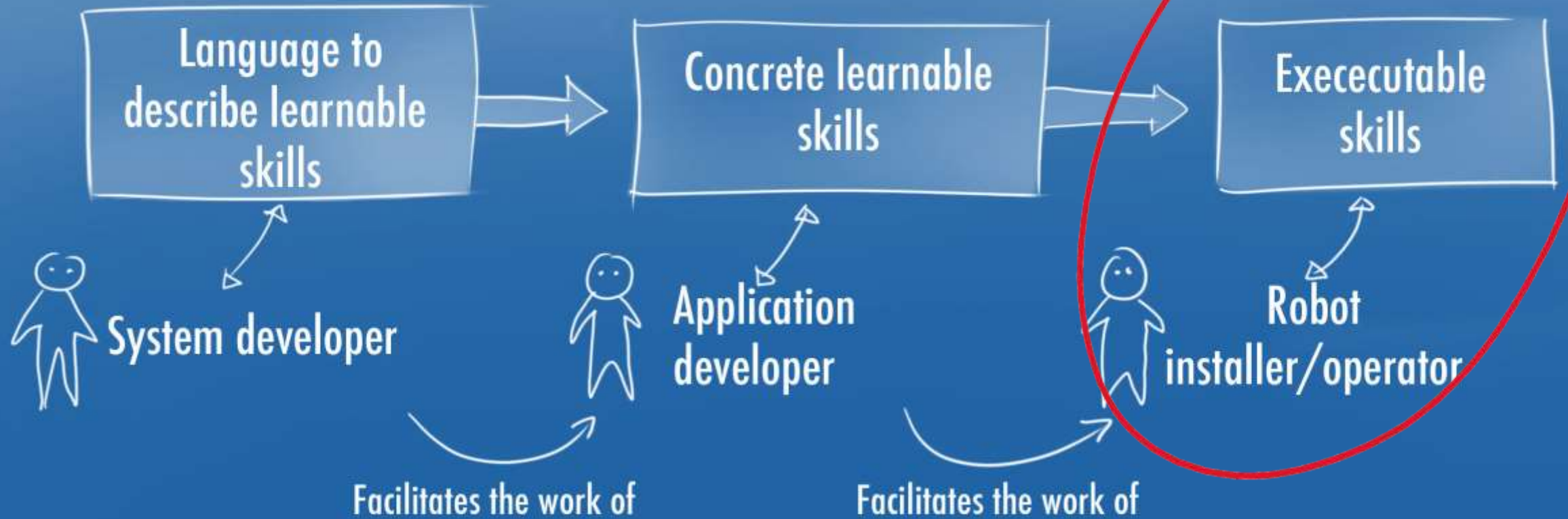
- For a given product line
- Small effort
- Quick deployment



- Almost independent of app domain
- Larger effort
- Larger development time

- Application domain specific
- Expert in the app. Domain

- For a given product line
- Small effort
- Quick deployment



# Available software



# eTaSL Software

<https://etasl.pages.gitlab.kuleuven.be/>

Higher level specification

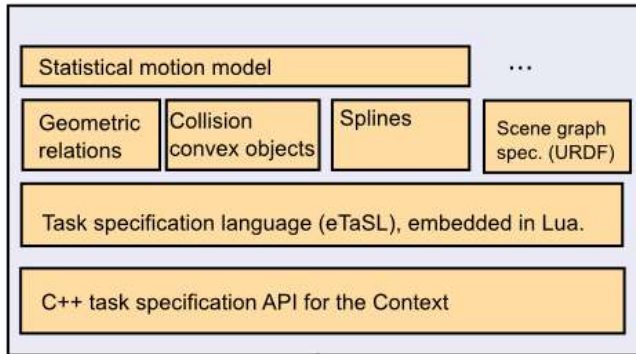
Formal, taking into account application specific semantics, intuitive, ...

Specification

Additional libraries

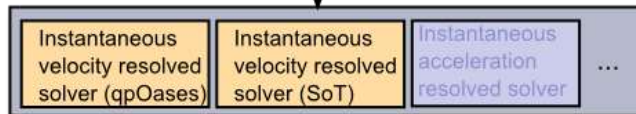
eTaSL

eTC



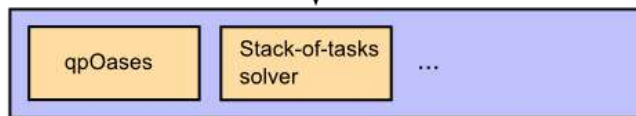
Solver

Context data structure  
(uses expression graphs)



Numerical solver

numerical QP-optimisation problem



The **separation of specification and controller implementation**

Specifications can be manipulated and assembled (even on-the fly if needed)

Layered approach:

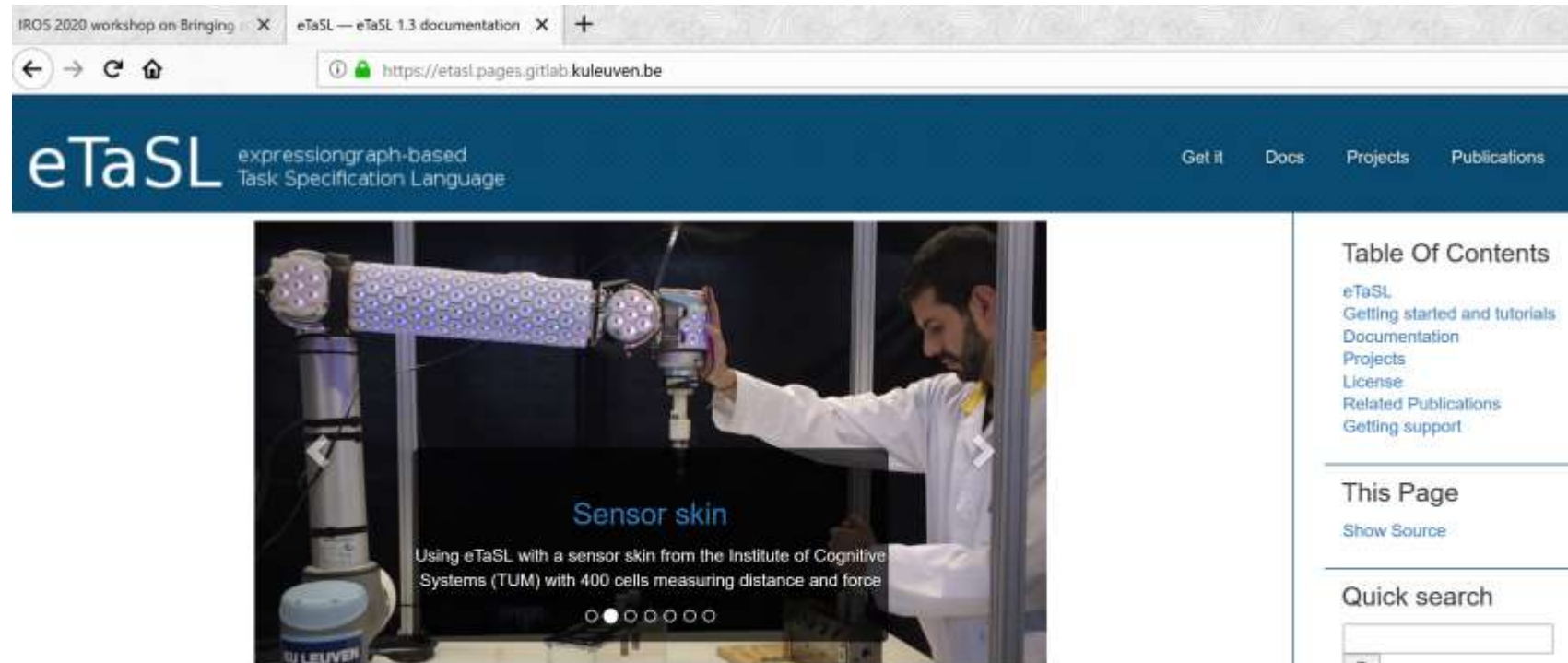
- eTaSL is a library.
- a Python driver for quick prototyping
- ROS/OROCOS-RTT/eTaSL for more complete robot applications.

Libraries for all different types of constraints



# eTaSL Software

- <https://etasl.pages.gitlab.kuleuven.be/>



The screenshot shows a web browser window with two tabs: "IROS 2020 workshop on Bringing" and "eTaSL — eTaSL 1.3 documentation". The address bar shows "https://etasl.pages.gitlab.kuleuven.be". The website header is dark blue with the "eTaSL" logo and the text "expressing long graph-based Task Specification Language". Navigation links for "Get it", "Docs", "Projects", and "Publications" are on the right. The main content area features a large image of a robot arm with a purple sensor skin. Below the image is the text "Sensor skin" and "Using eTaSL with a sensor skin from the Institute of Cognitive Systems (TUM) with 400 cells measuring distance and force". A sidebar on the right contains a "Table Of Contents" with links to "eTaSL", "Getting started and tutorials", "Documentation", "Projects", "License", "Related Publications", and "Getting support". Below that is a "This Page" section with a "Show Source" link, and a "Quick search" section with a search input field and a "Go" button.

For more information on the above examples, you can click on the links above or you can go directly to the [Showcase](#) page.

## eTaSL

**eTaSL** is a task **specification** language for **reactive** control of robot systems. It is a language that allows you to describe how your robotic system has to move and interact with sensors. This description is based on a **constraint-based** methodology. Everything is specified as an optimization problem subject to constraints. [What eTaSL/eTC is and is not](#) explains this further. The following [presentation](#) details the motivation behind eTaSL and explains the basic semantics/syntax of an eTaSL specification. The [Showcase](#) page gives many example videos.



# eTaSL Software

<https://etasl.pages.gitlab.kuleuven.be/>

- Two types of tutorials available:
  - 1. Python notebooks (via [Binder](#))

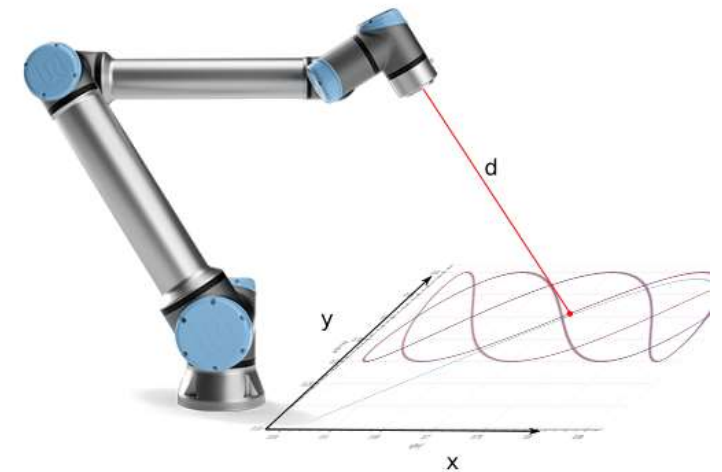
## Tutorial 2

We use the same robot definition as in Tutorial 1.

```
In [ ]: %matplotlib inline
import numpy as np
```

```
In [ ]: from IPython.core.display import display, HTML
display(HTML("<style>.container { width:80% !important; } </style>"))
```

In a similar way as in tutorial 1, we define a robot from an URDF-file. Our new task specification deals with a laserspot following the z-axis at the end-effector. The distance from the end effector to the laserspot is modeled using a **feature** compute it explicitly. In this case, it is still possible to compute it explicitly, but for more complicated surface, this will not th



We then state that the laserspot should be on the ground plane (constraint `z`: `coord_z(laserspot) == 0 == tgt_z`), an (constraints `x` and `z`).

To make things more interesting, we also impose some limits on the laser-distance (constraint `laserdistance`, `dista` 0.55 [m])

# eTaSL Software

<https://etasl.pages.gitlab.kuleuven.be/>

- Two types of tutorials available:
  - 2. Full robot application example and template using ROS/Orocos/eTaSL

(Directly supporting: simulation, UR10, Kinova Gen 3, Franka Emika-Panda, KUKA-iiwa)

## Tutorial 5 - Using another robot

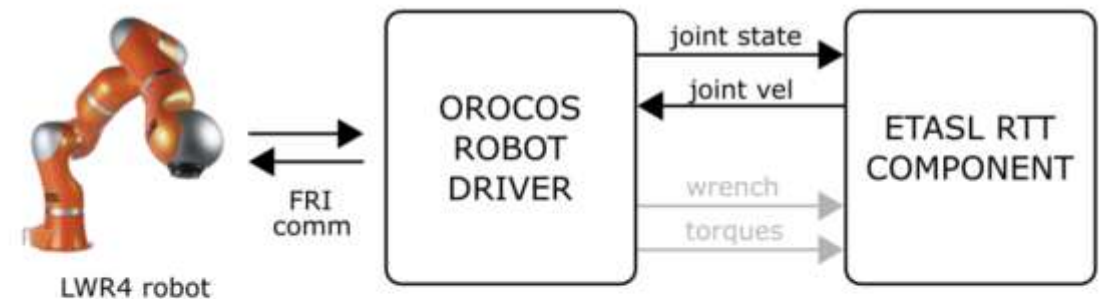
This tutorial gives an overview on the important aspects you need to take into account when integrating your own robot. This is exemplified by integrat example is not provided in the tutorial.

### Overview of an Orocos Robot Driver

As eTaSL is executed as an Orocos component, it is recommended to create an Orocos-based robot driver as well. This way, real-time communication i better handled. Another less recommended way is to handle the communication through ROS topics. This is made possible by using the RTT-ROS inter

A minimum requirement for an Orocos robot driver to be used with eTaSL is that the component must return joint positions and receive joint velocity : may also have extra IO ports for other data such as joint torques, but this largely depends on the specific robot features. The default eTaSL IO port type command is `array`. However, it is also possible to add IO ports of different types. Currently, eTaSL also supports receiving joint state of type `sensor_ command of type motion_control_msgs::JointVelocities.`

As an illustration, figure belows shows how an LWR4 driver communicates to the eTaSL component. The communication to the robot is using the FRI (f scope of this tutorial. Interested users can find the driver [here](#). The driver comes with the minimum IO ports while also provides extra ports for the wre



### Deploying the Robot Driver and Connecting the Minimum IO Ports

Using the already developed LWR4 driver, the next task is to write a LUA library that contains functions to deploy the robot driver and connects the api convention, this library is located at the folder `/scripts/lib/`. Some examples are already provided, such as the UR10 robot (`etasl_ur10.lua`) and L

# eTaSL Software

<https://etasl.pages.gitlab.kuleuven.be/>

- Show-case of examples:

<https://etasl.pages.gitlab.kuleuven.be/showcase.html>



A model-based task specification that includes programming by demonstration aspects

The video shows a robotic pick and pack application, where model-based task specification and programming by demonstration are combined in a learnable skill for online and reactive execution. Trajectories and its variations are extracted from programming by demonstration while allowing incremental learning.



A demonstration of dual arm motion on the PR2 robot using our expression-based Task Specification language (eTaSL).

In this example a constraint-based task specification and control framework is used to control the robot to satisfy a whole range of constraints, such as self-collision avoidance, joint limits, joint velocity limits, camera that looks at the end effector of the right arm, circular trajectories for both arms that intersect...



# Acknowledgements

Erwin Aertbeliën



Joris De Schutter



Cristian Vergara



Santiago Iregui



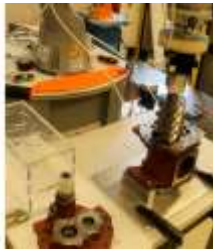
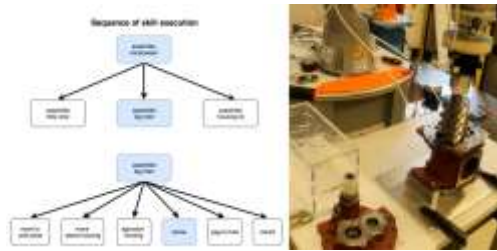
Yudha Pane



A list of publications related to this presentation:

<https://etasl.pages.gitlab.kuleuven.be/pub.html>

# Acknowledgements

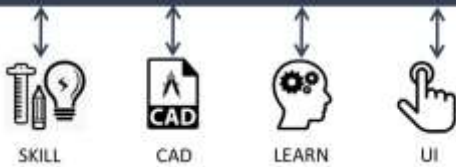


Flanders Make project  
FINROP



Flanders Make project  
MULTI-ROB

**PROUD-ARCHITECTURE**



Flanders Make project  
PROUD



EU-FP7 Robohow.org

**ROBOHOW.CO**



EU-FP7 Factory-in-a-day



EU-FP7 Echord++ - 3D Smart Sense and  
Control

The Robotics research group of KU Leuven is a  
core lab of Flanders Make

**KU LEUVEN**

FLANDERS  
**MAKE**  
DRIVING INNOVATION IN MANUFACTURING