Robot Calibration and Optimization

Dr. Nicholas Nadeau, Ph.D., P.Eng.
2022-10-15


KUKA
KUKA


AON3D

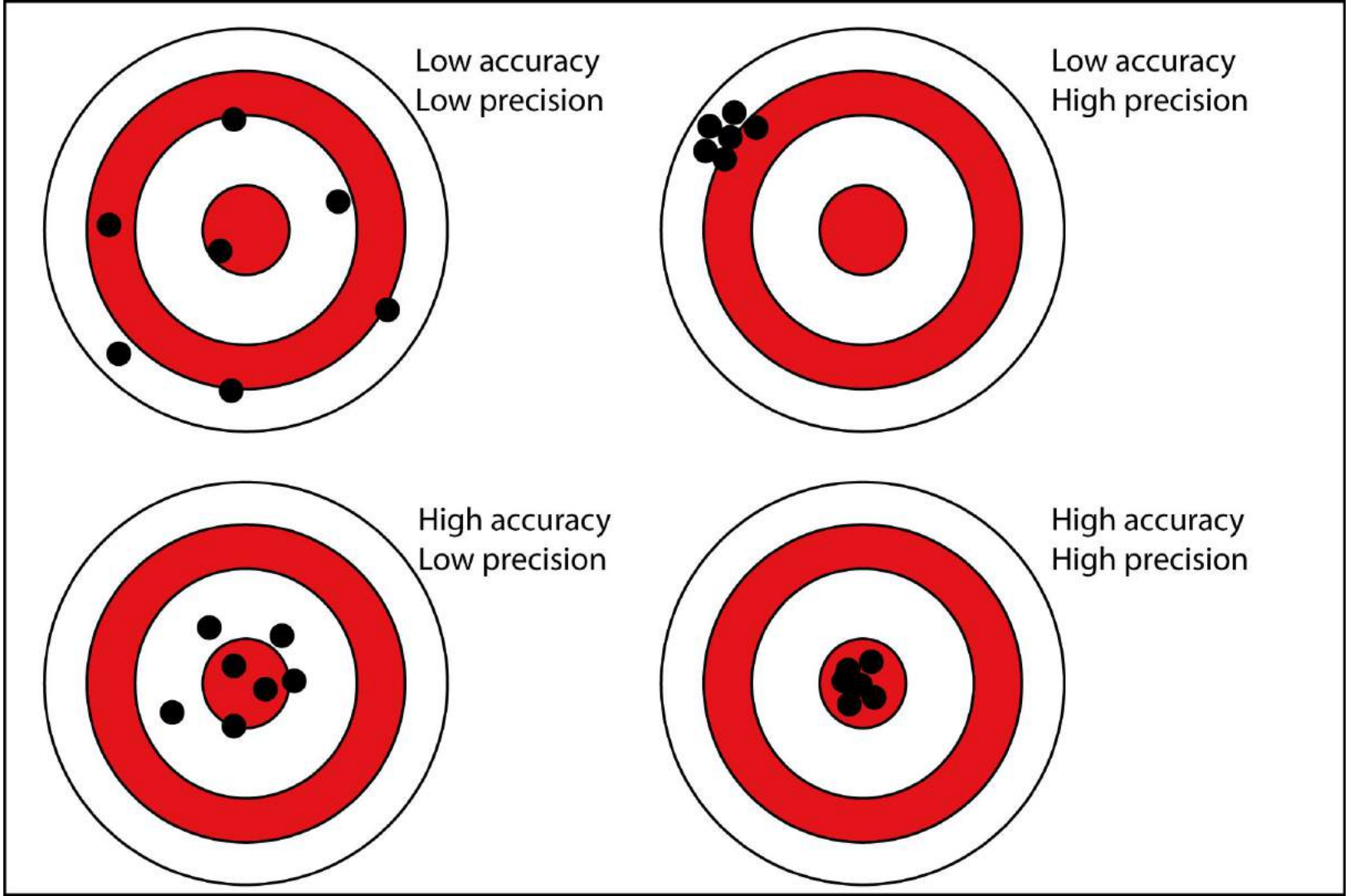
Today’s Talk

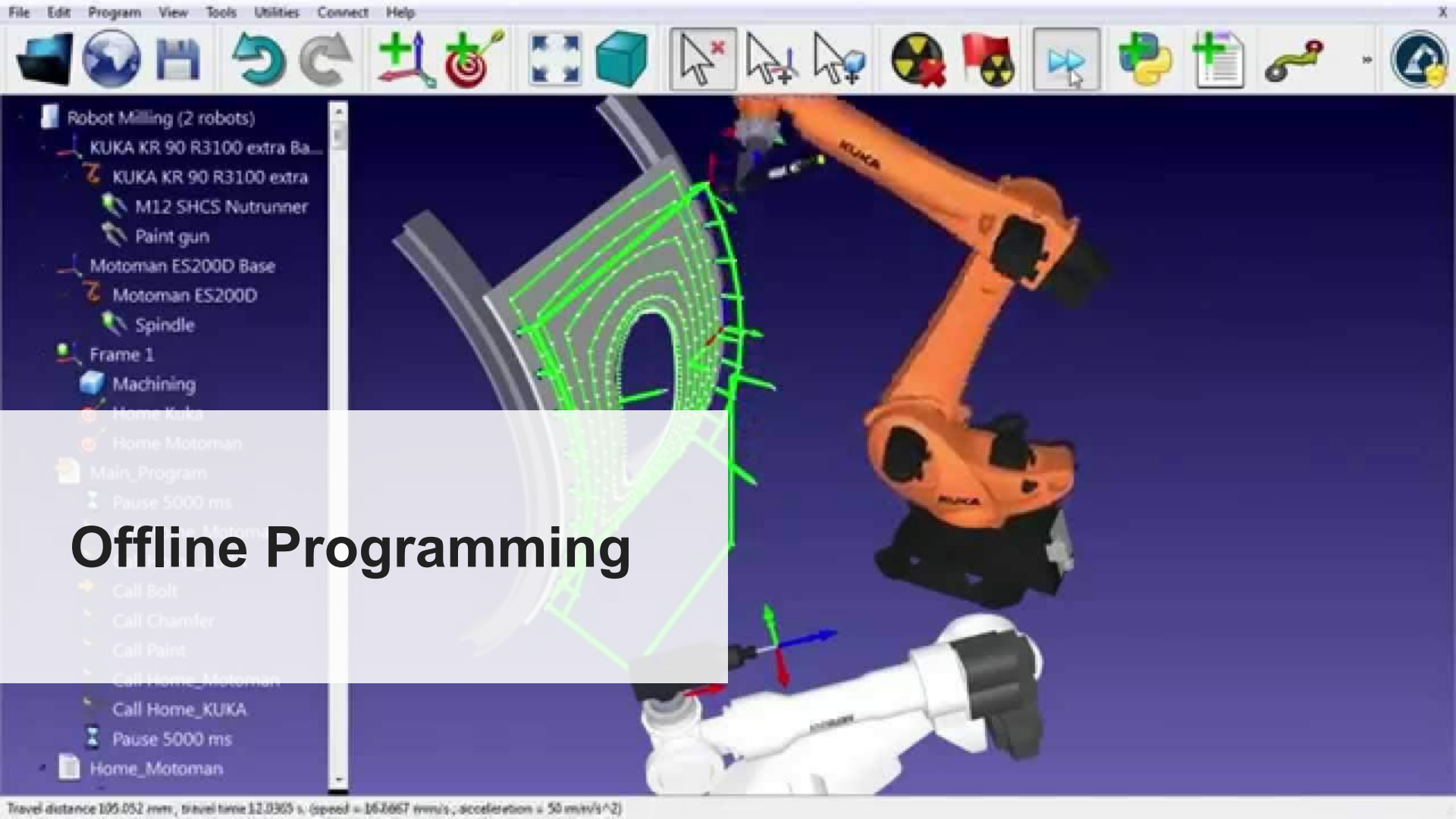
1. Pybotics and Calibration
2. Impedance Control Calibration
3. Hardware-in-the-loop Optimization Architecture
4. Evolutionary Motion Control Optimization

Pybotics and Calibration

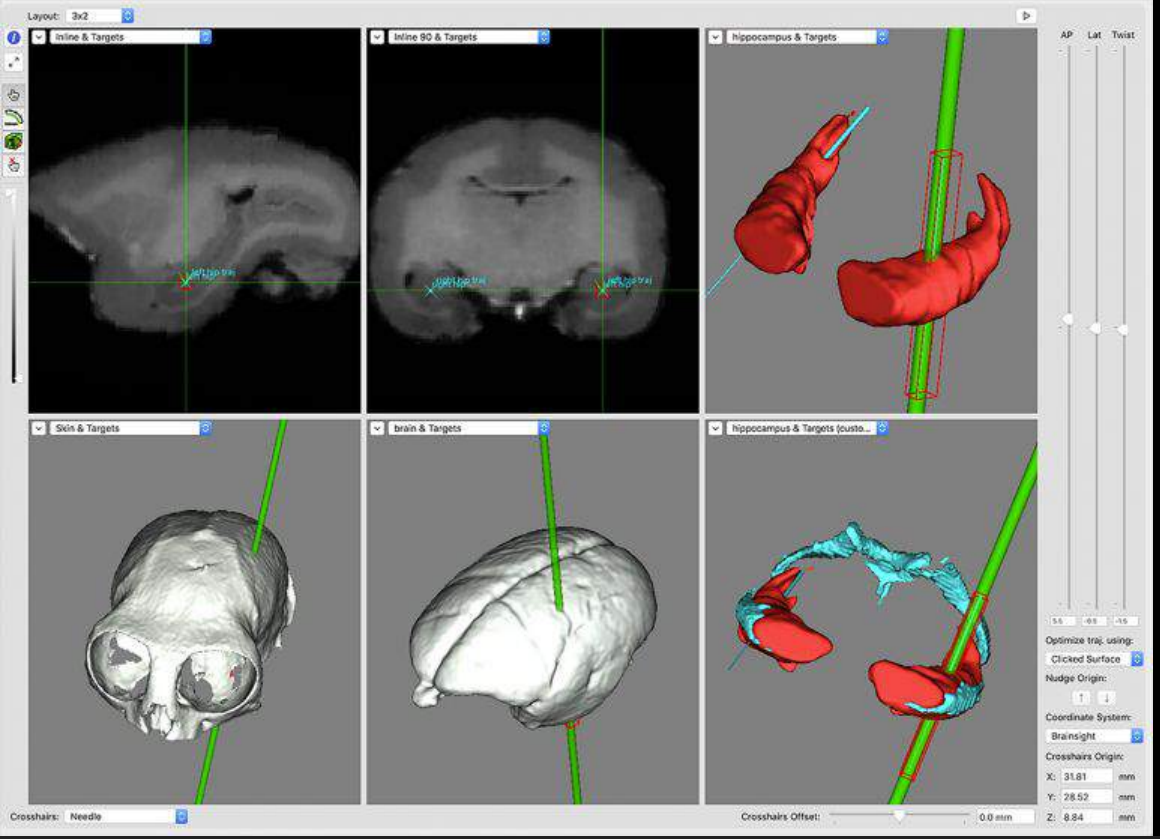
What’s the difference?

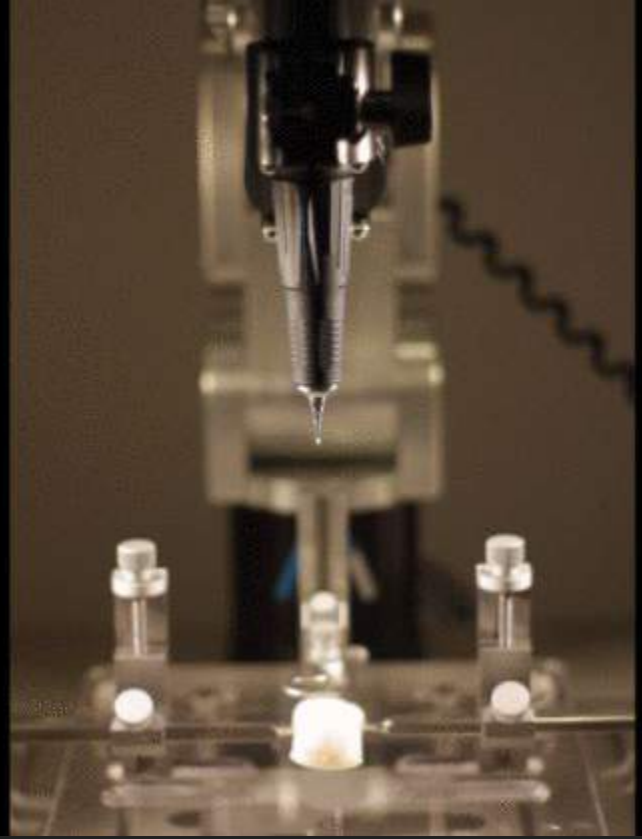
Why calibrate?


Low accuracy
Low precision
Low accuracy
High precision
High accuracy
Low precision
High accuracy
High precision



Offline Programming

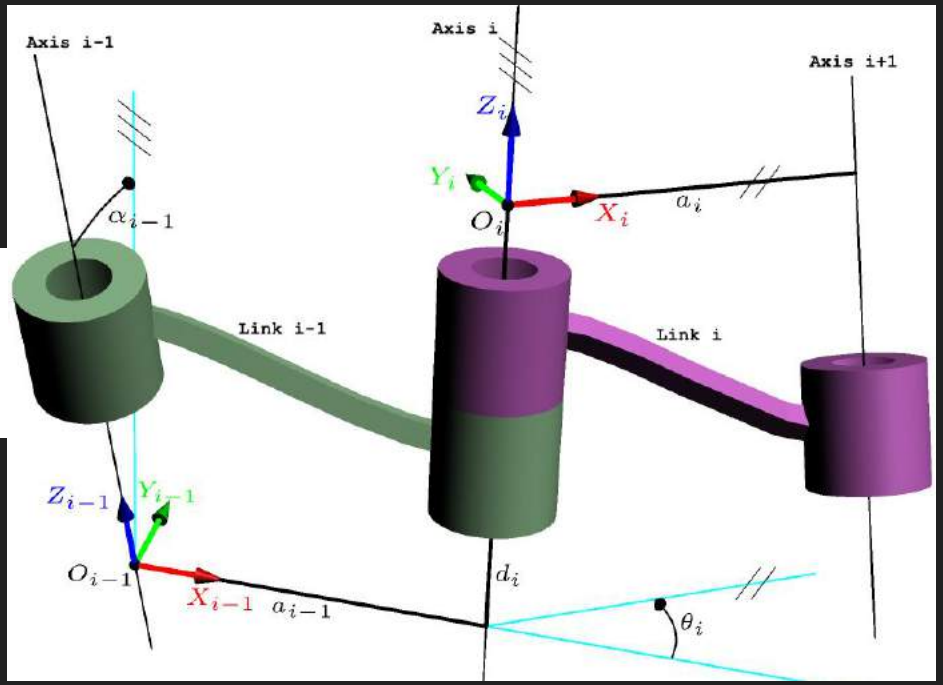




Robot Surgery

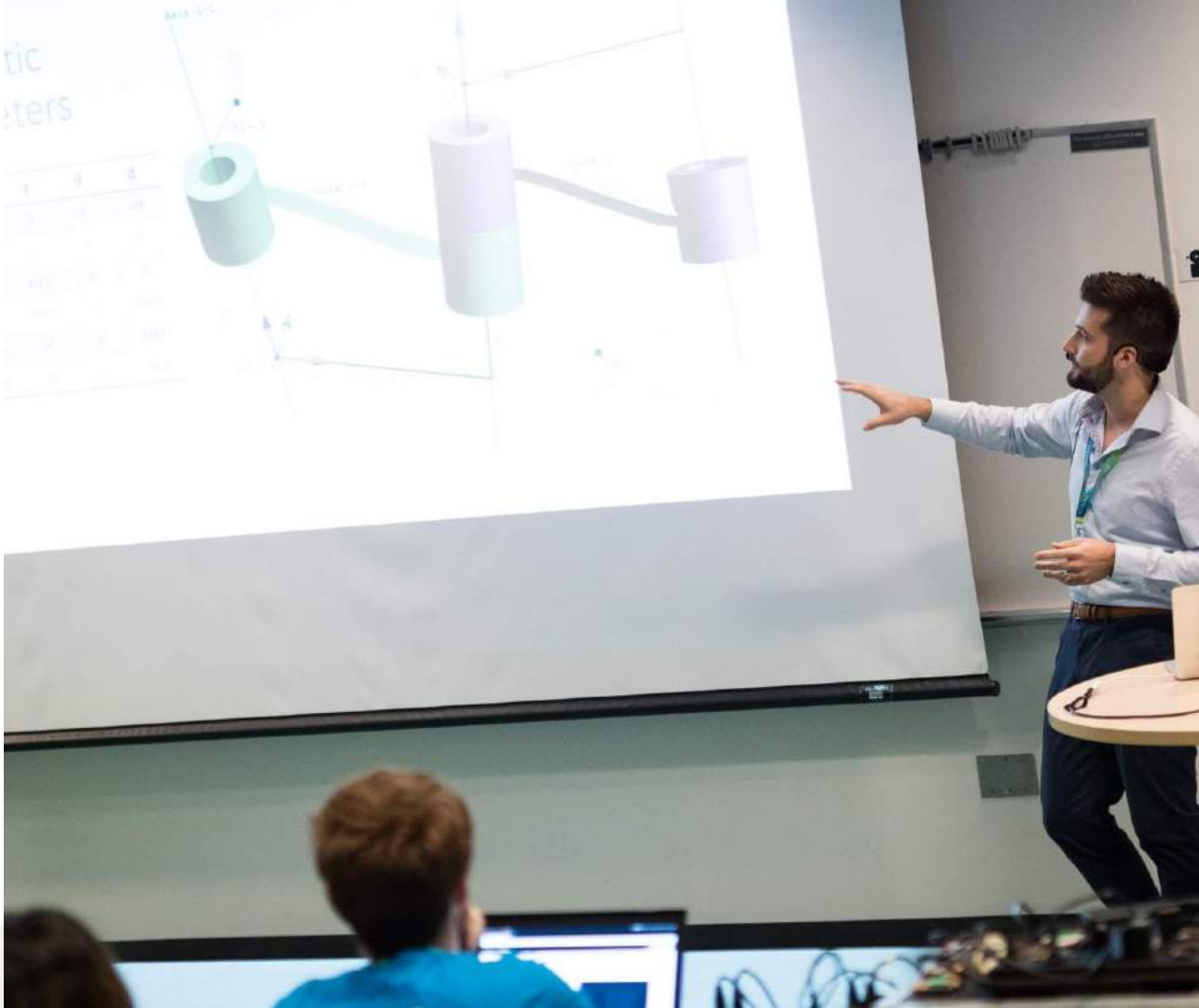
Kinematic Parameters

$$
{}^{n-1}T_n = \left[\begin{array}{cccc}
\cos\theta_n & -\sin\theta_n & 0 & a_{n-1} \\
\sin\theta_n\cos\alpha_{n-1} & \cos\theta_n\cos\alpha_{n-1} & -\sin\alpha_{n-1} & -d_n\sin\alpha_{n-1} \\
\sin\theta_n\sin\alpha_{n-1} & \cos\theta_n\sin\alpha_{n-1} & \cos\alpha_{n-1} & d_n\cos\alpha_{n-1} \\
\hline
0 & 0 & 0 & 1
\end{array}\right]
$$

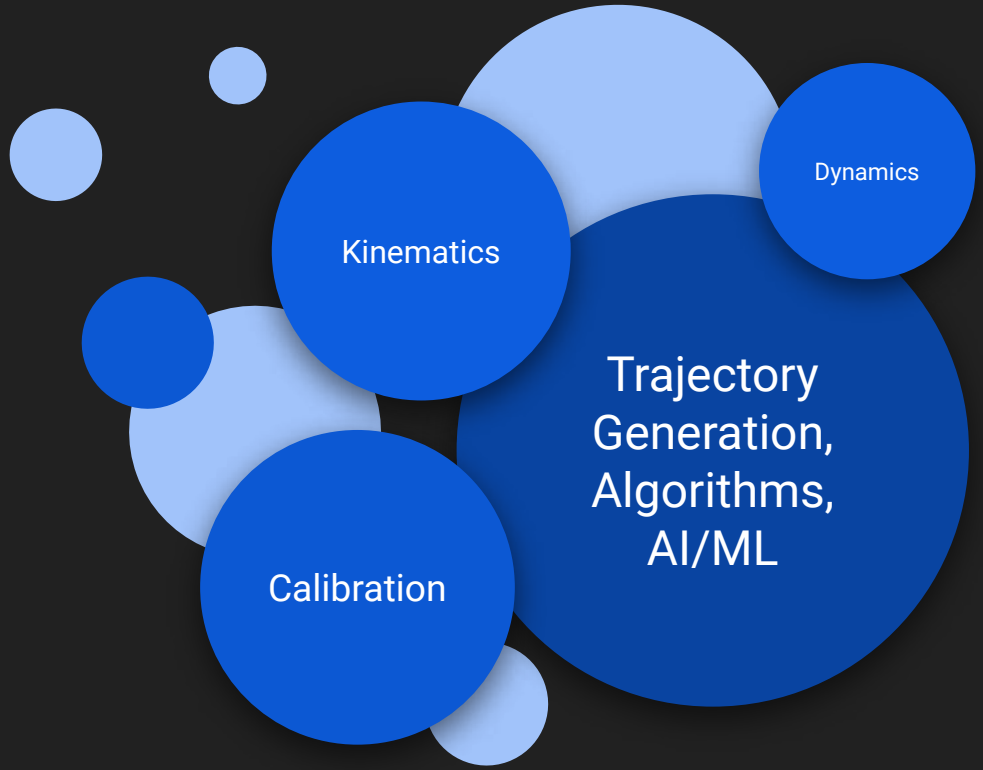


Pybotics

- Replacement for MATLAB toolboxes that were the norm in robotics calibration
- Access the scientific compute and ML packages of the Python ecosystem



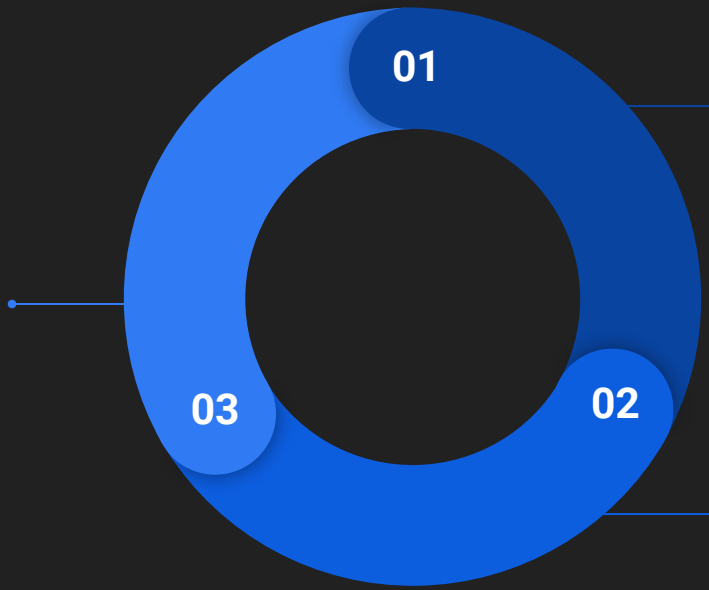
github.com/engnadeau/pybotics


Dynamics
Kinematics
Trajectory Generation, Algorithms, AI/ML
Calibration

Calibration Process

Measure Poses	Optimize Model	Validate Model
Collect a dataset of joint configurations and Cartesian poses in the workspace	Minimize the errors between the calculated FK model and the actual Cartesian poses	With a second dataset, validate the optimized model performs as expected

Optimization Cycle



Calculate FK

Compute the expected Cartesian pose of a model given a set joint angles

Calculate Errors

Difference between the actual and expected Cartesian pose

Update Model

Modify the kinematic parameters to reduce the errors

Optimization Vector

Tool Frame

1 Transform from last joint to TCP

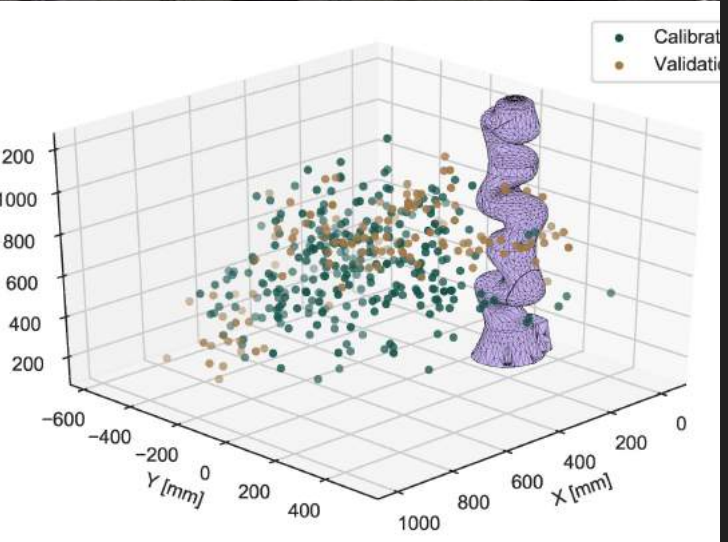
Kinematic Chain

Modified Denavit-Hartenberg parameters of the robot 2

World Frame

3 Transform from the world origin to the robot base

Measurement Tools

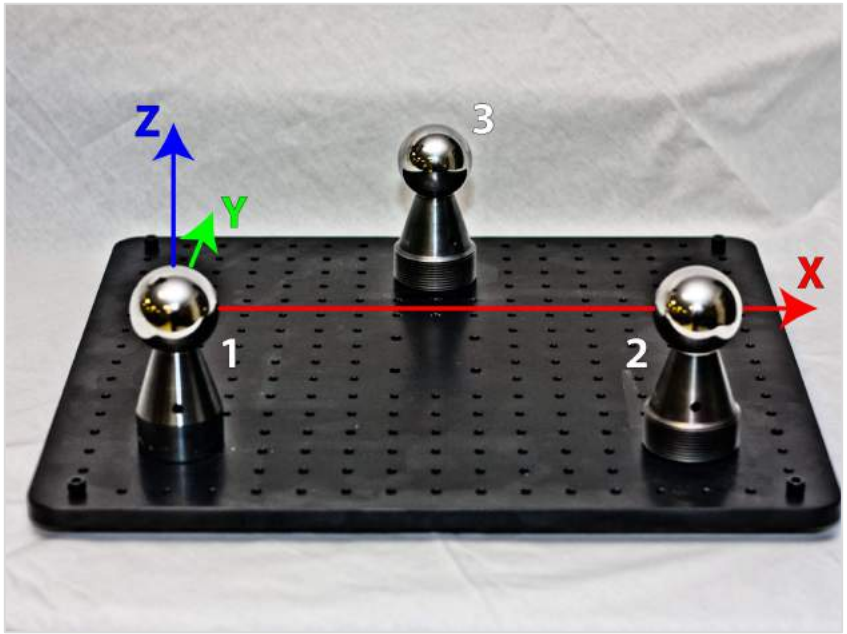


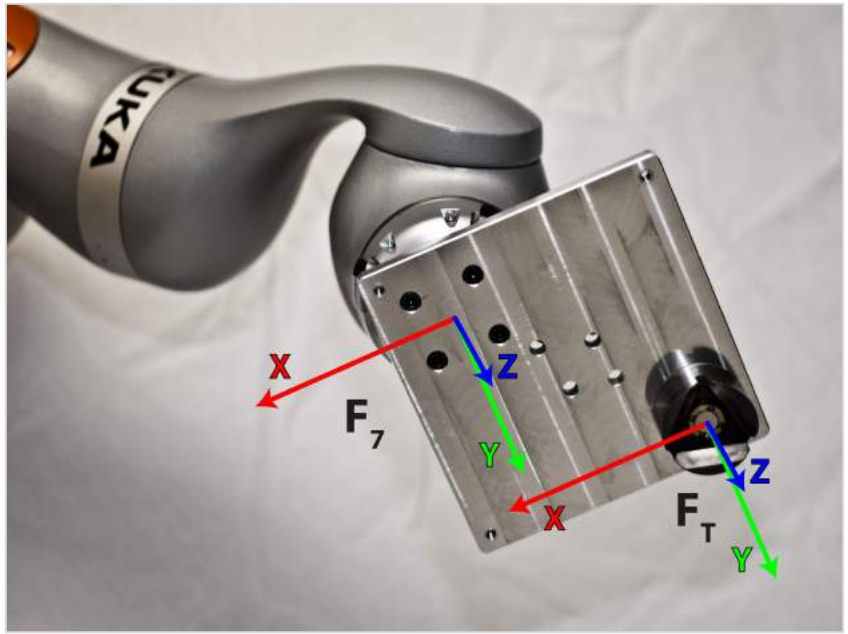
Measurement Process

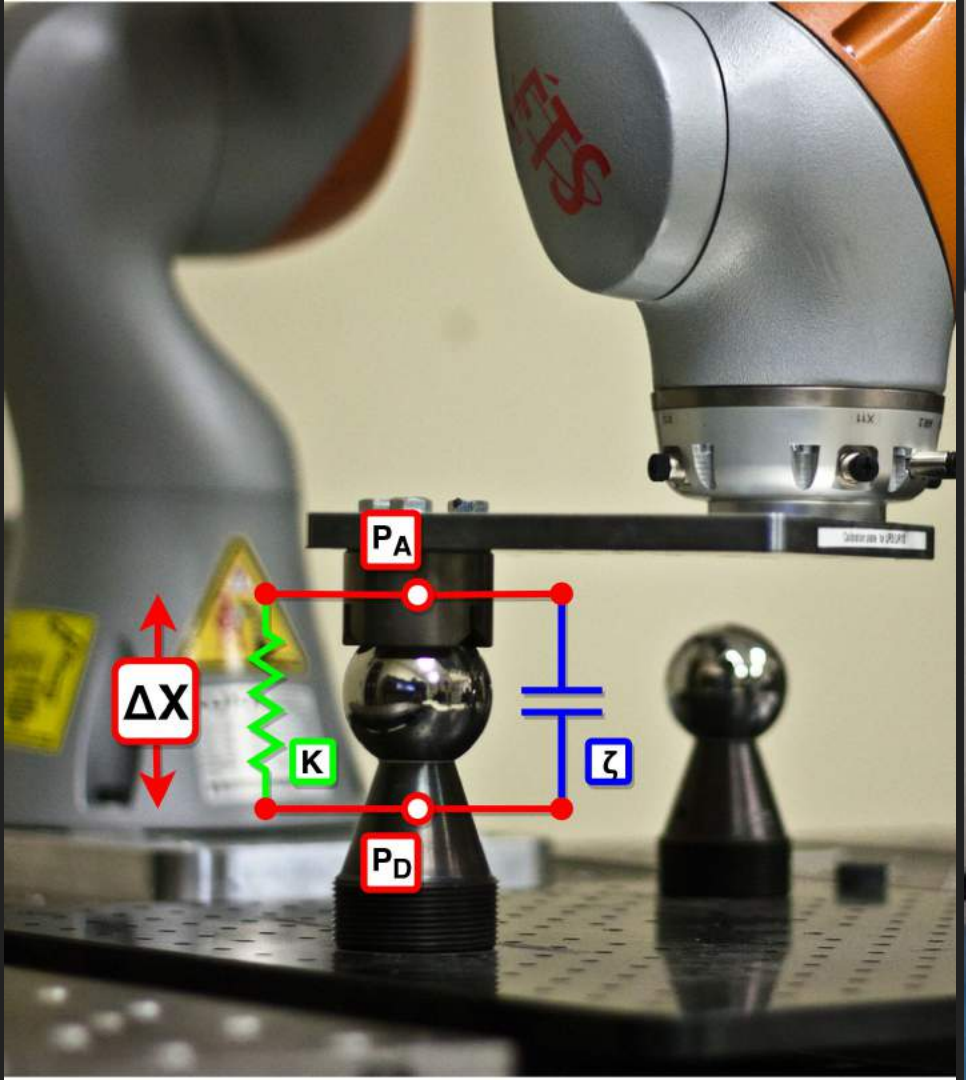
Code

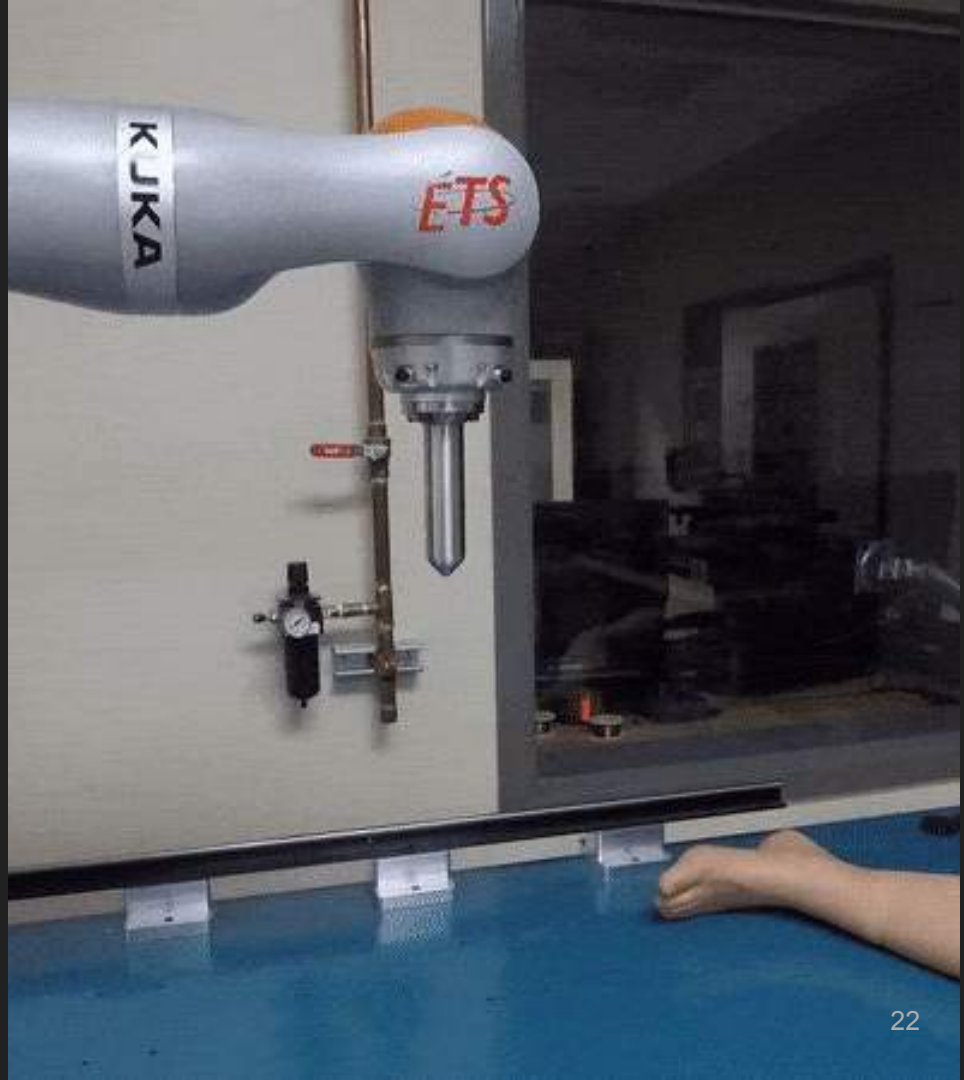
```python
import pandas as pd
from pybotics.optimization import OptimizationHandler, optimize_accuracy
from pybotics.robot import Robot
from scipy.optimize import least_squares
from sklearn.model_selection import train_test_split

# load robot model
robot = Robot()

# load joint measures
joints = pd.read_csv("joints.csv")
positions = pd.read_csv("positions.csv")

# split data into training and test sets
train_joints, test_joints = train_test_split(joints)
train_positions, test_positions = train_test_split(positions)

# initialize calibration handler
handler = OptimizationHandler(robot)

# run optimization
result = least_squares(
    fun=optimize_accuracy,
    x0=handler.generate_optimization_vector(),
    args=(handler, train_joints, train_positions),
)

# get calibrated robot
calibrated_robot = handler.robot
```

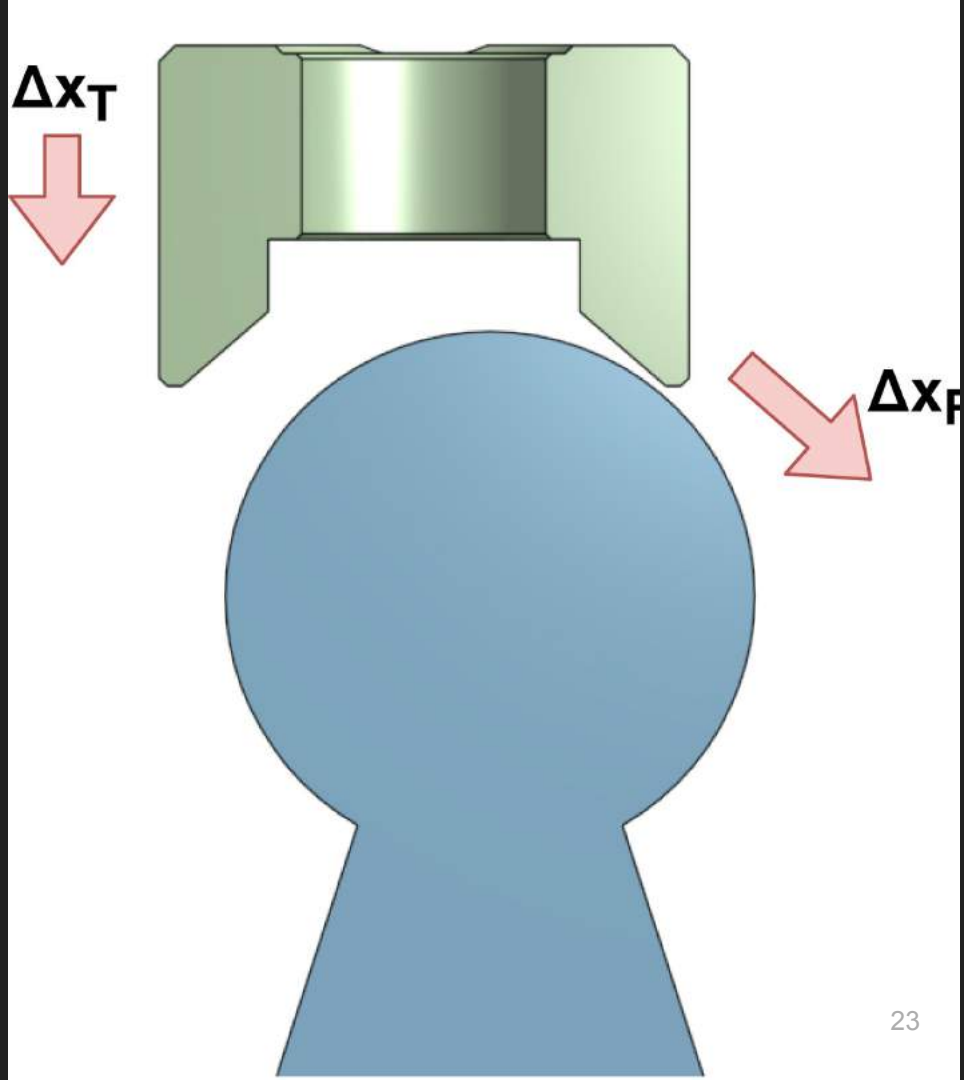
Impedance Control Self-Calibration of a Collaborative Robot Using Kinematic Coupling


Z
Y
3
X
1
2


X
Z
F 7
Y
Z
X
F T
Y

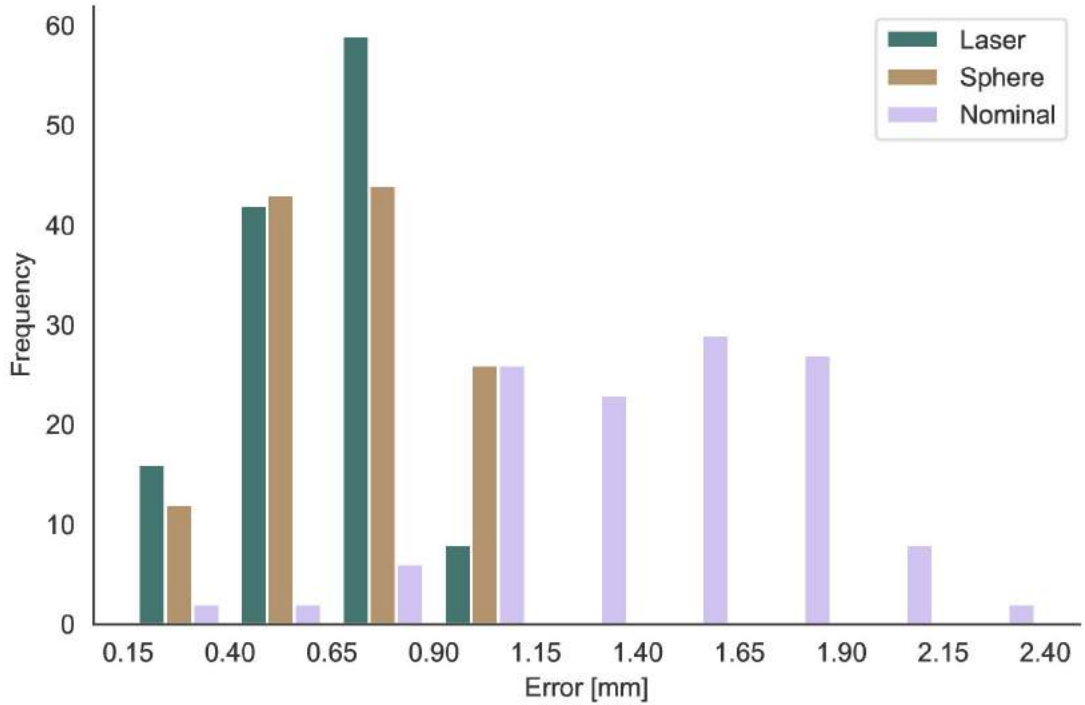

ÉTS
PA
ΔX
K
ζ
PD


KUKA
ÉTS


Δx_T
Δx_P

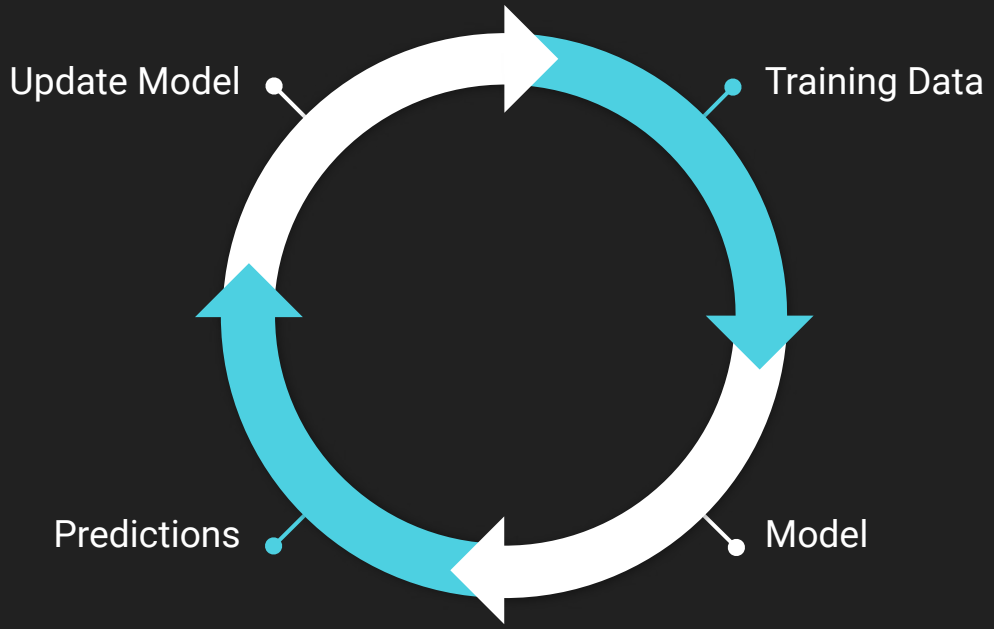

KUKA
KUKA
KUKA

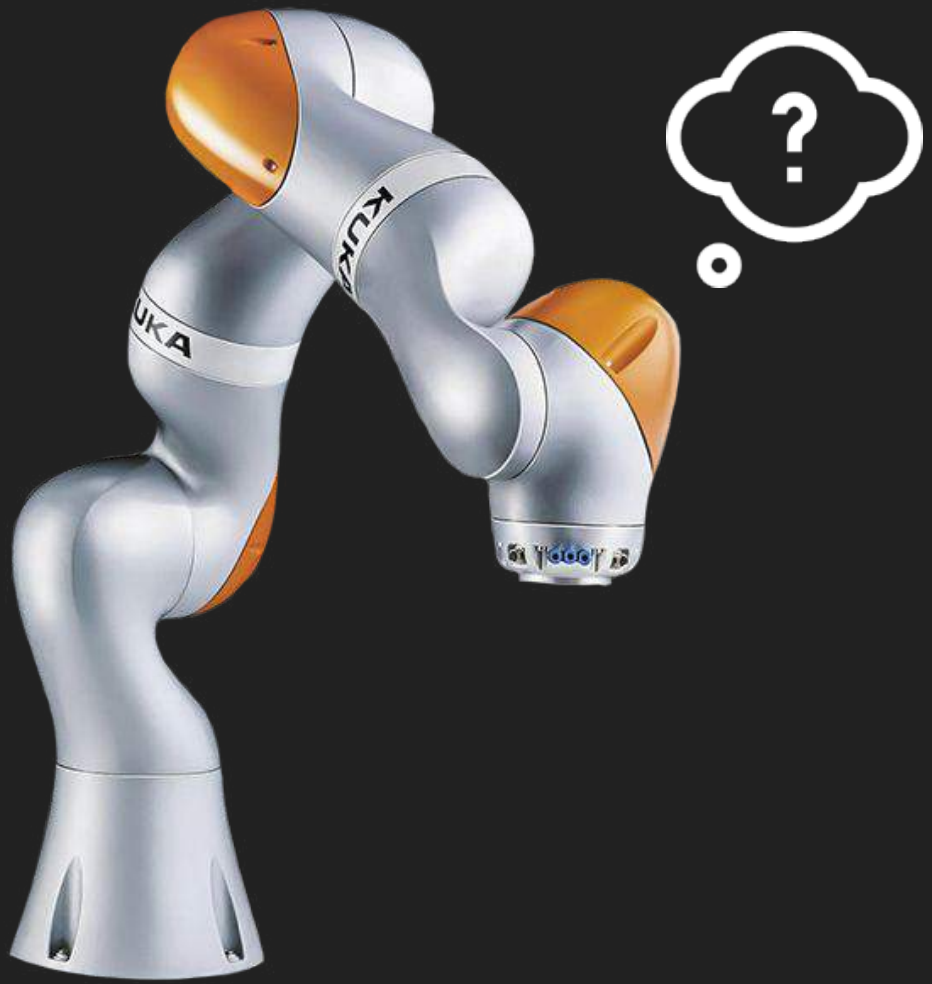
Calibration Validation



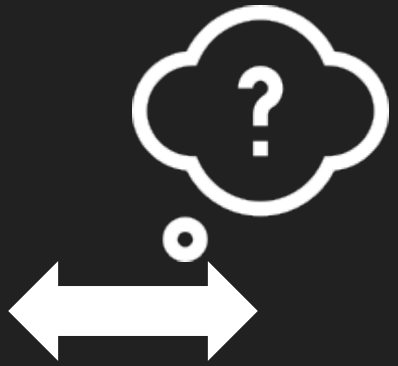
- Low-cost, robust, and simple
- Ideal for force-controlled robotic systems

Hardware in the Loop


Update Model
Training Data
Predictions
Model


?
KUKA

Data Processing
& ML



Robot Controller

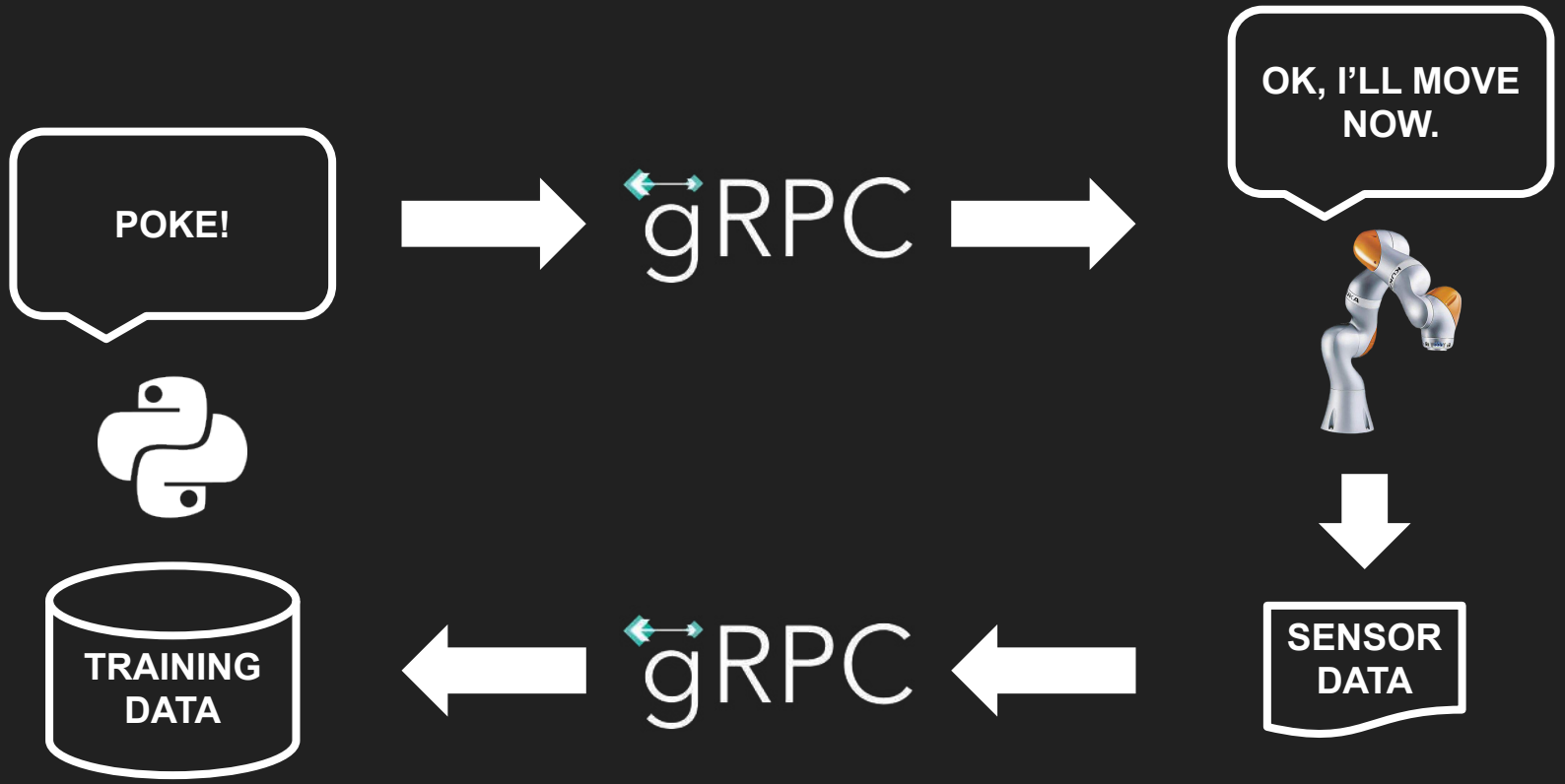

gRPC

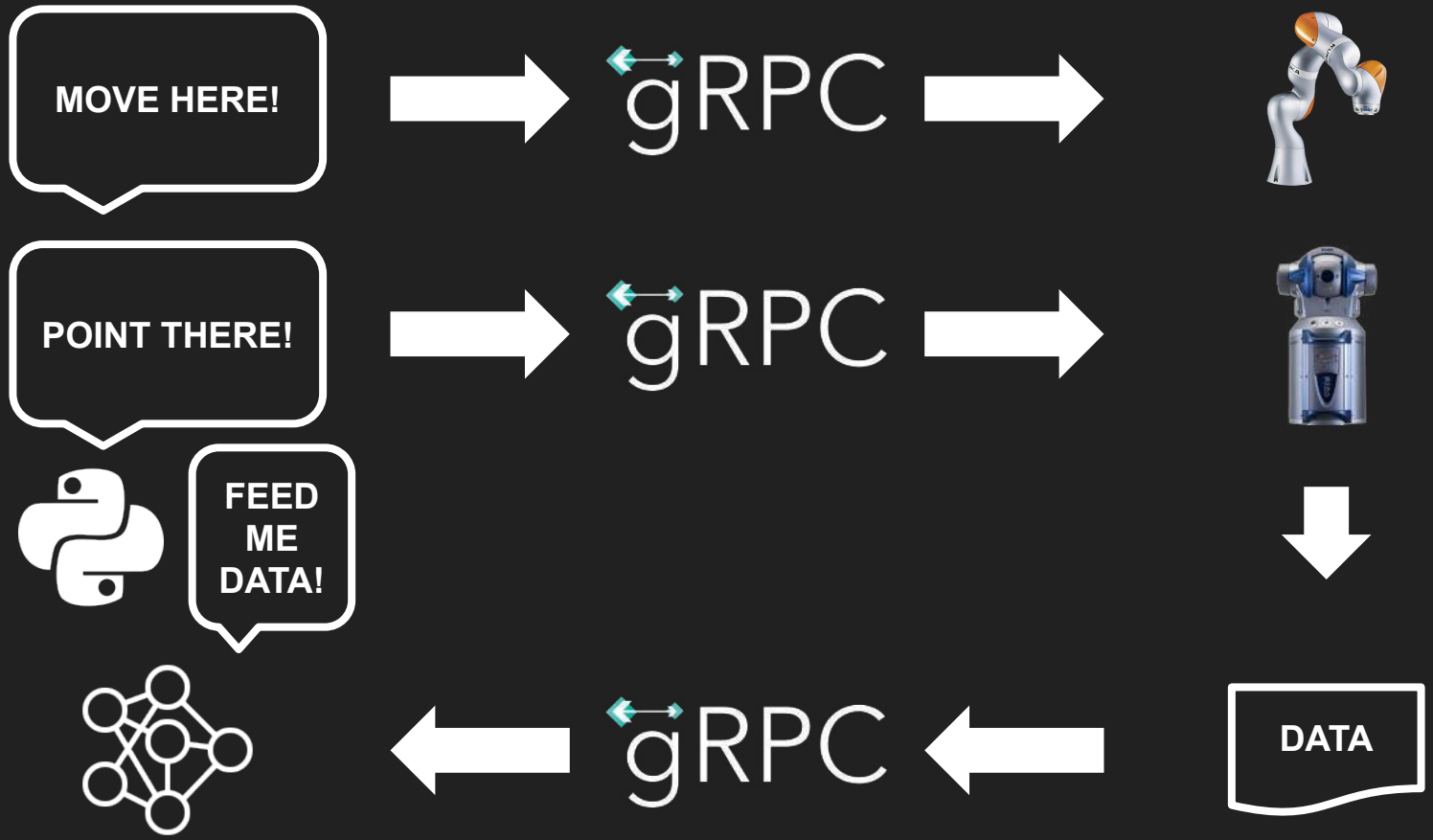
```
syntax = "proto3";

service RobotService {
    rpc Move (Joints) returns (SessionResult) {
    }
}

message SessionResult {
    int32 status = 1;
}

message Joints {
    repeated double joints = 1;
}
```

```python
import grpc
from robot_control_pb2_grpc import RobotStub


class ControllerClient:
    def __init__(self) -> None:
        self.stub = None
        self.channel = None
        self.host = "172.31.1.147"
        self.port = 30000

    def connect(self):
        self.channel = grpc.insecure_channel(f"{self.host}:{self.port}")
        self.stub = RobotStub(self.channel)
```

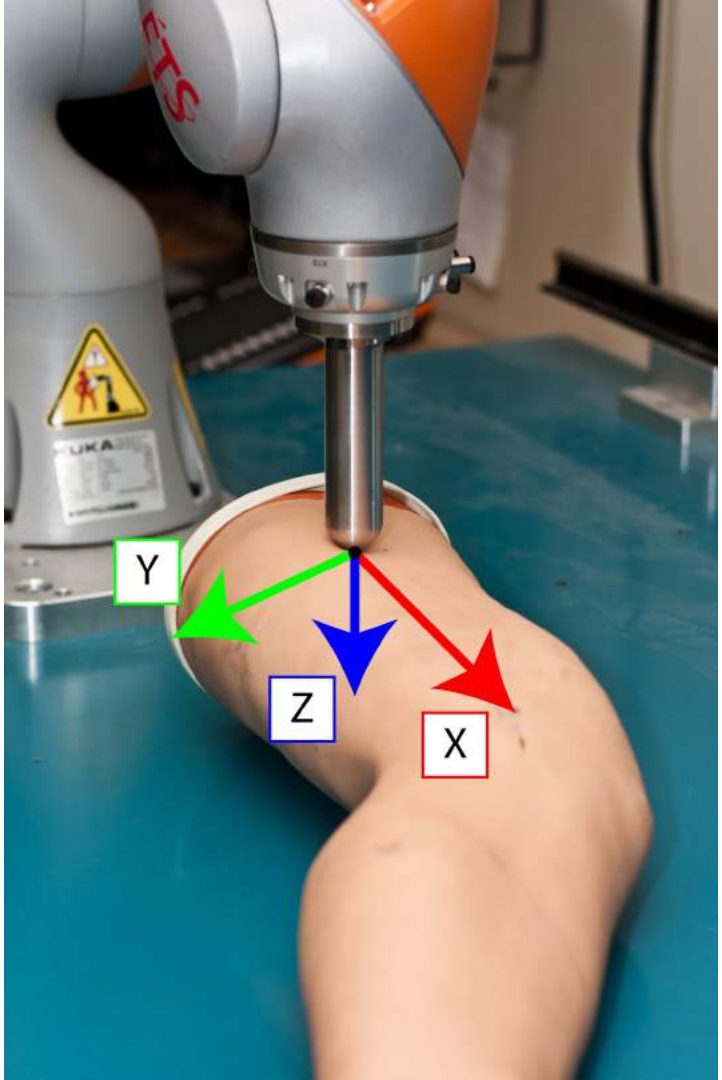
```python
from controller_client import ControllerClient
from robot_control_pb2 import SessionResult
from scipy.optimize import differential_evolution

client = ControllerClient()

def cb_objective(x):
    """Objection function callback"""
    session_result = client.stub.RunSession(x) # type: SessionResult
    session_value = evaluate_result(session_result) # type: float
    return value

# connect to robot
client.connect()

# run optimization
result = differential_evolution(cb_objective, bounds)
```

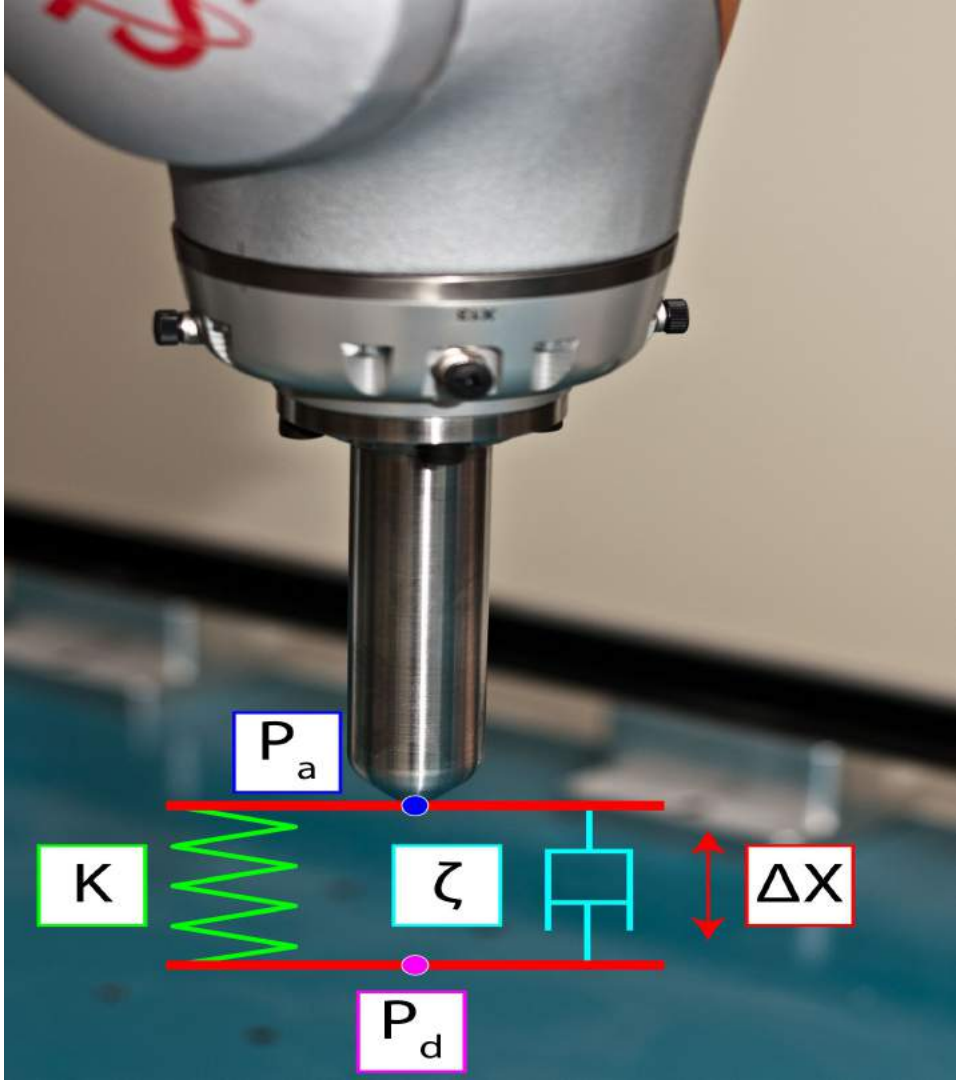
```
syntax = "proto3";

service RobotService {
    rpc Poke (SessionSettings) returns (SessionResult) {
    }
}
```

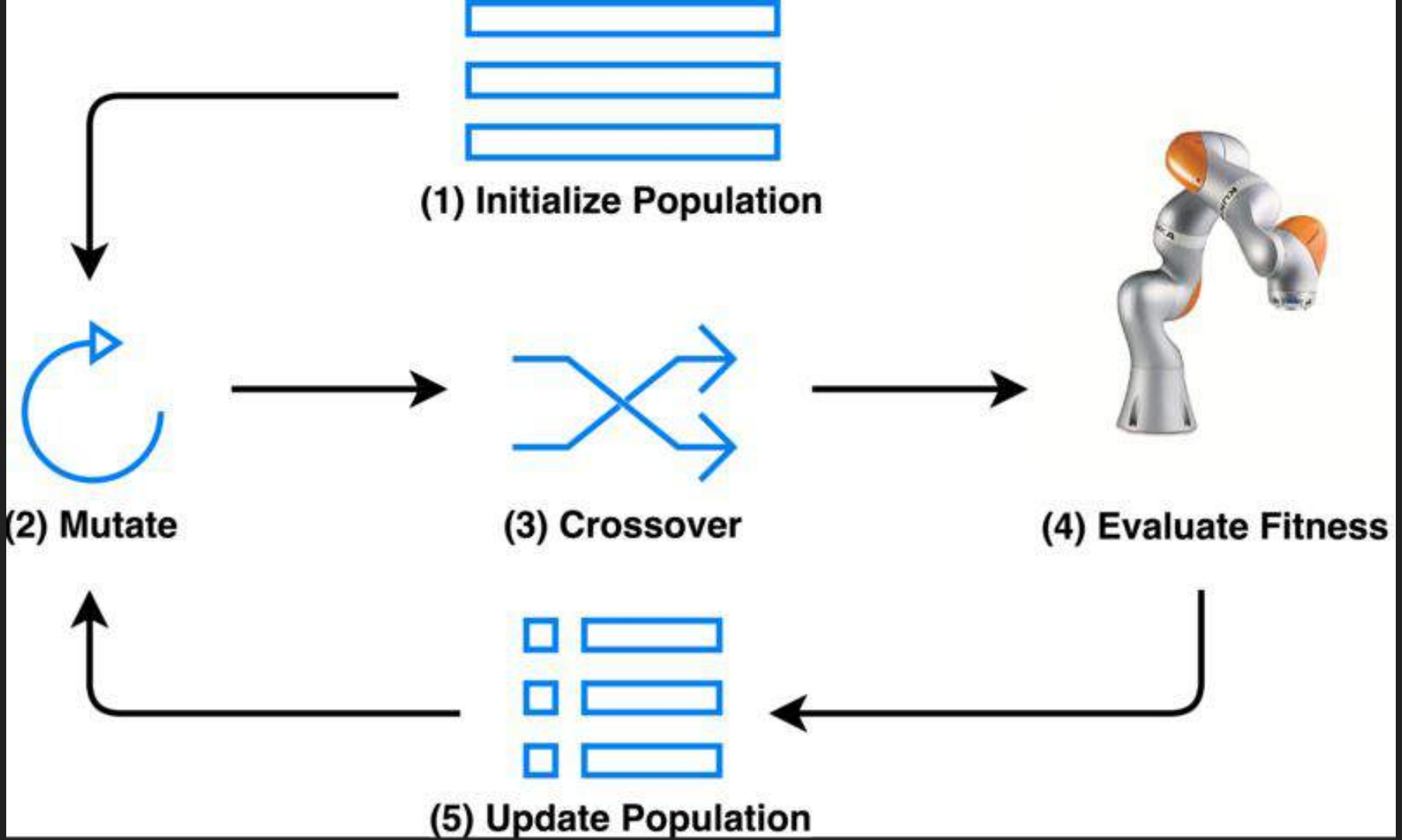

POKE!
gRPC
OK, I'LL MOVE NOW.
TRAINING DATA
gRPC
SENSOR DATA

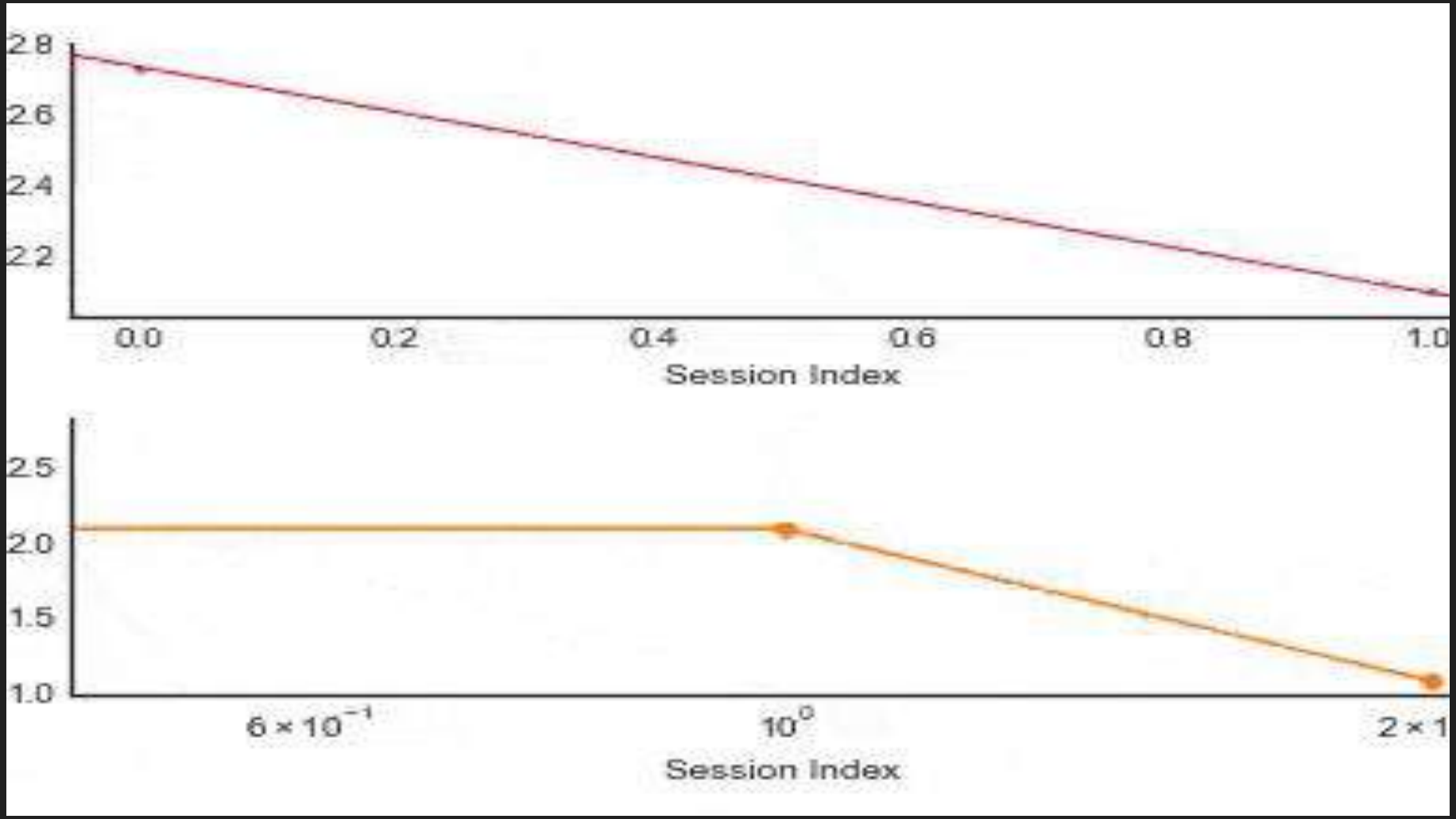

MOVE HERE!
gRPC
POINT THERE!
gRPC
FEED ME DATA!
gRPC
DATA

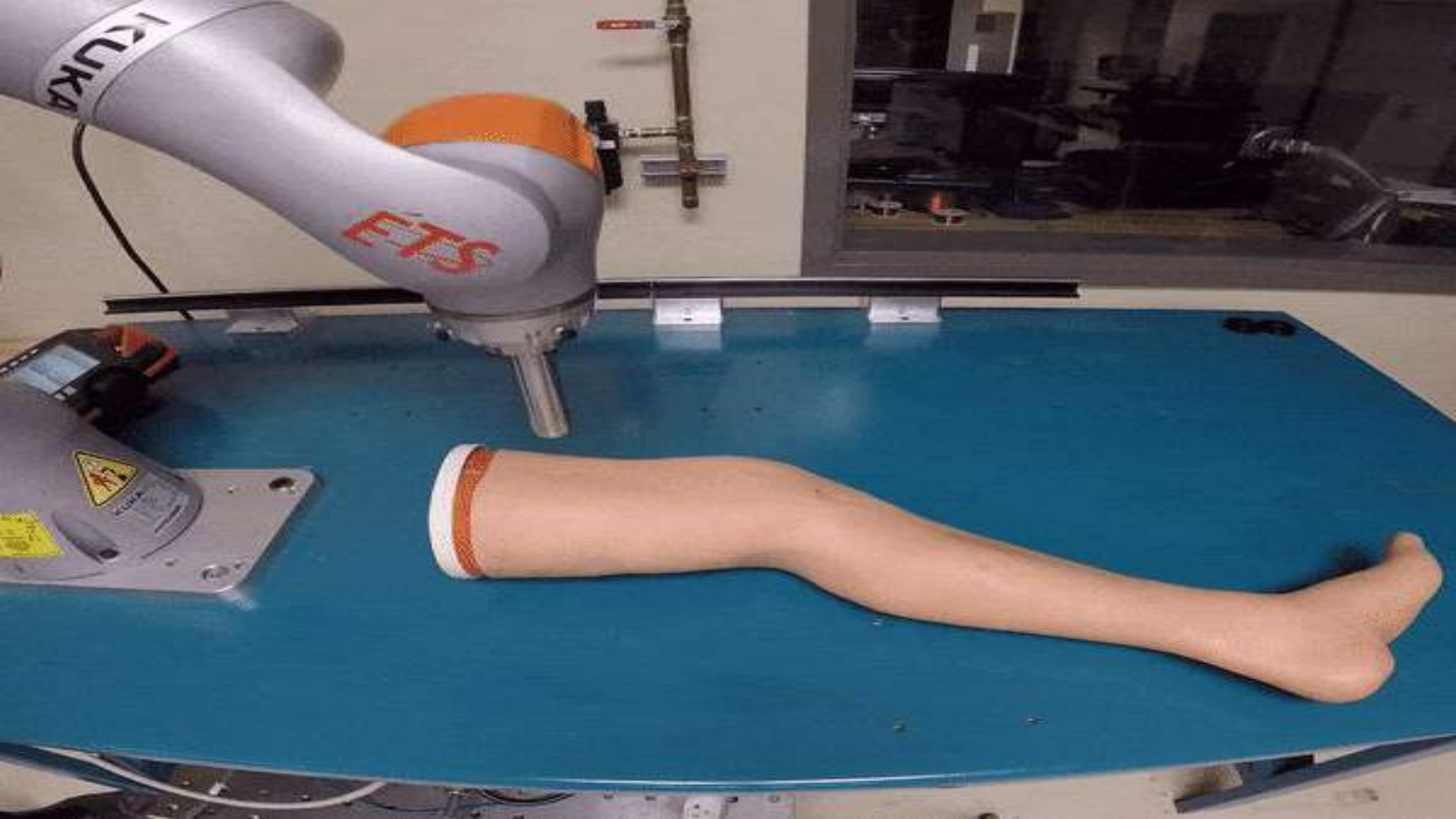
Evolutionary Motion Control Optimization in Physical Human-Robot Interaction


Y
Z
X


P a
K
ζ
ΔX
P d


(1) Initialize Population
(2) Mutate
(3) Crossover
(4) Evaluate Fitness
(5) Update Population


2.8
2.6
2.4
2.2
0.0
0.2
0.4
0.6
0.8
1.0
Session Index
2.5
2.0
1.5
1.0
6×10^{-1}
10^{0}
2×1
Session Index


KUKA
ETS


KUKA

Nicholas Nadeau, Ph.D., P.Eng.

www.nicholasnadeau.com

@EngNadeau