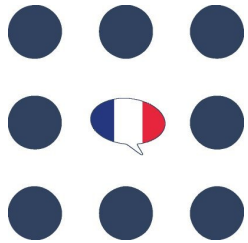
`robot_arm_calibration`: a kinematic calibration tool for serial robots, or the unberdening of a crucial task

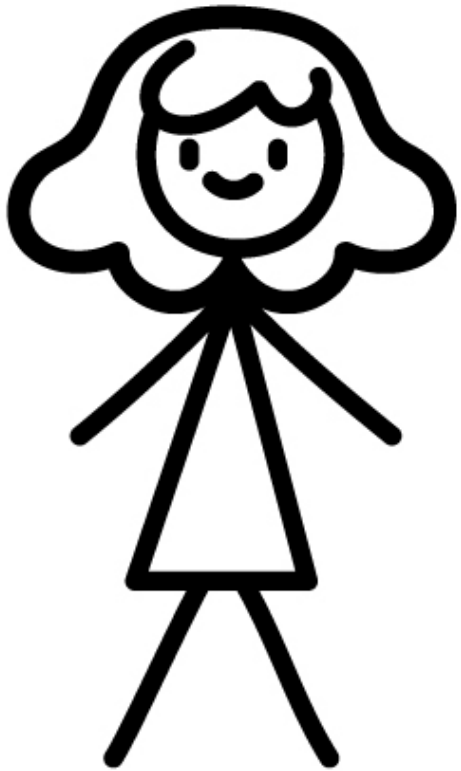
Caroline PASCAL

U2IS & UME – ENSTA Paris

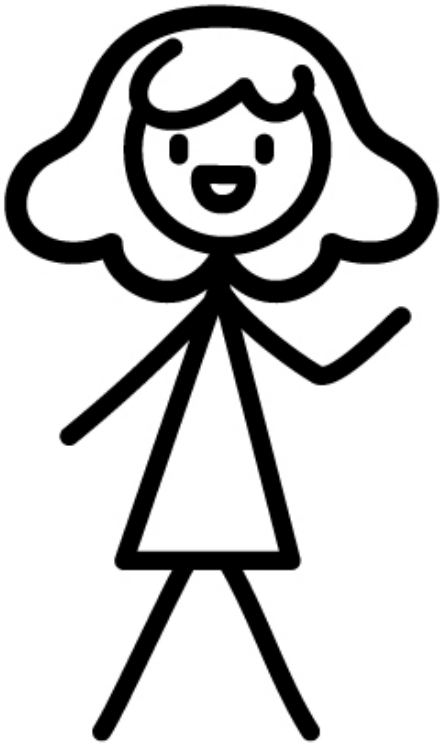
4 Juillet 2023



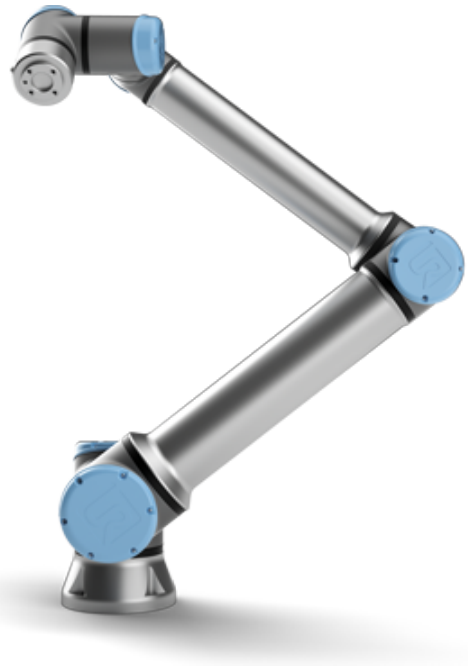
Introduction



This is Billie.



Billie recently started working with a robotic arm, and can't wait to take the most of its capabilities !



Universal Robots - UR10e

→ 6 degrees of freedom;

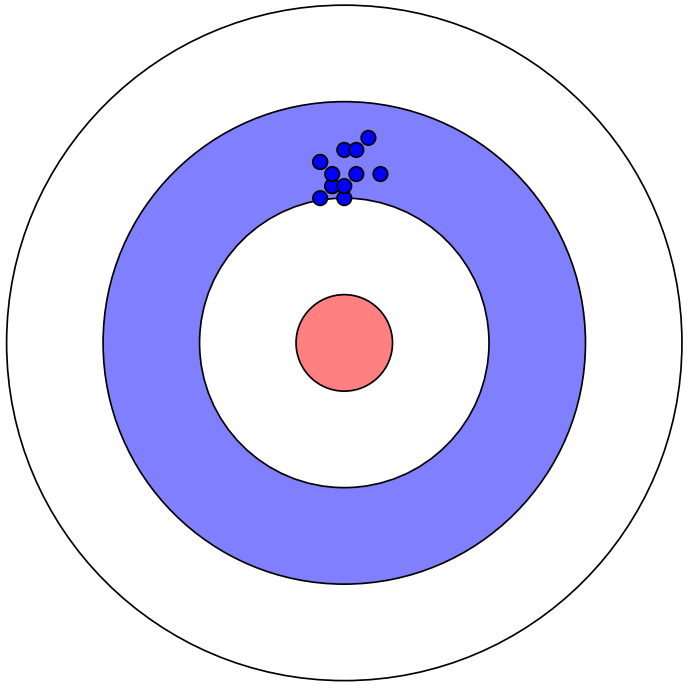
→ 12.5 kg payload;

→ 1.3 m reach;

→ **0.05 mm precision.**

Introduction

It would be shame if Billie knew that such robots suffer from a poor absolute accuracy...

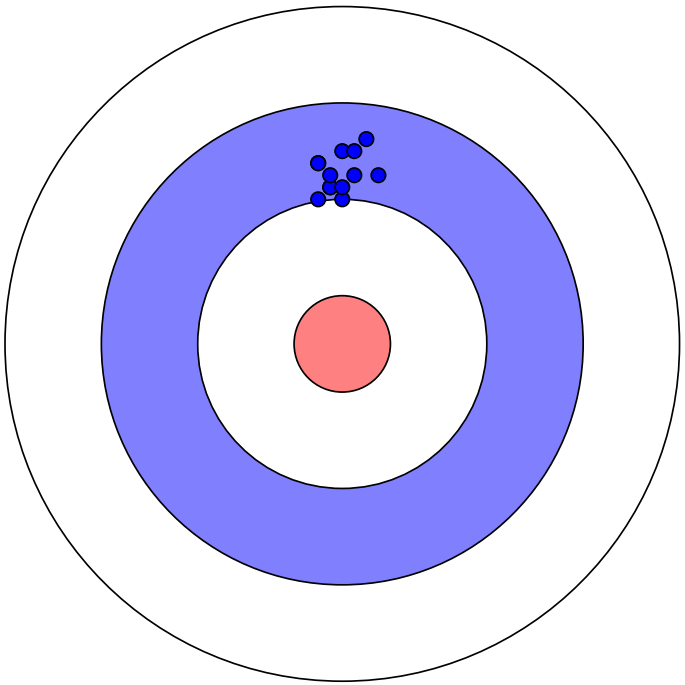


✓ Precise
✗ Accurate

→ **Centimetric accuracy !**

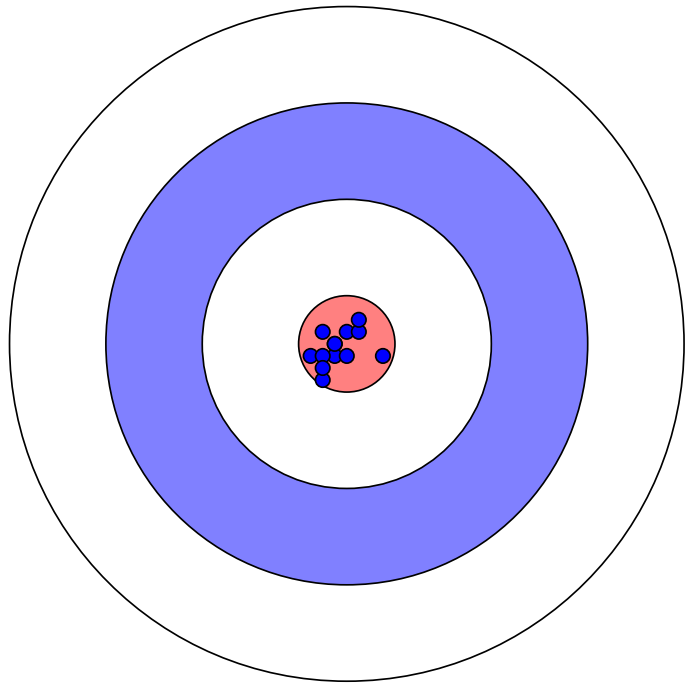
Introduction

Fortunately for Billie, there is a solution to this issue:
kinematic calibration !



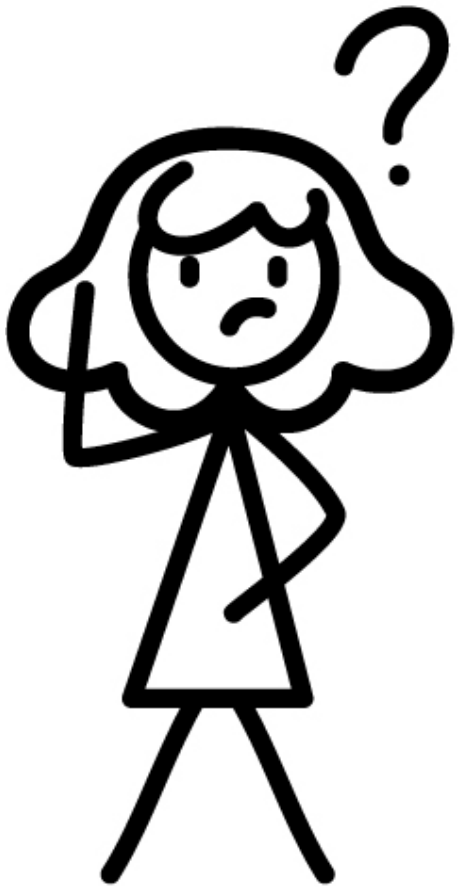
✓ Precise
✗ Accurate

KINEMATIC CALIBRATION



✓ Precise
✓ Accurate

Introduction



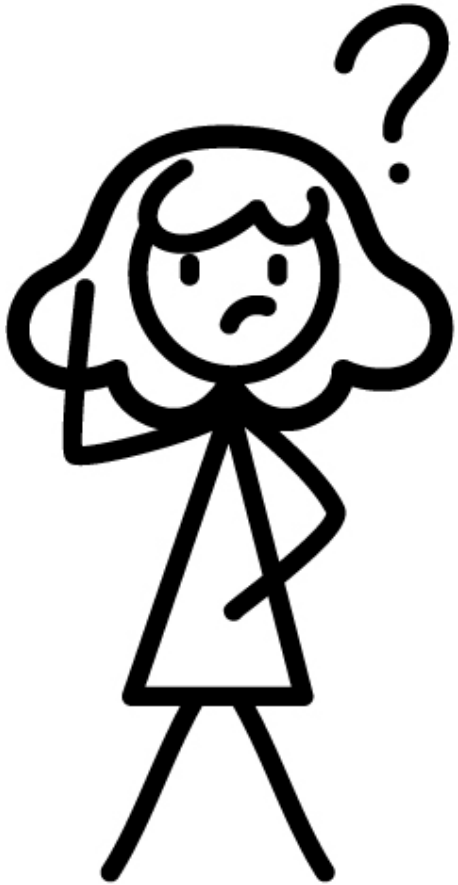
Yet, Billie is a bit lost, but no wonder : calibration is often a burden.

→ What is kinematic calibration ? How does it work ?

→ How do I manage hardware, software/ algorithmic interfaces ?

→ **Isn't there already a all-in-one and user-friendly tool for that ?!**

Introduction



Yet, Billie is a bit lost, but no wonder :
calibration *was* often a burden.

→ What is kinematic calibration ? How does it work ?

→ How do I manage hardware, software/algorithmic interfaces ?

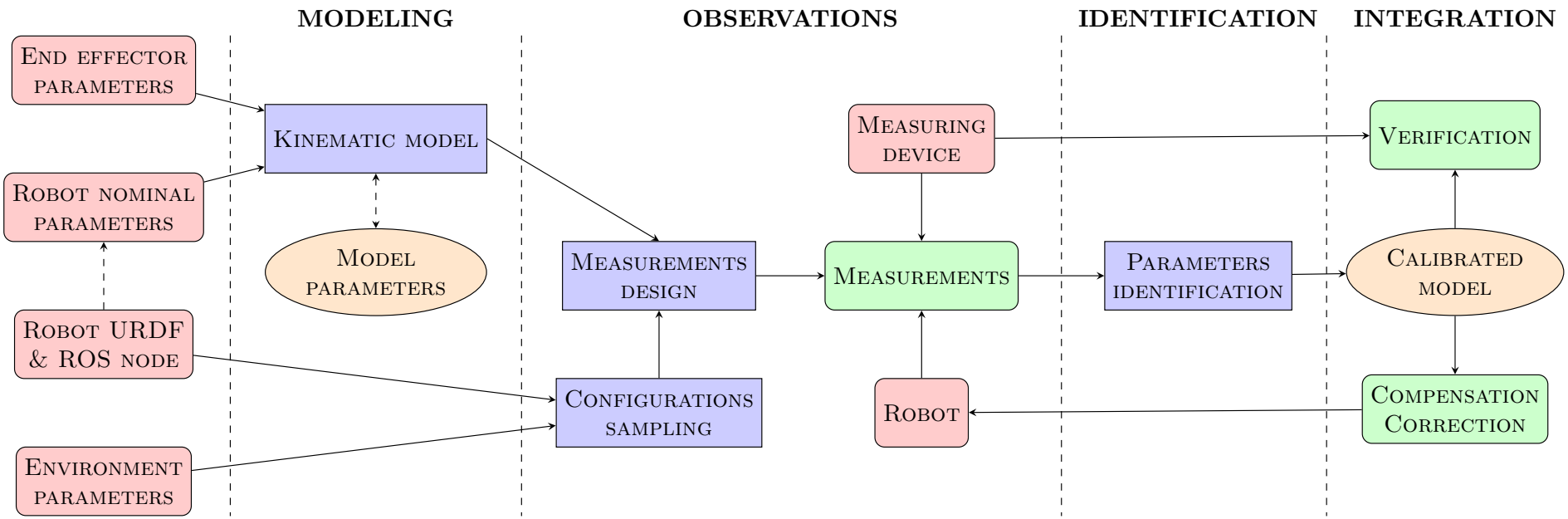
→ **Isn't there already a all-in-one and user-friendly tool for that ?!**

robot_arm_calibration

Table of Contents

Table of Contents

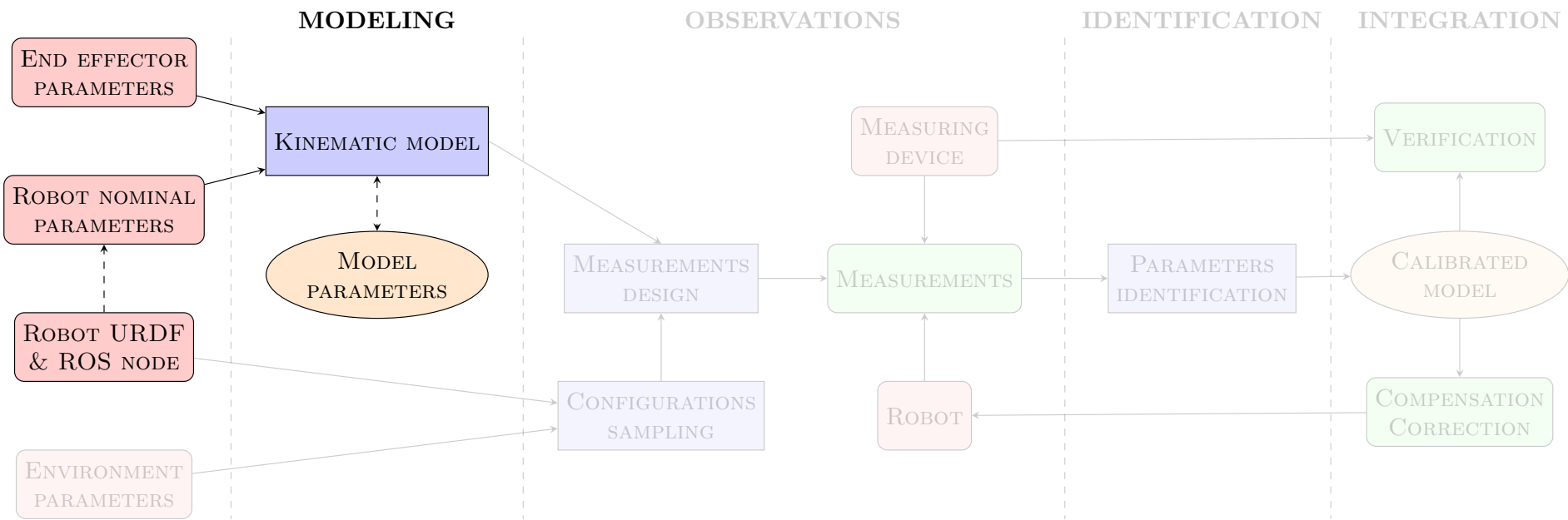
The 4 steps of kinematic calibration



Kinematic calibration procedure description

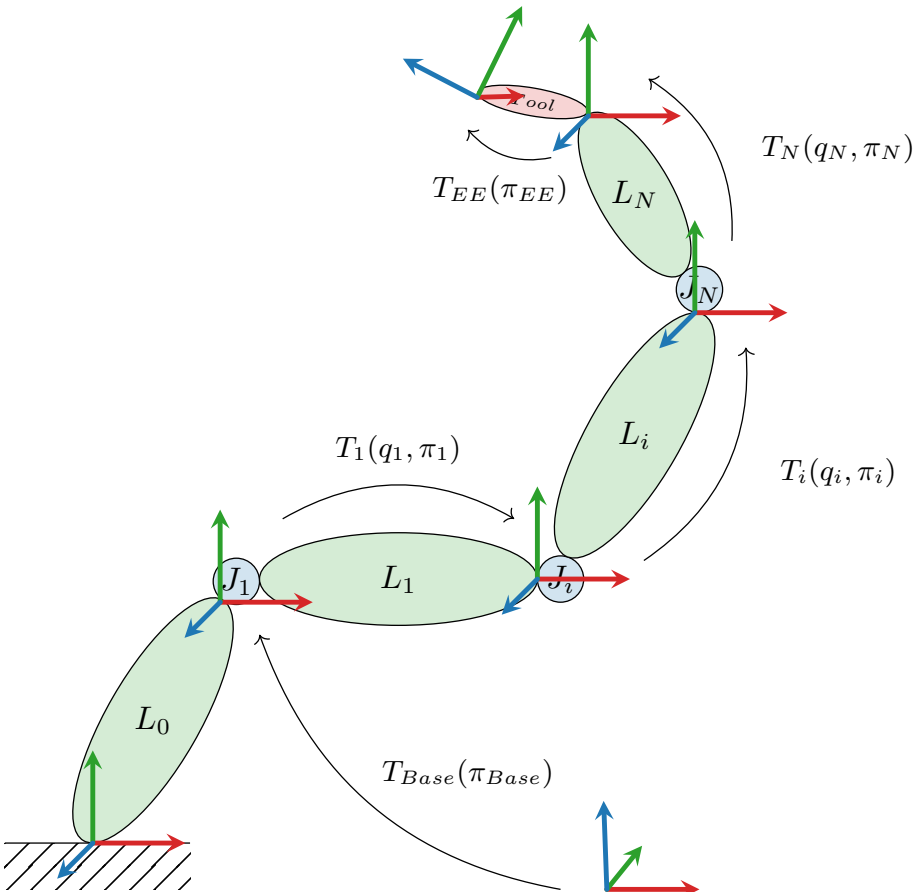
$\Longrightarrow$ All 4 steps are fully implemented (`python` & `C++`) and clearly identified in the `robot_arm_calibration` package.

Step 1 : Modeling



Kinematic calibration procedure description - Modeling

Step 1 : Modeling



Seeked model properties

→ Faithfullness;

↪ Compliance with the robot true behaviour;

→ Completeness, but without any redundancy;

↪ Model defined by a determined set of parameters;

→ Continuity;

↪ Continuous function of the parameters;

Step 1 : Modeling

Full-pose geometric modeling

$$T(q,\pi) = T_{Base}(\pi_{Base}) \cdot [T_{Joint_1}(q_1,\pi_{J_1}) \cdot T_{Link_1}(\pi_{L_1})] \cdot [T_{Joint_N}(q_N,\pi_{q_N}) \cdot T_{Link_N}(\pi_{L_N})] \cdot T_{EE}(\pi_{EE})$$

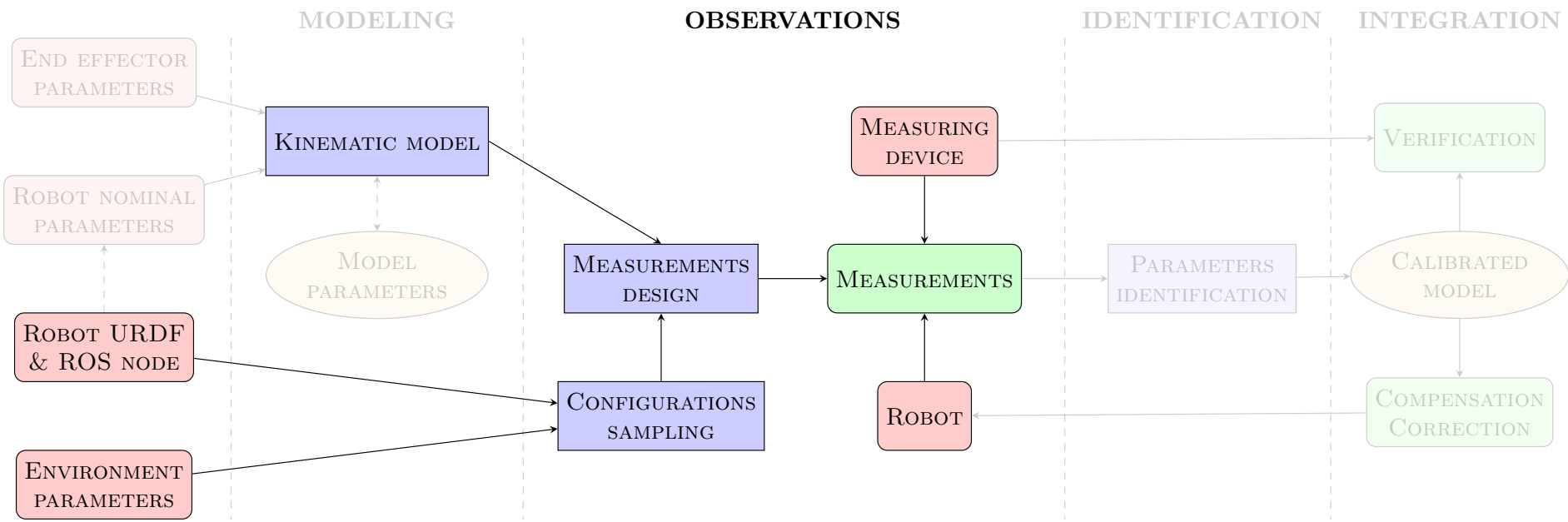
Where $\pi = (\pi_{Base}, \pi_{J_i}, \pi_{L_i}, \pi_{EE})$ are the *kinematic model parameters*

Partial-pose geometric modeling [1]

$$\left(P^i(q,\pi)\right)_{i=1...M} = T_{Base}(\pi_{Base}) \cdot [T_{Joint_1}(q_1,\pi_{J_1}) \cdot T_{Link_1}(\pi_{L_1})] \cdot [T_{Joint_N}(q_N,\pi_{q_N}) \cdot T_{Link_N}(\pi_{L_N})] \cdot T_{EE_i}(\pi_{EE_i})$$

Where EE_i refers to the end effector point $i \in \{1...M\}$

Step 2 : Observations



Kinematic calibration procedure description - Observations

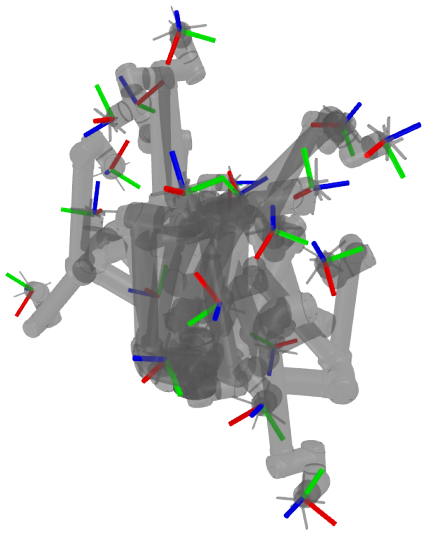
Step 2 : Observations

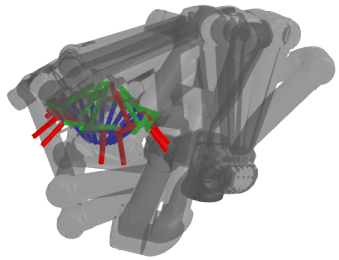
→ Open-loop observations, using internal monitoring (joints encoders) and external measurements (end effector points positions).

↪ *How to pick the measured robot configurations ?*

1) Perform a reachable and task-oriented sampling of the robot workspace;

Random sampling

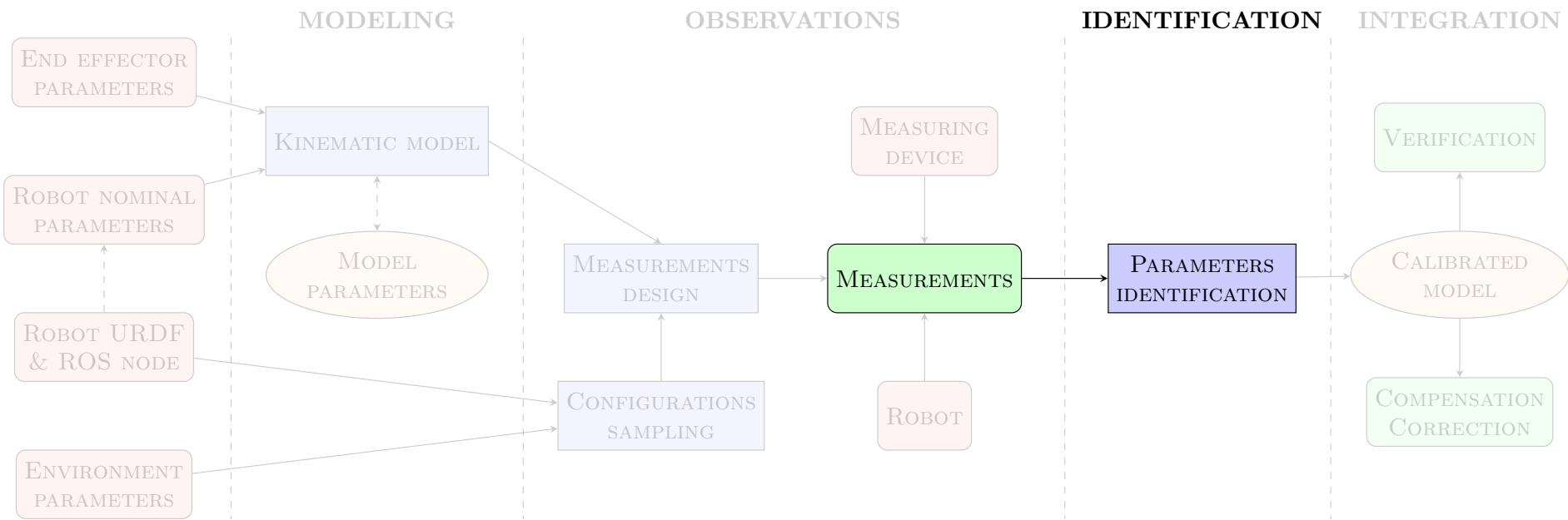




3D scan inspired sampling

2) Choose the configurations maximizing parameters identifiability.

Step 3 : Identification



Kinematic calibration procedure description - Identification

Step 3 : Identification

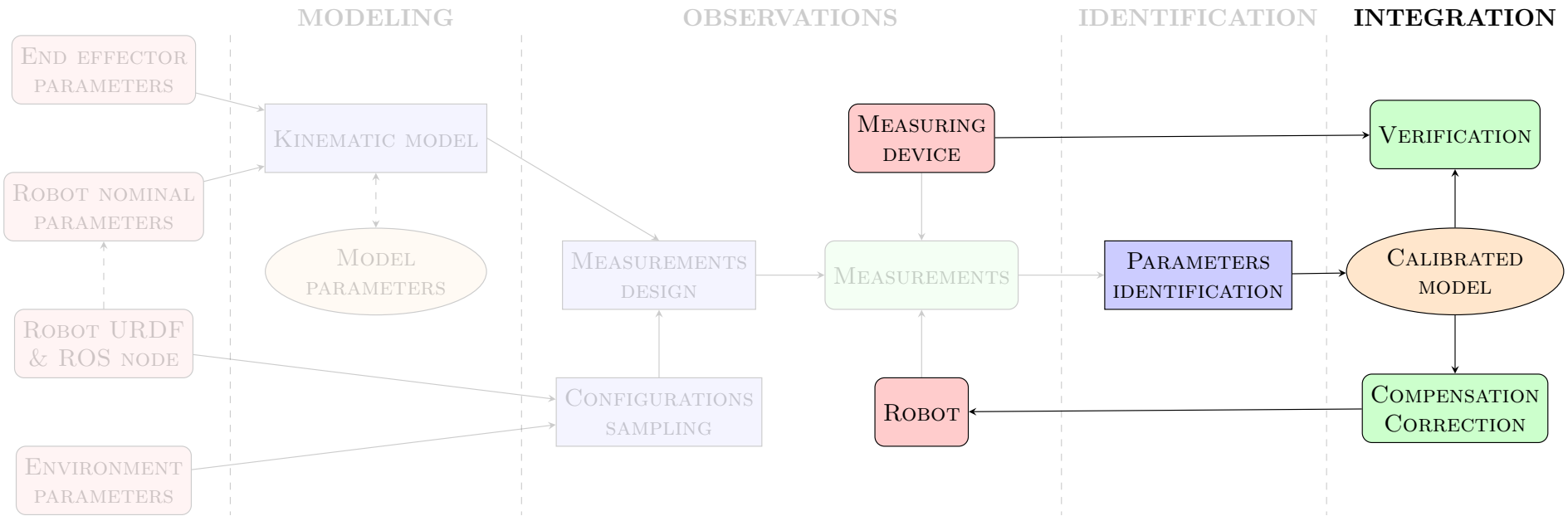
Partial-pose model identification

$$\pi^* = \arg\min_{\pi} \sum_{i=1}^{N_{\text{meas.}}} \sum_{j=1}^{N_{EE}} \underbrace{\left\| P^j(q_i, \pi) - P^j_{\text{measured}}(q_i) \right\|^2}_{\epsilon_i^j}$$

Where ϵ_i^j defines the *positioning error* of the end effector point j for the i-th measurement.

$\rightarrow$ The sum of positionning errors over all end effector points and measurement configurations defines the robot accuracy.

Step 3 : Integration



Kinematic calibration procedure description - Integration

Step 4 : Integration

Verification

→ Validate the calibrated parameters with the accuracy obtained on a new set of randomly picked measurements.

Integration

→ Compensation

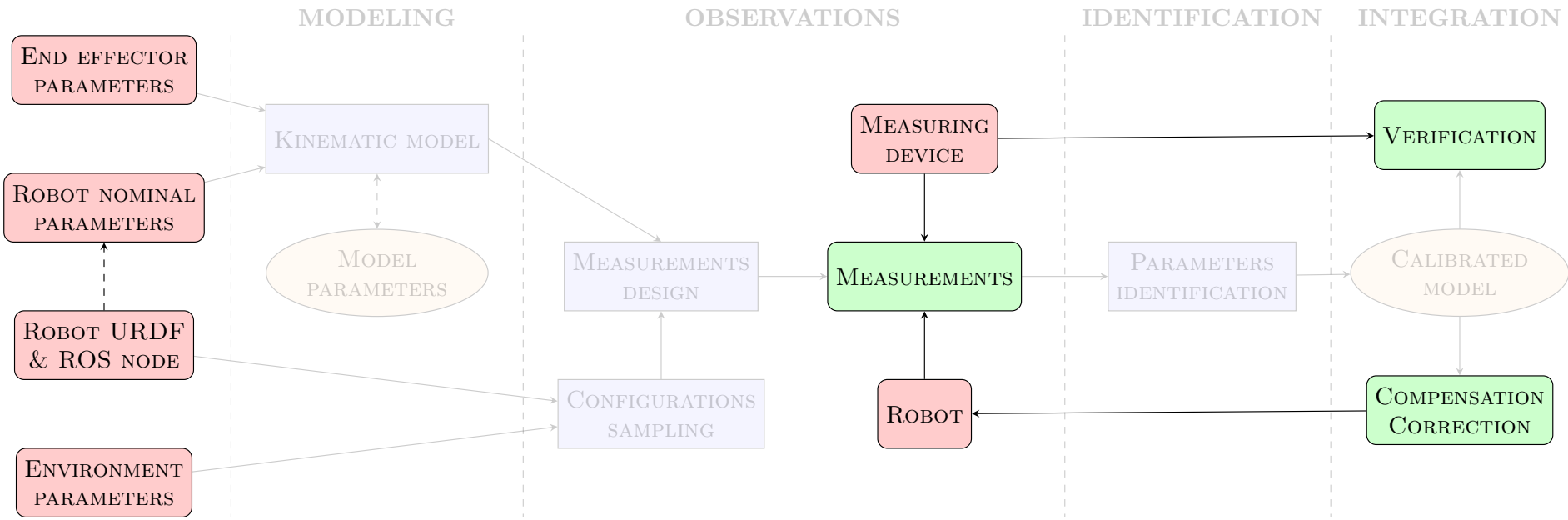
↪ Integrate the calibrated parameters directly in the robot controller.

→ Correction

↪ Build a new robot description with the calibrated parameters.

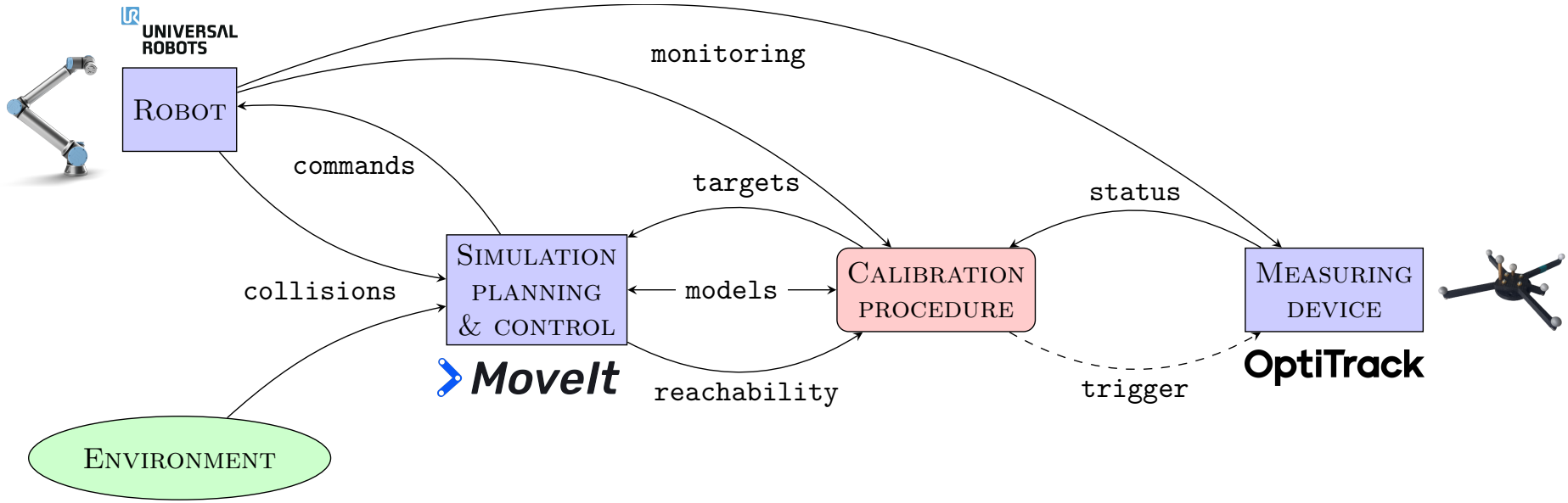
Table of Contents

Hardware/Software interfaces - In theory



Kinematic calibration procedure description - H/S Interfaces

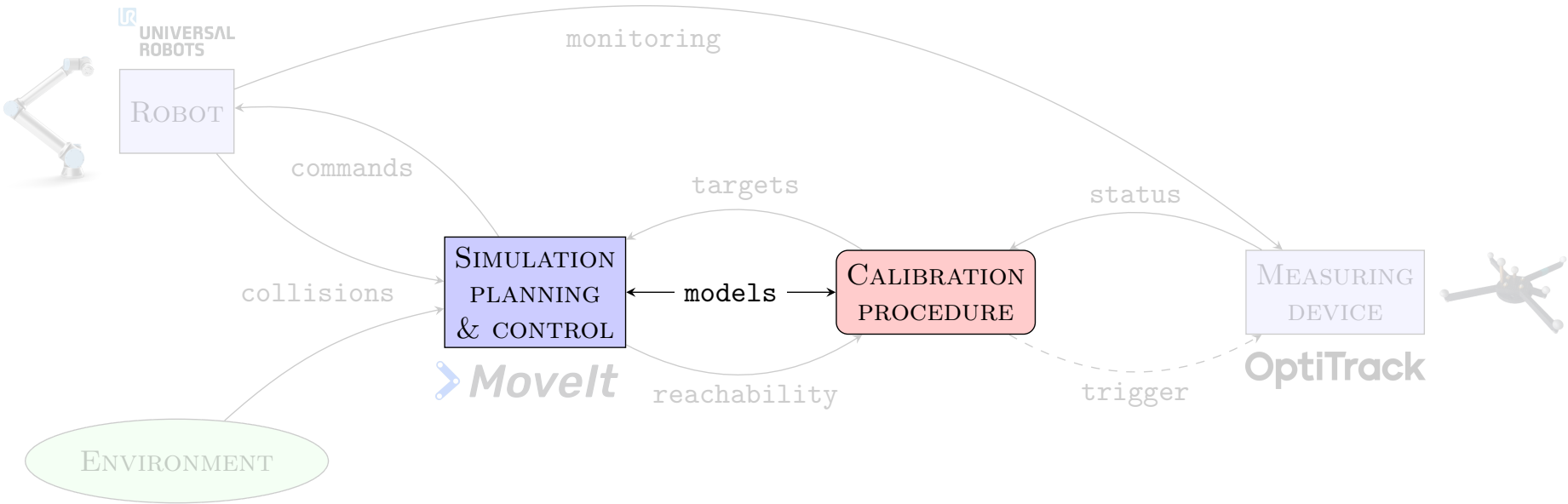
Hardware/Software interfaces - In practice



Hardware/Software interfaces overview

$\Longrightarrow$ All interfaces are handled as smoothly and generically as possible in the `robot_arm_calibration` package.

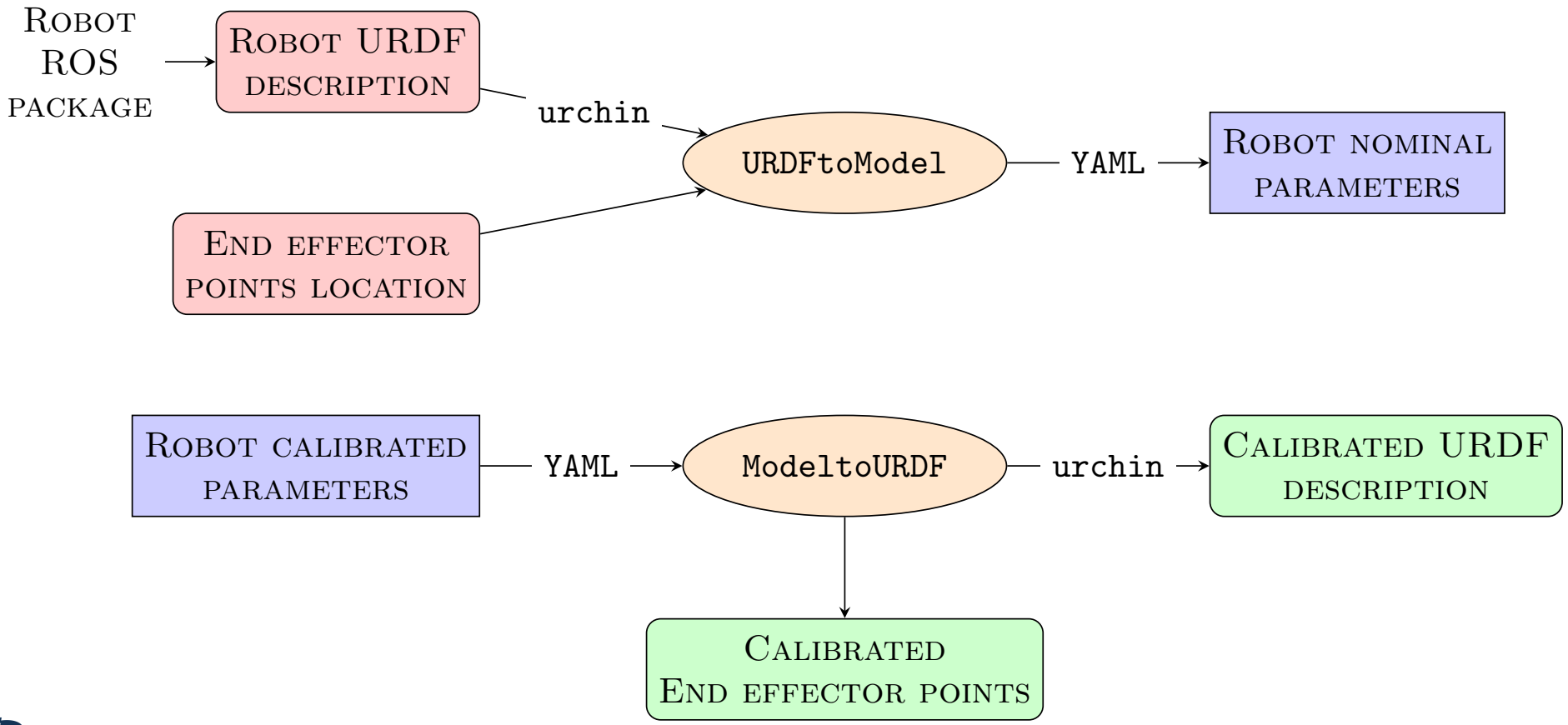
Model parameters integration



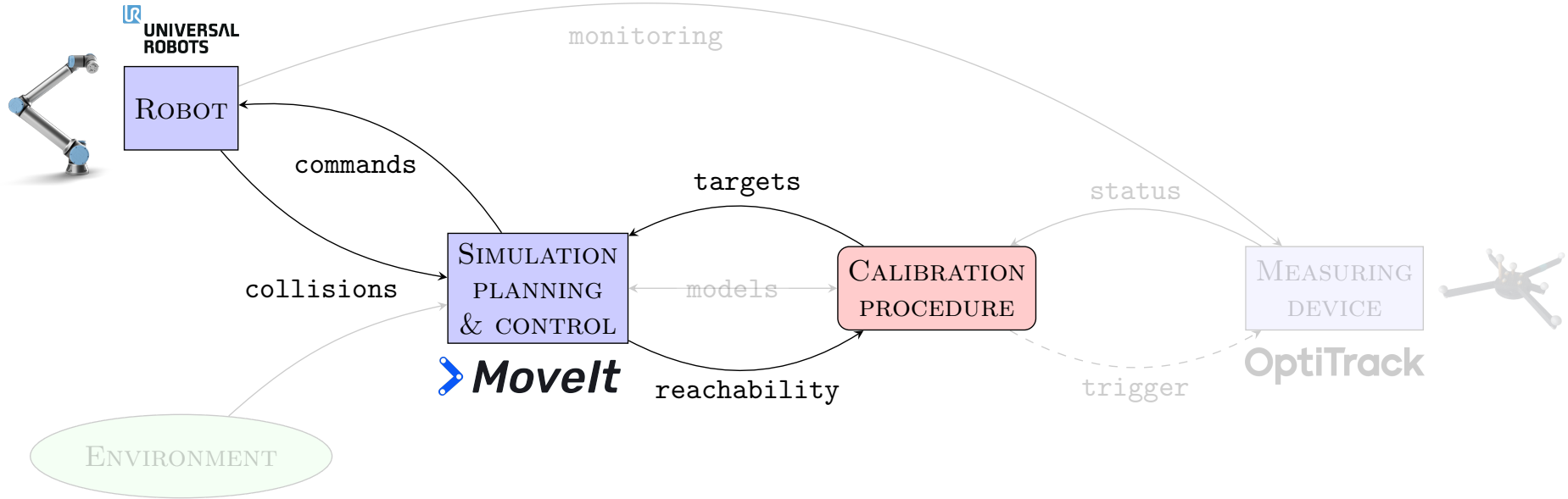
Hardware/Software interfaces overview - Model parameters

Model parameters integration

→ Automated URDF parsing and generation via standardized YAML files.



Robot integration



Hardware/Software interfaces overview - Robot

Robot integration

→ *MoveIt* motion planning framework API
+ custom overlay : `robot_arm_tools`

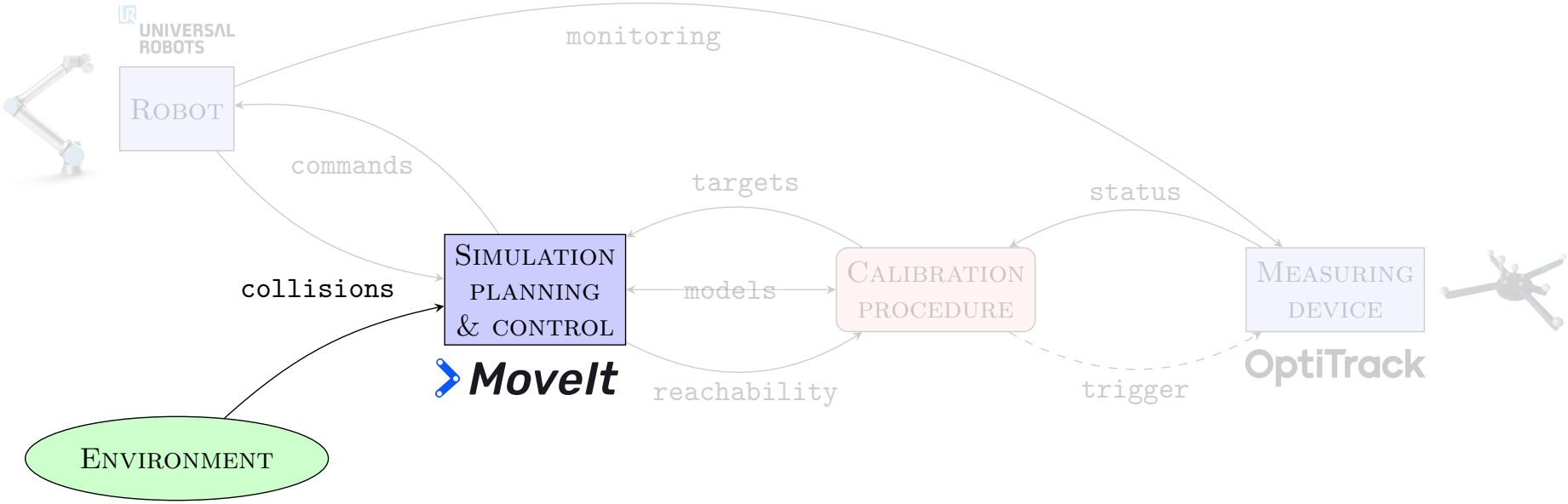
Simulation

- *Collision* and *singularity* aware motion planning tool, providing *reachability* insights;
- *Modular* yet *generic* planning and kinematic pipelines;
- A simplified definition and integration of common use-cases;

Real robot

- *User-friendly* and *high-level* integration of real robots ROS controllers;
- Dynamic interruption and low-level recovery of motion execution;
- A logging solution to monitor and recover multiple waypoint trajecotry execution.

External environment integration

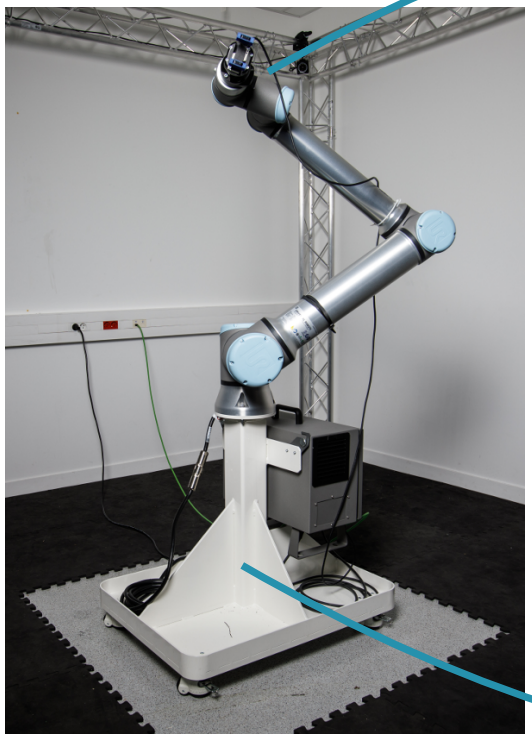


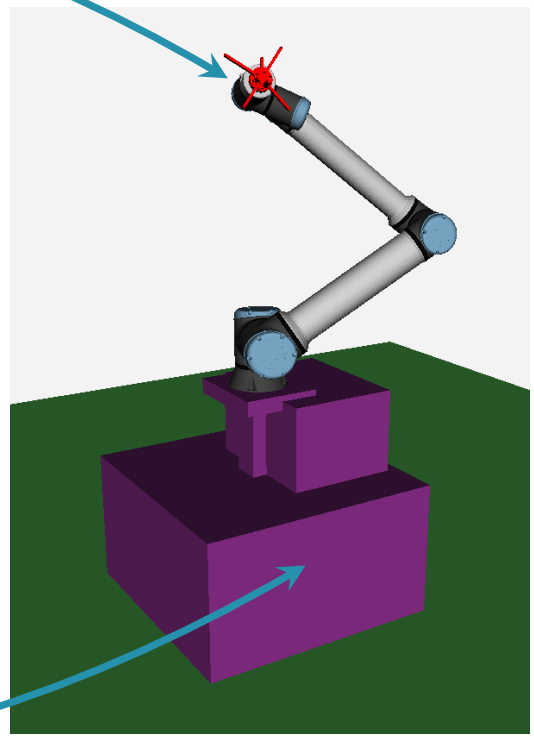
Hardware/Software interfaces overview - External environment

External environment integration

Automated `URDF` generation from `STL` mesh file

↪ `/robot_description`
tweakage

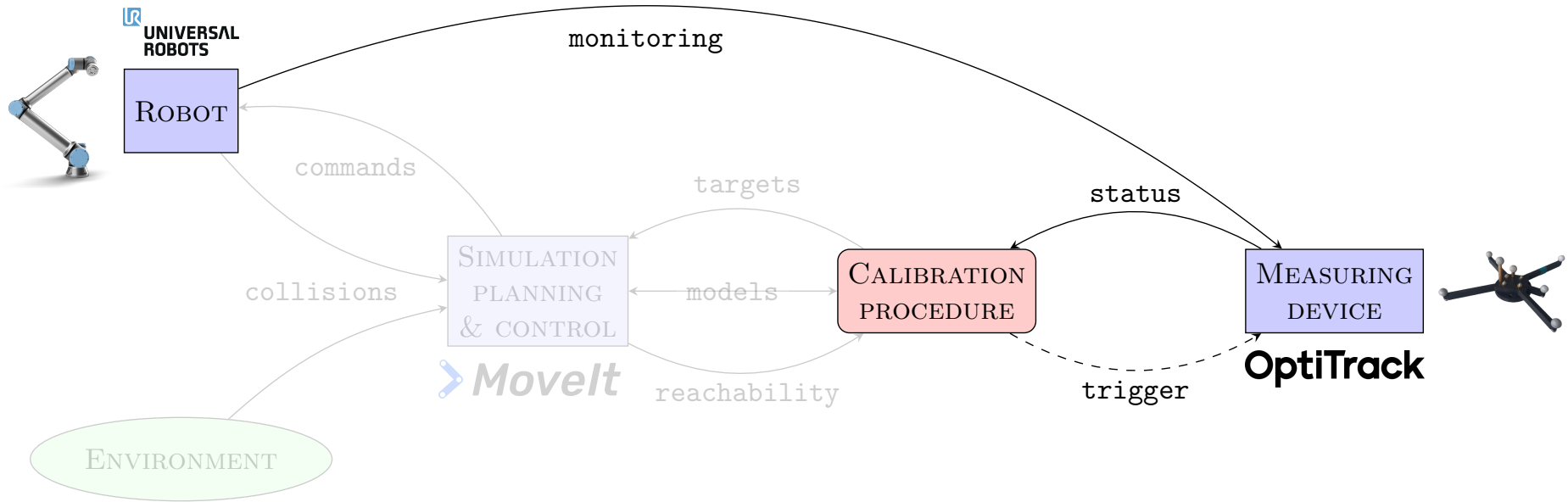




User defined `YAML` file with basic geometric primitives

↪ `moveit_visual_tools`
overlay

Measuring devices integration



Hardware/Software interfaces overview - Measuring devices

Measuring devices integration

→ Creation of a generic ROS service *MeasurementService* for motion and measurements synchronization.

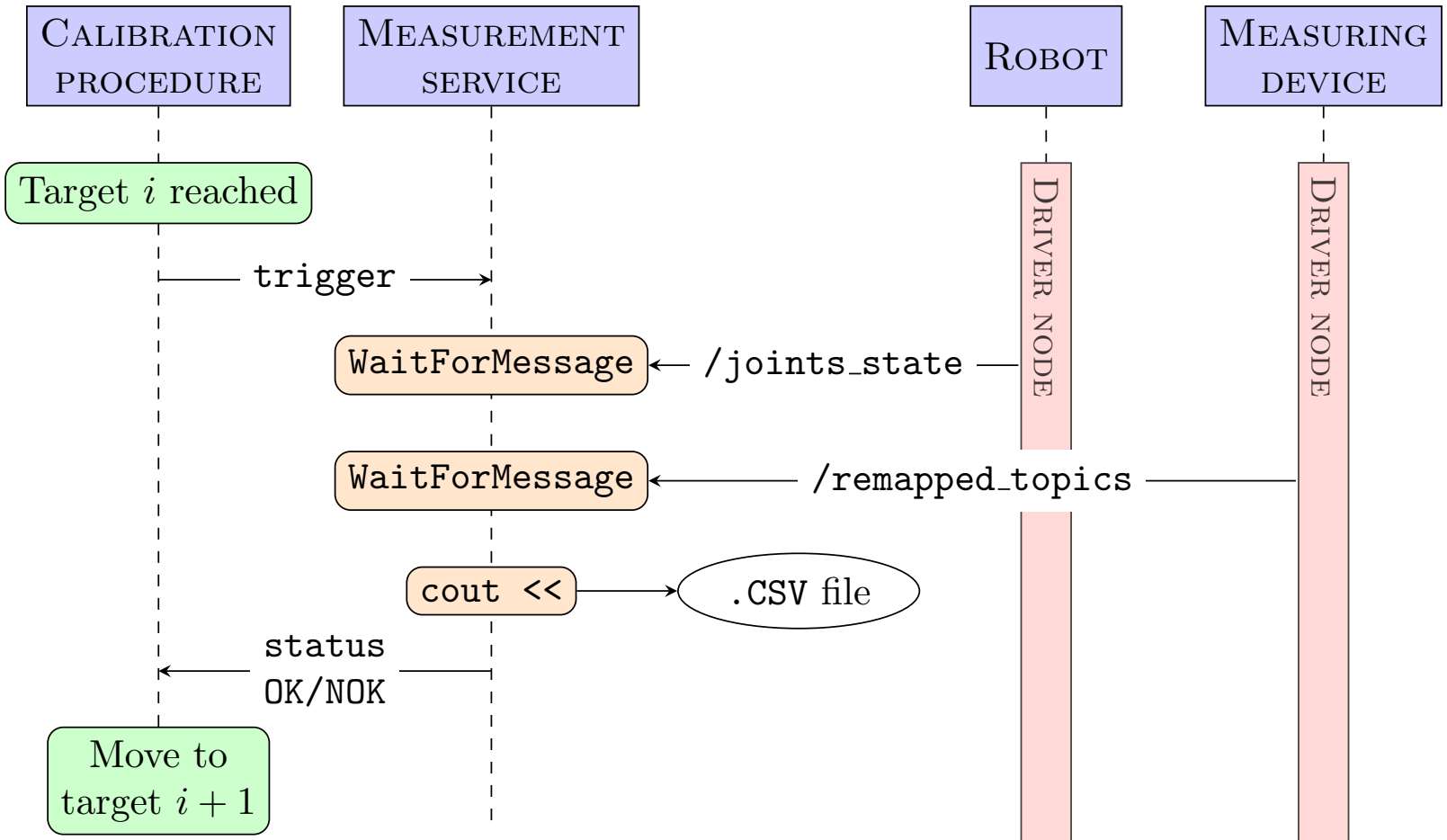
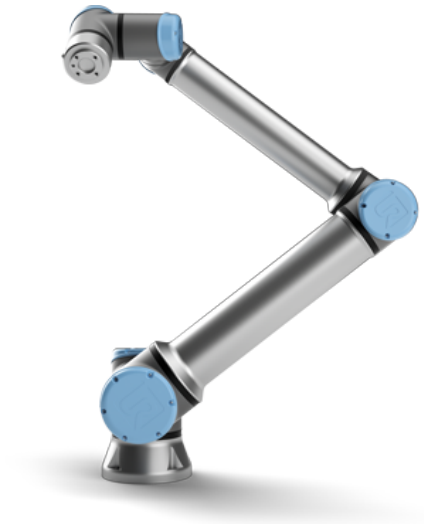


Table of Contents

Experimental validation setup - Robots



Universal Robots - UR10e

→ 6 axis

→ 18 geometric parameters

→ ROS packages :
- `universal_robots`
- `ur_robot_driver`

Franka Emika - Panda

→ 7 axis

→ 22 geometric parameters

→ ROS packages :
- `panda_moveit_config`
- `franka_ros`

Experimental validation setup - External measuring device

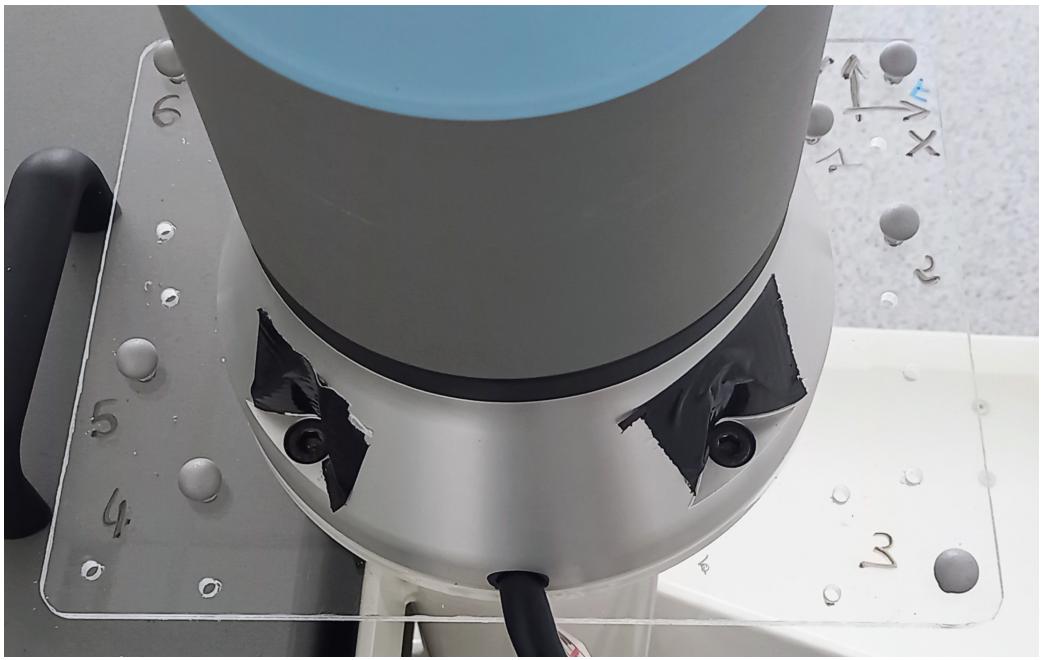
→ Optitrack position tracking tool.

↪ 6 *Prime 13* cameras

$\Longrightarrow$ ± 0.2 *mm* accuracy.

↪ ROS package : `mocap_optitrack`

Experimental validation setup - Measurements bodies



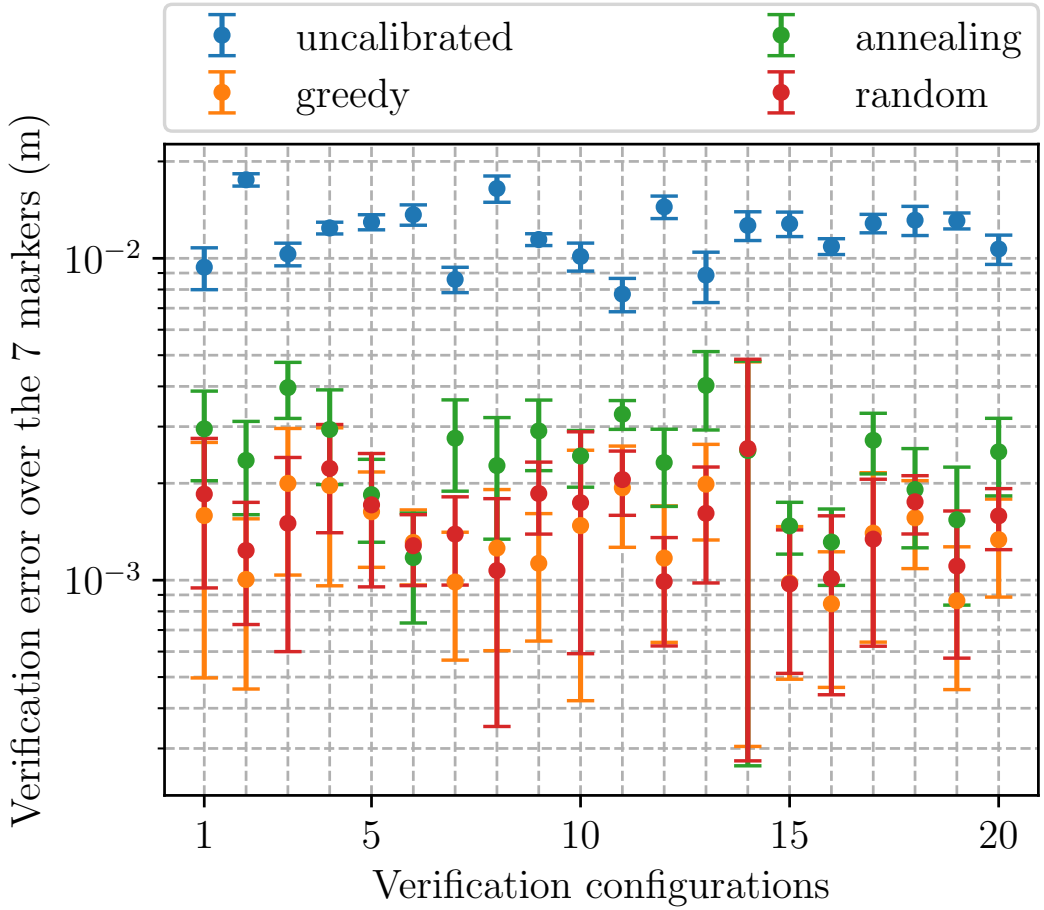
→ Robot base and end effector points measurements bodies.
7 reflective spheres in a precise 3D layout.

Experimental validation setup

Calibration measurements of an UR10e (speed $\times 2$)

Experimental validation - Results (Panda)

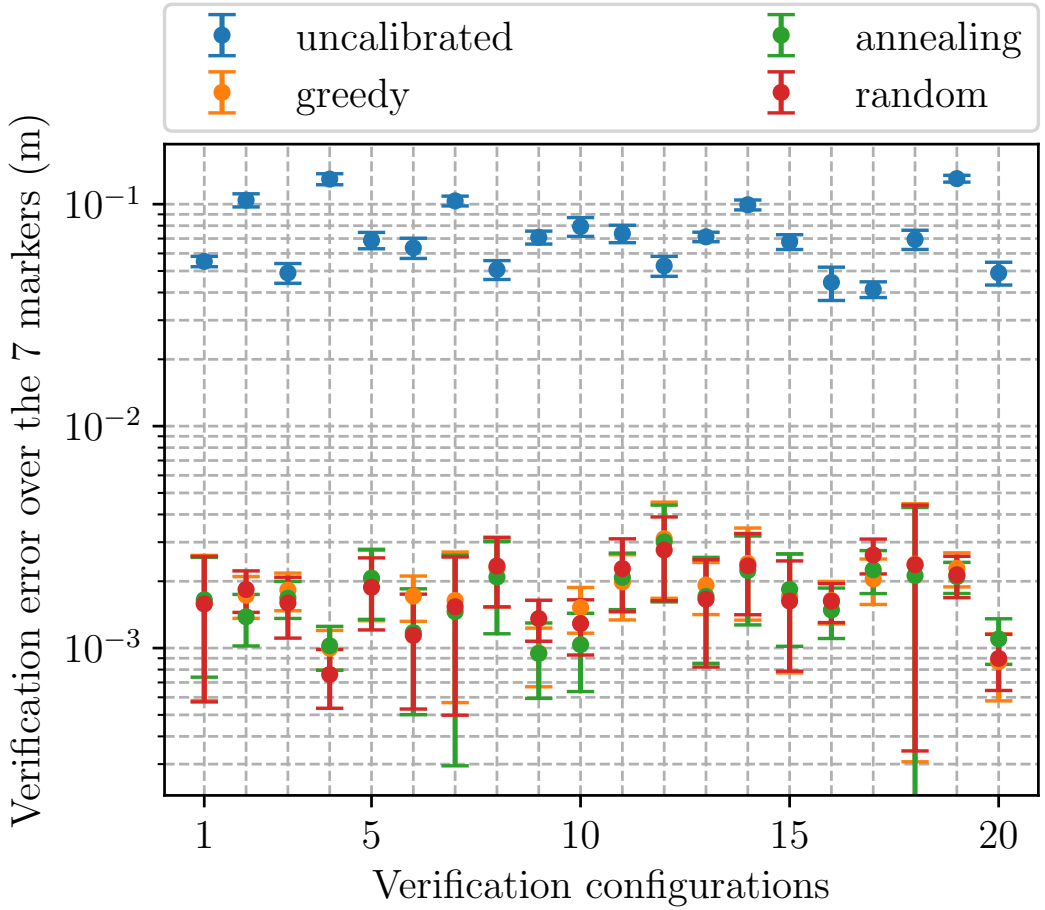
	F.E. Panda
Total parameters	49
Measurements configurations	98
Overall duration (h)	4.38
Modeling (min)	55
Measurements (h)	3.1
Identification (min)	22
Initial accuracy (mm)	12.3
Best final accuracy (mm)	1.7
Improvement rate	86.0%



Average positioning errors over the 7 markers with and without calibration

Experimental validation - Results (UR10e)

	UR10e
Total parameters	45
Measurements	90
configurations	
Overall duration (h)	7.03
Modeling (min)	45
Measurements (h)	6.0
Identification (min)	17
Initial accuracy (mm)	77.7
Best final accuracy (mm)	2.0
Improvement rate	97.4%



Average positioning errors over the 7 markers with and without calibration

Table of Contents

Conclusion

How to sum up `robot_arm_calibration` in 3 points ?

→ A **complete kinematic calibration procedure** of serial robots, including modeling, measurements, identification, verification and integration steps;

→ A **ROS-package**, providing generic and user-friendly hardware/software interfaces for a simplified set up of any robot or measuring device;

→ An **open-access project**, whose main intent is to leave as much room as possible for further algorithmic improvements and customization.

And what next ?

→ **Work on ROS 2 migration !**

↪ Move up from ROS 1 *Noetic* to the latest ROS 2 release.

→ **Introduce actuator flexibilites in the kinematic model**

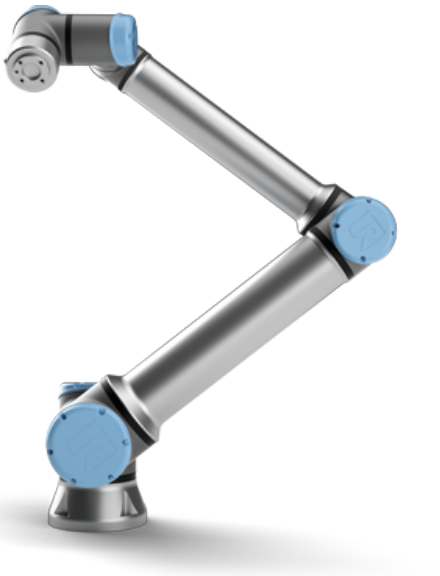
↪ Take the effects of gravity on the robot actuators into account [2], in a dedicated kinematics plugin.

→ **Allow full-pose and indirect measurements**

→ **Increase measurements robustness**

↪ Avoid robot induced obstructions while selecting measurements configurations thanks to planning constraints.

Thank you for your time and attention !



gitlab.ensta-paris.fr/caroline.pascal.2020/robot_arm_tools

gitlab.ensta-paris.fr/caroline.pascal.2020/robot_arm_calibration

References I

Yier Wu, Alexandr Klimchik, Stéphane Caro, Benoît Furet, and Anatol Pashkevich.
Geometric calibration of industrial robots using enhanced partial pose measurements and design of experiments.
Robotics and Computer-Integrated Manufacturing, 35:151–168, 2015.

Alexandr Klimchik, Yier Wu, Stéphane Caro, Benoît Furet, and Anatol Pashkevich.
Geometric and elastostatic calibration of robotic manipulator using partial pose measurements.
Advanced Robotics, 28, 2014.