POLITECNICO DI TORINO

In collaboration with

École polytechnique fédérale de Lausanne (EPFL)

Master's Degree in Mechatronic Engineering

Master's Thesis

Human-Robot Collaboration Framework for Industrial Assembly Tasks

Supervisors
Prof. Marcello Chiaberge, Politecnico di Torino
Prof. Aude Billard, EPFL
Dr. Soheil Gholami, EPFL

Candidate
Carla De Vittorio

Academic Year 2024-2025

Acknowledgements

I would like to express my sincere gratitude to my supervisors at LASA, Prof. Aude Billard and Dr. Soheil Gholami, for giving me the opportunity to carry out my thesis project in their group. Their advice, availability and encouragement have been fundamental to the development of this work and to my personal and professional growth.

I would also like to warmly thank all the members of the LASA for welcoming me into their team, for the stimulating discussions and for their technical support and collaboration during my stay.

I am also grateful to Prof. Marcello Chiaberge, my supervisor at Politecnico di Torino, for his valuable guidance, support and insightful feedback throughout my studies.

Finally, I would like to express my heartfelt thanks to all the people who have supported me during this experience, both academically and personally. I am especially grateful to my family for their unconditional love, patience and encouragement, which have been a constant source of motivation throughout my studies and this project.

Abstract

In recent years, industries have been looking for ways to make production more flexible without losing efficiency. One promising approach is human-robot collaboration (HRC), where robots and people share the same workspace. Robots are well-suited for precise, repetitive actions, whereas humans are better at adapting to small variations and making quick decisions. Assembly is a natural test case: many operations still need human judgment, but robots can reduce strain by handling lifting and positioning. However, to work well together, the system must provide an easy way to assign tasks, sense the position of parts and move safely when contact occurs. This thesis project addresses human–robot collaboration for industrial assembly. The tasks involved in this HRC project include feasible grasping, accurate part tracking and compliant in-contact behavior. A practical framework is presented that links operator-driven task selection with grasp optimization, motion-capture–based tracking of both aluminium profiles, trajectory segments, respecting position/velocity/acceleration limits and a joint impedance controller. Hardware experiments on a L-shape assembly, with the robot holding a horizontal aluminium profile, show consistent behavior. The experimental results achieved contact-rich profile assembly with smooth execution, stable accuracy and consistent effort across the starting poses tested.

Contents

List of Tables

List of Figures

Acronyms

DOF Degree of Freedom.

EE End-Effector.

FCI Franka Control Interface.

FSM Finite-State Machine.

GUI Graphical User Interface.

HG Hand Guiding.

HRC Human–Robot Collaboration.

HRI Human–Robot Interaction.

MPC Model Predictive Control.

PFL Power and Force Limiting.

pHRI Physical Human–Robot Interaction.

QP Quadratic Programming.

RMSE Root Mean Square Error.

ROS Robot Operating System.

RViz ROS Visualizer.

SRMS Safety Rated Monitored Stop.

SSM Speed and Separation Monitoring.

TCP Tool Center Point.

TF Transformation Matrix.

URDF Unified Robot Description Format.

Chapter 1

Introduction

Human–robot collaboration (HRC) is an increasingly adopted practice across modern industry, where operators and robots share workspace, tasks and responsibilities. The aim is to improve safety, ergonomics, productivity and quality of production by closing the gap between fully manual manufacturing and fully automated production. In HRC, humans and robots share the workspace with complementary roles: robots handle non-ergonomic, repetitive or hazardous operations with speed, consistency and accuracy, while humans provide local perception, on-the-fly adjustment and decision-making when conditions change or are hard to model. HRC is well suited for assembly tasks. Industrial manipulators, constrained by programming and kinematics, excel at precise, repeatable motion over limited ranges, but are less effective in variable, contact-rich settings. Human operators complement these limits with higher dexterity and the ability to adapt to changing geometry. In practice, the robot provides repeatable motion and load support, while the operator handles contact level adjustments and context specific decisions. This division of roles keeps the system both efficient and adaptable.

Despite progress, many studies still treat interaction, perception, planning and control as separate problems, making it difficult to build frameworks that remain safe and responsive to the operator. This project has targeted the main integration challenge to connect the different problems: grasp and part positioning planning should be derived from the GUI (graphic user interface) configuration and from the tracked poses of the profiles and the robot, with continuous trajectory updates remaining within bounds and coordinated with compliant control during execution.

This thesis addresses a practical instance of human–robot collaboration: an operator and a 7-DoF (degree of freedom) robot assemble aluminium profiles into “L” and “T” geometries. Holding and positioning parts is repetitive, accuracy-sensitive and tiring, whereas humans are effective at on-the-fly adjustment. A collaborative setup splits the work: the robot provides repeatable motion and steady effort, while the operator governs fine positioning at contact. The objective is to develop a framework that completes these assemblies reliably and preserves predictable behaviour at contact. The approach is to keep sensing and decision-making aligned with what the operator can observe and influence, and to couple specifications, perception and control so that responses remain smooth and predictable.

To achieve this, the proposed framework connects five elements. The first element of the framework is an operator-driven GUI, which defines the task and breaks it down into executable

sub-tasks with explicit state transitions. Next, grasp optimization places the gripper along the robot-held profile within safe margins and passes that choice to motion planning, so that the chosen grasp is both reachable and useful for later phases. Human motion is then inferred from the pose of the profile grasped by the operator, providing the information needed for alignment and coordinated execution. Time-parameterized trajectories are then generated with explicit feasibility checks on reachability, collisions and joint limits, ensuring that each segment can be executed safely from the current state. Execution is handled by joint impedance control with torque rate limits, so contact is taken up smoothly and actuation remains within bounds. Poses for both profiles are expressed in the robot base frame, which keeps sensing, planning and control consistent.

The operator specifies the target geometry (L or T), assigns the robot's role (horizontal or vertical profile), and may set options such as hand dominance and table side. Given these inputs, the framework produces the full robot sequence: it computes a suitable grasp, plans feasible motions, tracks the human-held profile[1] and aligns the robot-held profile[2] to it for assembly.

The evaluation focuses on hardware performance and on how robot posture[3] and initial profile poses affect tracking and effort. Seven experiments execute L-shape assemblies with the robot assigned to the horizontal profile. The experiments span different initial profile poses and starting arm postures, allowing both profiles orientation and postural effects to be assessed. Results indicate that the system completes all assemblies reliably and within safety margins. Motions follow the intended paths closely before contact. During lifting and alignment, small but consistent deviations appear, influenced by the orientation of the operator's profile and the robot's starting posture. Once contact is reached, interaction forces stabilize and overall effort remains moderate. Complementary simulations confirm the same behavior and show that the main limitations observed on hardware are attributable to posture rather than to the framework itself.

All activities (design, implementation and experiments) were carried out at LASA, EPFL.

Chapters overview

The chapters are organized as follows:

- **Chapter 1** introduces the thesis motivation and objective and outlines the contributions.
- **Chapter 2** surveys related work on human–robot collaboration for assembly, with emphasis on task allocation, perception–planning–control integration and existing software frameworks.
- **Chapter 3** presents the framework: GUI-driven task setup, grasp optimization, profile tracking, motion planning with feasibility checks and impedance control.

[1] "Human-held profile": the aluminium profile grasped by the human operator.

[2] "Robot-held profile": the aluminium profile grasped by the robot.

[3] "Robot posture": is the specific configuration of its joints, meaning the particular set of joint angles or displacements that determines its shape and position in space. It differs from the pose, which describes the position and orientation of the end-effector in space.

- **Chapter 4** summarizes the hardware and software used in this work, including the robotic platform, motion capture and supporting libraries.
- **Chapter 5** details the experimental setup, state-machine breakdown and the conditions for the executed trials.
- **Chapter 6** reports the hardware results per experiment and closes with an overall analysis across experiments.
- **Chapter 7** concludes the thesis, discussing limitations and directions for future work.
- **Appendix A** lists the custom ROS package structure and selected implementation details.
- **Appendix B**: presents brief robotics notions.

Chapter 2

Human-Robot Collaboration

2.1 HRC and cobots

Modern manufacturing increasingly relies on human–robot collaboration, where human operators and robots collaborate in the same workspace and perform tasks together. Unlike conventional industrial robotics, where machines are caged and optimized for maximum speed and throughput, HRC embeds the operator within the collaborative work process and uses the robot for precise, repeatable actions when neither fully manual nor fully automated execution works well [1] (Figure 2.1). Current implementations range from assembly assistance in manufacturing to medical support and rehabilitation, and converge on a common design challenge: maintaining productivity while meeting strict safety requirements and keeping interaction operator friendly in a shared workspace.

Collaboration is commonly implemented in two ways. In *physical collaboration*, intentional contact and force exchange are allowed and regulated via compliant actuation, torque sensing and motion policies that react to human intent [3]. In *contactless collaboration*, coordination is achieved without touch using environment perception and attention cues (vision based human tracking, gesture recognition or speech interfaces) to coordinate task sequencing [3, 4].

HRC systems are often distinguished into *workspace sharing*, where human and robot carry out different handling or assembly tasks within the same area and coordination is limited to collision avoidance with predictable trajectories, and *workplace-and-time sharing*, where human and robot may execute handling or assembly together at the same time, demanding tighter coordination and rapid replanning to accommodate human motion [5].

The research literature shows that this collaborative operating mode places new demands on sensing, planning, control and safety and has led to collaborative robots (“cobots”) designed for close human–robot interaction [6]. Traditional industrial manipulators are optimized for high payload, long reach and fast, repeatable motion in fenced cells, while cobots are lighter, easier to redeploy and prioritize safety functions to collaborate with humans and intuitive programming so cells can be reconfigured quickly (Figure 2.2). This distinction impacts robot speed/force limits, sensor selection and integration, validation and certification workflow. In practice, cobots prioritize human presence and safety-rated interaction, provide intuitive programming for frequent task changes and trade top speed for accuracy and compliance when sharing space with

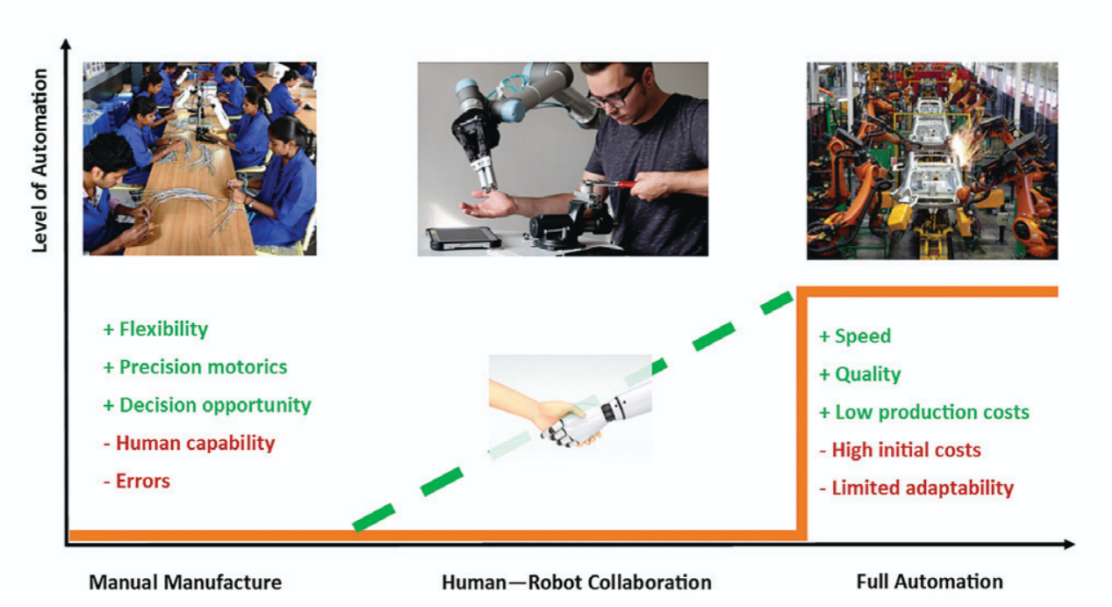


Figure 2.1: Standing of Human–Robot Collaboration [2].

operators. Modern cobots incorporate perception (vision, force/torque sensing, proximity and presence detection) to interpret task context and adapt motion for smoother, safer interaction. The same sensors also provide the safety inputs used to monitor the workspace and to operate the collaborative modes defined in the standards. The differences between cobot and robot are summarized in Figure 2.3.

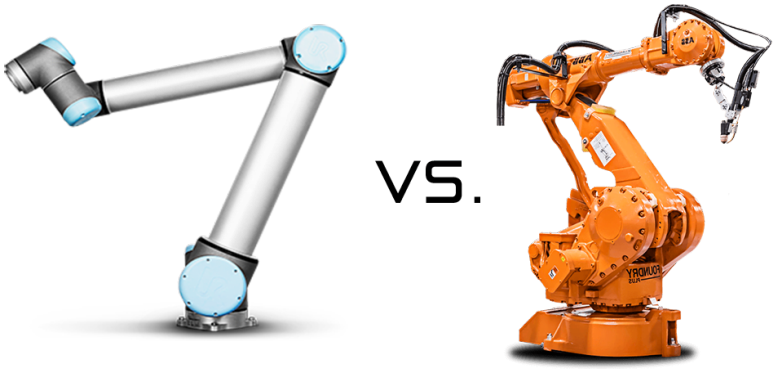


Figure 2.2: Cobot (left) vs. Industrial robot (right).

	Conventional robot	Cobot
Purpose	Perform tasks automatically	Collaborate with humans
Programming	Typically pre-programmed or scripted	Easily programmed by humans
Safety	Require safety measures and barriers to protect humans	Designed to work alongside humans without endangering them
Flexibility	Designed to perform a specific group of tasks	Can perform a wide range of tasks
Cost	Typically expensive	Less expensive than conventional robots
Complexity	More complex to program and operate	Simple to program and operate
Payload	High payload capacity	Lower payload capacity
Accuracy	Lower precision	High precision
Size	Bulky and Space consuming	Smaller and dense
Application	Manufacturing and assembly lines	Small-scale manufacturing, research and development

Figure 2.3: Differences between robot and cobot [7].

HRC is well suited when tasks combine repetitive segments with local decisions and adjustments that are hard to predefine. In manufacturing, robots deliver endurance and precision for holding, positioning and moving parts along controlled trajectories, while the operator manages variable geometry, tolerances or on-the-fly corrections at contact. Because people and robots share the same space, safety is a key factor to address. The basic safety requirements for the robot and for the integrated robot system are set out in ISO 10218-1 [8] and ISO 10218-2 [9]. These standards also define the four protective principles that enable collaborative operation: safety-rated monitored stop (SRMS), hand guiding (HG), speed and separation monitoring (SSM) and power and force limiting (PFL). ISO/TS 15066 [10] provides additional guidance for each of the four collaborative modes. The Technical Specification sets out requirements, limits and examples for collaborative robots and applications and clarifies the roles of robot manufacturers and system integrators.

To achieve safe, predictable and productive collaboration, HRC systems rely on: (i) perception and human intention estimation for situational awareness and role allocation, (ii) interaction and communication mechanisms that keep actions legible to the operator and (iii) task allocation and workflow design that split responsibilities transparently under practical constraints.

2.1.1 Perception and Human Intention

Perception supplies the safety and state of the workspace needed for shared workspaces. For safety, it delivers the inputs for speed and proximity monitoring and interaction state (who is in the workspace, where they are and how fast they are moving). For environment state: it localizes objects, estimates poses, detects hands during handover and regulates contact forces so motion remains predictable to the operator. At the workstation level, perception typically blends vision (cameras or scanners) with safety certified sensing and joint torque feedback embedded in compliant control. Together, these inputs determine when the robot may switch operating mode and keep motion within safe limits. In settings such as welding or sealing, safety devices supervise the robot's trajectory, while torque sensing cobots support hand guided teaching and automatically slow down in shared zones. In practice, this combination improves ergonomics and maintains throughput without requiring physical barriers.

Intention complements perception by revealing the human operator's goal or next move and enabling the robot to choose suitable roles and assistance levels. Intent estimates, obtained by inference or explicit cues, feed situation awareness to anticipate upcoming states and make timely decisions [11, 12]. Human intent detection, together with workspace perception and feedback, lets the system allocate tasks between the operator and the robot [13]. Role allocation during physical collaboration is adaptive: intent cues indicate when the robot should defer, support or take the lead [14]. This role adaptation links intention estimation to real time allocation, motivating the task allocation methods discussed next.

2.1.2 Interaction and Communication

Physical Human–Robot Interaction (pHRI) concerns contact, force exchange and compliant control during shared manipulation. Human robot collaboration also depends on cognitive interaction: cues that let the human and the robot anticipate each other's intentions and coordinate smoothly without continuous verbal guidance. Nonverbal signals such as gaze, body orientation and gesture coordinate turn taking and readiness in handovers and collaborative assembly. Embedding these cues in robot behaviour improves predictability and user comfort and reduces coordination effort.

2.1.3 Task Allocation and Workflow Design

Task allocation defines who does what in a collaborative cell: which actions are assigned to the robot and which remain with the operator. The objective is a division that is safe, traceable and efficient for the given context and constraints. A practical workflow starts by decomposing the job into concrete subtasks (pick, align, hold, fasten, inspect) and by selecting criteria, that can be checked on the workstation, such as required accuracy and force, dexterity and grasp complexity, visibility or occlusion, reach and tooling and the speeds permitted under the chosen collaborative mode [4, 15].

Each subtask is then matched to the agent that can satisfy those needs with predictable behavior: robots take repeatable, posture intensive or force stable work, while the operator performs tasks that depend on judgment, variable geometry or quick on-the-spot correction [1, 4]. The tasks divisions are verified against task constraints, collision and separation limits, ergonomic load and cycle time. If these checks fail, the mapping is adjusted or recomputed. A possible solution is lightweight multi-criteria scoring, sometimes paired with (re)scheduling, that provides clear decision criteria, reducing unplanned changes and preserving workflow continuity [16, 17].

Where the task set is larger or there is need of parallel task execution by human and robot, optimization-based assignment can be used. The inputs remain the same, while the optimization targets time, cost or a multi-criteria utility function [18–20]. Estimating the *automation potential* of tasks can guide pre-screening for assignment [21]. Workplace design and preliminary analysis (layout, robot placement, collision checks, achievable times under the chosen safety mode) should be integrated into the planning loop too [15, 22]. Crucially, allocations are revisited when the product mix or tooling changes so that the reasoning behind each assignment remains valid and transparent.[4].

2.1.4 Current Limitations and Open Challenges

Recent researches highlight recurring challenges spanning safety assurance, perception robustness, human factors, task allocation at scale and workforce skills [23].

Safety assurance and runtime supervision

Collaborative cells must ensure protection without increasing task time: distance/speed supervision and stopping behaviour, together with coverage, occlusions and sensor uncertainty, drive the safety–productivity trade-off [23, 24].

Perception limits in real environments

Accurate perception under occlusions and lighting changes, real-time operation and integration with planning and control remain key bottlenecks in collaborative factories [23, 25].

Psychological safety and trust

Beyond physical protection, operators need to perceive safety, calibrate trust appropriately and avoid excessive workload. Clear robot behaviour leads better coordination [1, 23].

Task allocation and reconfiguration

Allocation still struggles with parallelism, human work patterns and changing conditions. Design/validation of the workplace (layout, sensing, feasibility of motion) must stay in the loop during redeployments [22, 23].

2.2 Frameworks for HRC in assembly

Modern collaborative assembly frameworks tie together perception and intent, ergonomic assessment with adaptation and decision/execution, while explicitly embedding operator knowledge, preferences and comfort in the loop.

Below are practical frameworks used in collaborative assembly, grouped by what they focus on: people, ergonomics, perception, safety and planning.

Human-centric learning and optimization

Roveda et al. [26] propose a human-centric framework that captures operator knowledge via *preference-based optimization*: the operator gives simple, pairwise judgements ("this run is better than that one") and marks outcomes as acceptable or not. The system uses these judgements to learn what "good" looks like and to propose the next settings to try. Once this preference and acceptability model is learned, it computes a single, consistent reference for execution (for example, an end-effector speed profile) that also respects process and hardware limits.

Preference learning with ergonomics

Falerni et al. [27] combine preference learning with a standard ergonomic index (RULA) to choose the robot end-effector pose in a handover task. The operator compares candidate poses and the method searches for a pose that fits those preferences while keeping the ergonomic score low. In tests, it converges to poses that reduce musculoskeletal load without ignoring what the operator finds comfortable. The authors also note that comfort and ergonomics do not always align. They add a penalty so that the search avoids risky postures but still leaves room for personal adjustment.

Perception and decision integration

Iodice et al. [28] present a framework that ties together vision (object detection and tracking), 3D human pose and action recognition and continuous ergonomic scoring, then use a Behaviour Tree to decide when the robot should step in or stand back. The idea is straightforward: watch the scene with non-invasive cameras, understand what the operator is doing and how demanding the posture is and trigger the right intervention at the right time. In their tests, the system recognized operator intent reliably, flagged ergonomic risk quickly and issued decisions with very low delay, showing that it can run in real time on a collaborative cell.

Natural language and multi-modal communication

A practical multi-modal pipeline combines object recognition, speech understanding, user-of-interest detection, gesture and gaze so the robot can infer what is being asked, who is addressing it and which object is intended. Paul et al. [29] show that speech can supply the task while gaze/gesture and user-of-interest resolve references and addressability, reducing ambiguity and making commands more reliable. Lim et al. [30] use a large language model as a voice interface for a collaborative assembly cell. Spoken instructions are turned into tasks and subtasks the robot can execute. When something goes wrong, the system explains the issue, asks the operator for help and resumes from the right step once corrected. In their case study, the setup handled varied phrasings and supported quick error recovery with modest integration effort.

Safety and scene supervision

Assembly frameworks rely on safety building blocks that have matured across the HRC literature: depth-space distance monitoring for separation [31], adaptive damping compliant with collaborative safety guidance [32], multi-sensor person tracking for dynamic zones [33] and distributed proximity sensors for local reaction ([33] In practice, these modules provide safety envelopes and manage transitions between autonomous and cooperative modes without excessively degrading throughput, especially in workplace-and-time sharing [5, 7].

Chapter 3

Proposed Framework

The proposed framework is designed to achieve safe and effective collaboration between robot and human operator through a GUI-based task allocation, grasp optimization, robot state and profiles tracking, motion planning and impedance control. The human operator selects the task and role assignment via GUI. This information is then fed to the robot to optimize the grasp and to study how to approach the assembly. The system then tracks the human-held profile, planning feasible trajectories and aligns the robot-held profile to the human-held one. The human operator then assembles the two of them together.

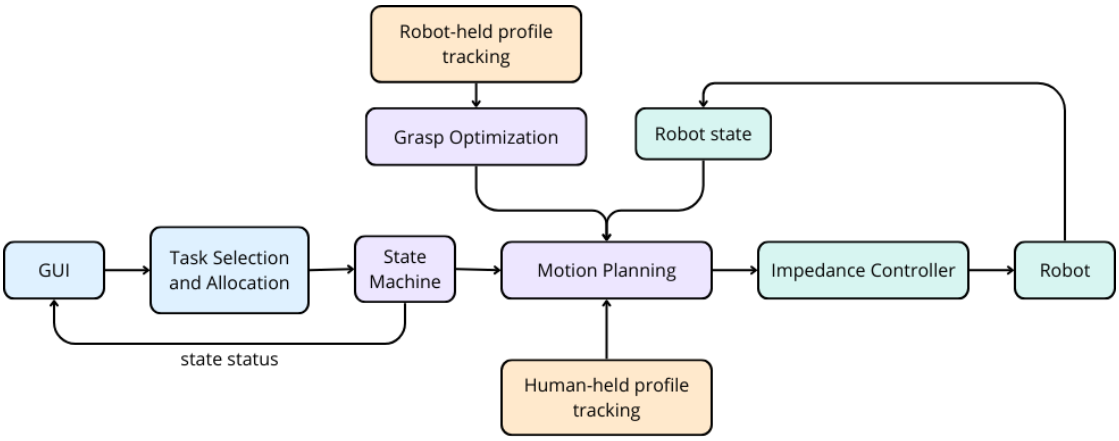


Figure 3.1: Framework for human–robot collaborative assembly of aluminium profiles.

The framework (Figure 3.1) operates as a pipeline that keeps the operator in the loop. The GUI captures task intent and role assignment, and task selection and allocation turn these choices into phase objectives while driving a finite state machine. A grasp optimizer then selects a feasible contact point on the robot-held profile. Perception streams (human held and robot held profile tracking and the current robot state) continuously supply the short step motion planner, which produces feasible trajectory segments. An impedance controller executes these segments with compliant behavior and execution status is returned to the GUI so the operator can follow progress and intervene when needed.

The following sections describe each element of the framework.

3.1 Task selection and allocation (GUI)

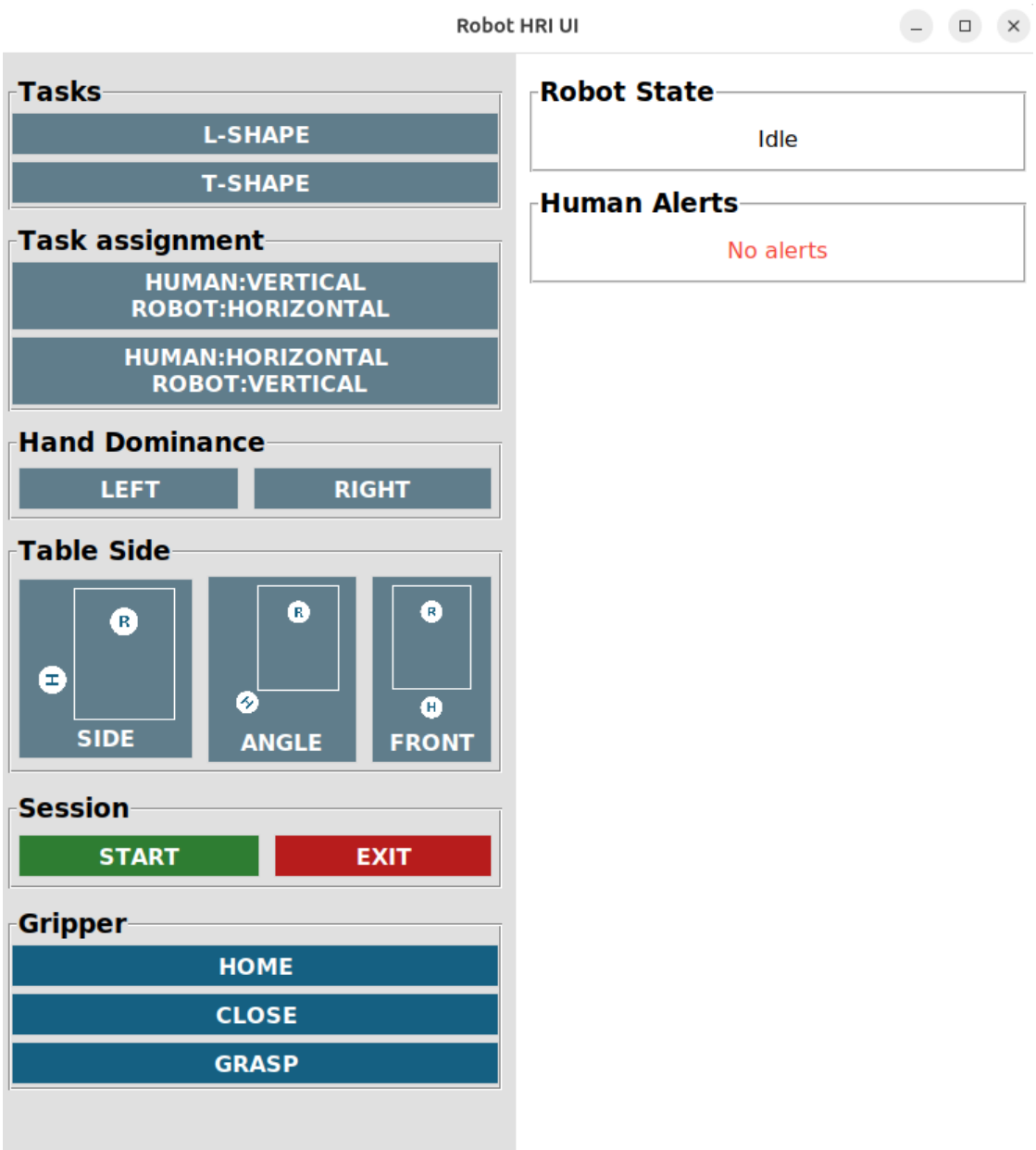


Figure 3.2: Graphical user interface for HRC, implemented in Python using `tkinter` [34].

Through the GUI (Figure 3.2), the human operator selects the task type in "Tasks" (L or T-shape) (Figure 3.3). If T-shape is selected, the operator can also choose in which point to assemble the robot-held profile, from the middle to the tip of the human-held profile. The human then assigns the robot either to the vertical or horizontal profile (from "Task Assignment"). If the robot is assigned to the vertical profile, the operator also specifies hand dominance and the table side. After these selections, the task can be started. A gripper control panel is available in the lower-left area for use during execution or when the robot is idle and the grasp width is set automatically from the profile width.

Information for the operator is shown on the right side. "Robot state" panel shows the current state of the robot, while "Human Alerts" gives important notices for safe execution and reports any errors as they occur. These inputs configure the robot's approach direction, grasp position along the robot-held profile and motion and alignment objectives.

3.2 Grasp Optimization

The grasp coordinate x is defined along the profile's longitudinal axis with the object frame at the center. End margins are introduced to account for sensor placement (c_s) at the negative end and assembly space (c_a) at the positive end, thus reducing the graspable region. Let $x \in [x_{\min}, x_{\max}]$ denote the grasp coordinate, where

$$x_{\min} = -\frac{L}{2} + c_s, \tag{3.1a}$$

$$x_{\max} = +\frac{L}{2} - c_a. \tag{3.1b}$$

where L is the profile length.

The preferred grasp x_d depends on the GUI:

$$x_d = \begin{cases} x_{\min} + \delta, & \text{horizontal robot-held profile,} \\ x_{\max} - \delta, & \text{vertical robot-held profile.} \end{cases} \tag{3.2}$$

with $\delta = 0.02\,\text{m}$.

To optimize the grasp position, the quadratic deviation subject to the above constraints is minimized:

$$\begin{aligned} \min_{x} \quad & (x - x_d)^2 \\ \text{s.t.} \quad & x_{\min} \le x \le x_{\max}. \end{aligned} \tag{3.3}$$

3.3 Profile Tracking

The motion-capture system, expressed in the robot base frame, provides for each profile $i \in \{H, R\}$ (human-held, robot-held) a pose:

$$\mathbf{T}_i = \left(\mathbf{p}_i,\ \mathbf{q}_i\right),$$

where:

- **Position:** $\mathbf{p}_i = [x_i,\ y_i,\ z_i]^\top \in \mathbb{R}^3$ (meters, expressed in the chosen reference frame),
- **Orientation:** $\mathbf{q}_i = [w_i,\ x_i,\ y_i,\ z_i] \in \mathbb{S}^3$ is a unit quaternion (scalar-first) (B).

A common frame definition for the profiles (Figure 3.3), plus the robot's end-effector pose, gives the geometry needed for grasp, approach and alignment.

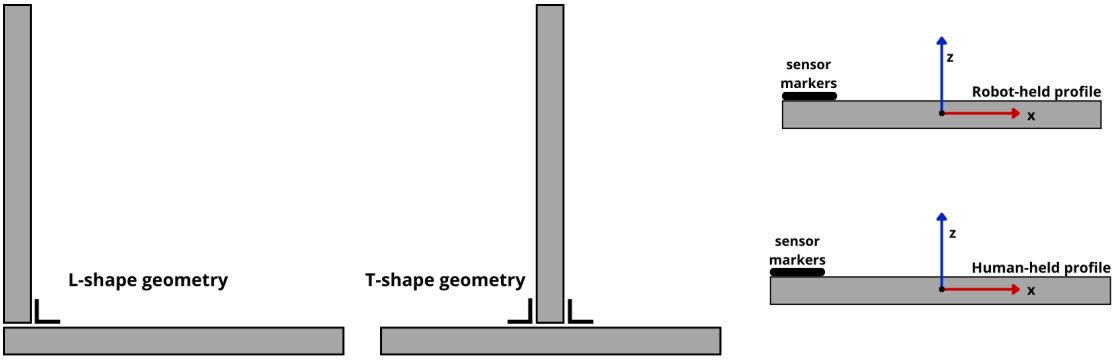


Figure 3.3: Assembly geometries and profile frames: L and T-shape assembly targets (left and middle). Profile coordinate frames and motion-capture marker placement (right). The x-axis runs along the profile toward the free end; z is the outward normal.

3.4 Motion Planning and Feasibility

The robot's actions are organized as a finite-state machine (FSM), a structure that breaks the task into a series of manageable steps. Each state defines a short motion toward an intermediate goal, such as approaching, grasping or aligning the profile. When the goal of one step is achieved, the system transitions smoothly to the next, ensuring that execution remains clear, traceable and easy to recover from in case of interruption.

The followed robot sequence is:

1. **Approach**: Move the end-effector above the assigned profile, pre-aligned for the picking action.
2. **Pre-grasp**: Move to the precise pose required for grasping.
3. **Grasp**: Close the gripper to grasp the robot's profile.
4. **Lift**: Lift the profile.
5. **Reorientation (Optional)**: Orient the profile vertical, when the vertical profile is assigned to the robot.
6. **Alignment**: Align the robot-held profile with the one held by the human operator.
7. **Assembly**: The robot keeps tracking the human-held profile, while the human operator assembles the two profiles together.

During Alignment, the target is continuously updated using the latest pose of the human-held profile, generating a new trajectory segment from the current robot pose to the new goal.

Before each motion, the system verifies that the next goal can be reached in the current scene. To check if the next goal is feasible, the target must be reachable, a safe path must be available and all joint limits must remain within bounds. If any of these conditions is not met, the robot enters a safe hold state and the operator is notified on the GUI. Feasibility is rechecked once new profiles' poses are received and, if the new goal becomes feasible, the workflow continues from where it stopped.

3.5 Impedance Control

A joint impedance controller [35], modeled after the Franka Panda joint-impedance example framework, is implemented. The controller receives the desired robot pose $(q_d, \dot{q}_d)$ for each segment and computes the commanded torque as follows:

$$\tau = c(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{K}_q(\mathbf{q}_d - \mathbf{q}) + \mathbf{D}_q(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}), \tag{3.4}$$

where $c(\mathbf{q}, \dot{\mathbf{q}})$ represents the model-based feedforward term (i.e., Coriolis and centrifugal effects, provided by the Franka model interface), and $\mathbf{K}_q$, $\mathbf{D}_q \in \mathbb{R}^{7\times7}$ are the diagonal stiffness and damping matrices, respectively. Gravity compensation for both the robot arm and gripper is handled by the Franka Panda's controller internally.

To promote smooth torque transitions, the applied torques are rate-limited:

$$\tau_i^{\text{sat}} = \tau_i^{\text{prev}} + \text{clip}\left(\tau_i - \tau_i^{\text{prev}}; -\Delta\tau_i^{\max}; \Delta\tau_i^{\max}\right), \tag{3.5}$$

with $\Delta\tau_i^{\max}$ specified individually for each joint.

The joint torques are then commanded via the effort interface.

Chapter 4

Hardware and Software Tools

4.1 Robotic Platform

4.1.1 Franka Emika Panda

The **Franka Emika Panda** robot (Figure 4.1), developed by **Franka Robotics GmbH** [36] (formerly *Franka Emika GmbH*), serves as the primary robotic platform for this project. This 7 DoF collaborative robotic arm integrates force/torque sensors on all joints, enabling precise force/torque control and making the Panda robot suitable for multiple tasks such as manipulation, grasping and assembly. With a weight of approximately 18 kg, the Panda robot can handle a maximum payload of 3 kg.
The Franka robot combines the precision of a classical stiff industrial manipulator, with a pose repeatability of ± 0.1 mm, enabling accurate and robust task execution. Its 1 kHz force control loop allows the application of very small forces, down to 0.05 N, supporting delicate operations such as insertion, screwing and contour tracking. In addition, gravity and friction compensating guiding modes facilitate smooth physical interaction between human and robot.

Joint space limits

Each joint of Franka Panda robot is subject to limits on position, velocity, acceleration, jerk and torque. Commands that violate any of these limits are rejected by the controller, resulting in a controlled stop of the robot to avoid damage.

The joint space limits are reported in Table 4.1.

- $q_{\max}$: Upper joint position limit (rad).
- $q_{\min}$: Lower joint position limit (rad).
- $|\dot{q}|_{\max}$: Maximum joint speed (rad/s).
- $|\ddot{q}|_{\max}$: Maximum joint acceleration (rad/s^2).
- $|\dddot{q}|_{\max}$: Maximum joint jerk (rad/s^3).

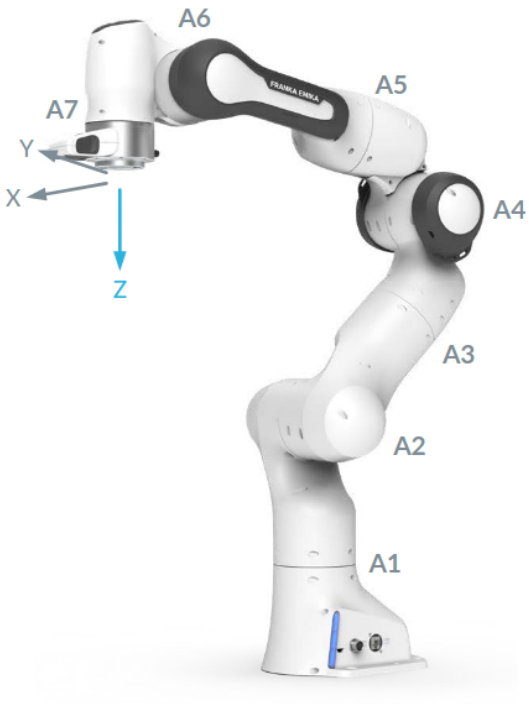


Figure 4.1: Franka Emika Panda robotic arm with seven revolute joints (A1–A7) and end-effector coordinate frame. Image reproduced from the Franka Emika robot's instruction handbook [37].

- $\tau_{j,\max}$: Maximum joint torque (N·m).

Name	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6	Joint 7	Unit
$q_{\max}$	2.8973	1.7628	2.8973	-0.0698	2.8973	3.7525	2.8973	rad
$q_{\min}$	-2.8973	-1.7628	-2.8973	-3.0718	-2.8973	-0.0175	-2.8973	rad
$\lvert\dot{q}\rvert_{\max}$	2.1750	2.1750	2.1750	2.1750	2.6100	2.6100	2.6100	rad/s
$\lvert\ddot{q}\rvert_{\max}$	15	7.5	10	12.5	15	20	20	rad/s^2
$\lvert\dddot{q}\rvert_{\max}$	7500	3750	5000	6250	7500	10000	10000	rad/s^3
$\tau_{j,\max}$	87	87	87	87	12	12	12	N·m

Table 4.1: Joints' position, velocity, acceleration, jerk and torque limits of Franka Panda robot [38].

Reachable workspace

The reachable workspace of the Franka Emika robot is defined by the combination of its joint limits (Figure 4.2). Horizontally, the arm can reach a maximum radius of 855 mm. Vertically,

the reachable range goes from -360 mm below up to 1190 mm above the mounting plane.

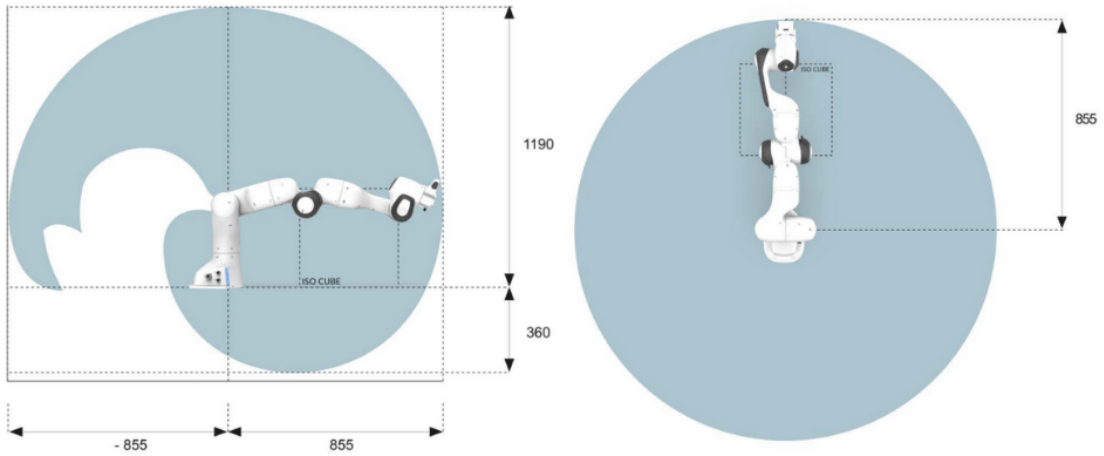


Figure 4.2: Franka Panda reachable space for the end-effector flange: side-view (left) and top-view (right) (dimensions in mm) [37].

Franka Control Interface

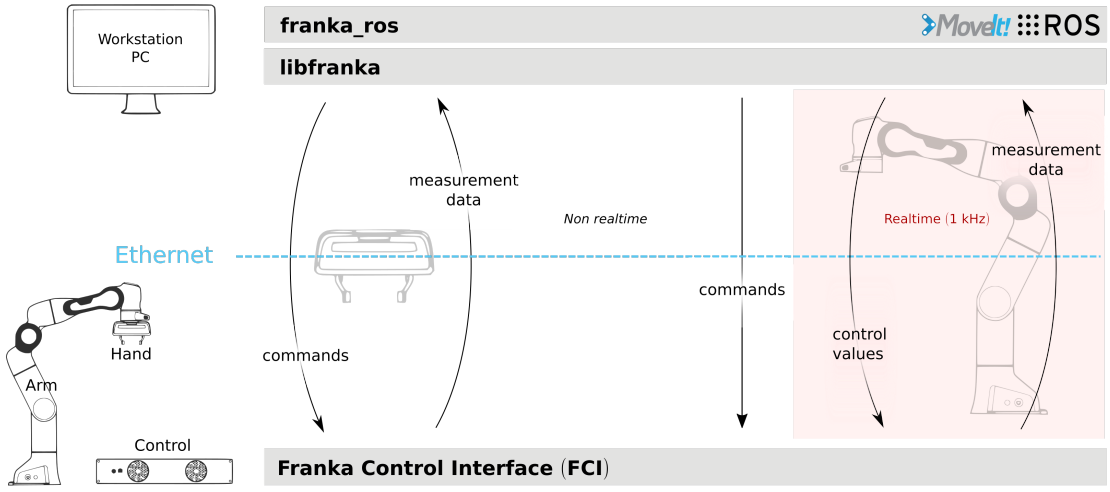


Figure 4.3: Data flow between the external workstation and the Panda robot through the Franka Control Interface (FCI) [38].

The **Franka Control Interface (FCI)** [38] is a fast, low-level bidirectional connection between the Panda arm and an external workstation (connected via Ethernet), providing the current status of the robot and enabling its direct control (Figure 4.3). Through the official ***libfranka***

C++ library (open-source library developed by Franka Robotics), the FCI allows real-time communication at 1 kHz, supporting different control modes such as gravity and friction compensated joint torques, joint position or velocity and Cartesian pose or velocity. At the same time, the user is provided with joint state measurements, estimations of externally applied torques and forces and collision or contact information. The interface also includes a robot model library with access to forward kinematics, Jacobians and dynamics quantities.

To interface the Franka robot with the ROS ecosystem, the ***franka_ros*** package is employed, as described in Section 4.2.1.

Franka Emika Robot Hand

The Panda robot [37] is supplied with an electrical two-finger parallel gripper (Figure 4.4) produced by Franka Robotics. This hand communicates directly through the connection to the arm and receives its power supply from the arm.
Its main technical specifications are summarized in Table 4.2.

Figure 4.4: Franka Emika Robot Hand.

Specification	Value
Weight	0.73 kg
Continuous grasping force	70 N
Maximum grasping force	140 N
Opening	80 mm
Finger speed	50 mm/s

Table 4.2: Franka Emika Robot Hand technical specifications [37].

The hand accepts commands for opening width (mm), speed (mm/s) and grasp force (N). Internal encoders track the commanded width and report status such as motion state and basic fault codes. In ROS, `franka_gripper` provides `Grasp`, `Move` and `Stop` actions with success and final width feedback. Parallel-jaw motion supports both precision pinches at small openings and stronger grasps near the 80 mm limit (Table 4.2). Flat fingertips are well suited for handling prismatic parts, whereas soft pads add friction, protect surfaces and offer slight compliance. Since the fingers move symmetrically, the grasp line stays aligned with the tool's **z** axis, which simplifies reasoning about the gripper pose during planning and execution. Continuous and peak forces are bounded by firmware and stall detection prevents damage. In collaborative use, pinch risks at the jaws are addressed in the risk assessment and motion should respect the active collaborative mode even though the gripper itself is not a safety-rated protective device. When fingertip geometry changes, the tool center point (TCP) should be re-identified to keep grasp planning consistent [37].

4.1.2 Robotiq 2F-85

The **Robotiq 2F-85** (Figure 4.5) is an adaptive two-finger parallel gripper designed for collaborative and industrial applications by **Robotiq Inc.** [39]. The main technical specifications are reported in Table 4.3.

Figure 4.5: Robotiq 2F-85 gripper.

Specification	Value
Weight	1 kg
Grasping force	20–235 N
Maximum opening	85 mm
Finger speed	20–150 mm/s

Table 4.3: Robotiq 2F-85 technical specifications [40].

It has a single actuator to open and close the fingers, which can adapt to the geometry of the object manipulated, which increases tolerance to pose error and supports stable grasps on a wider range of shapes than rigid jaws. The device requires an external power supply and a dedicated control interface. Typical commands specify position (opening in mm), speed (mm/s) and force (N), while feedback reports actual position, motor current and status bits for activation and faults. This enables "grasp until contact" behavior by monitoring current thresholds and final position windows. With the standard fingertips it covers an opening up to 85 mm and a force range of 20–235 N (Table 4.3), making it suitable for both delicate and more secure grasps in collaborative cells [40].

4.2 ROS

Robot Operating System (ROS) [41] is an open-source framework for developing robotic applications. Although it is called an "operating system", it is in fact a middleware running on top of Ubuntu Linux. It provides a high-level control layer that communicates with the lower-level interfaces of the robot.

ROS follows a distributed architecture in which independent processes, called *nodes*, perform specific tasks such as sensor acquisition, perception, planning or control. Nodes exchange information through a publish/subscribe mechanism based on *topics* and *messages*, or via direct client–server interactions using *services* and *actions*. The main advantage is the ability for developers to build and reuse code between robotics applications, reducing development time.

Applications in ROS are organized into *packages*, which provide drivers, libraries and algorithms for specific functionalities. ROS Noetic, the latest release of ROS1, includes a rich ecosystem of packages and tools widely adopted in both academia and industry.

In addition, ROS offers utilities for recording and replaying data streams, such as *rosbag*, which enable reproducible experiments and offline analysis.

4.2.1 franka_ros

The ***franka_ros*** [38] metapackage integrates *libfranka* into the ROS ecosystem and provides the necessary components to operate the Panda within a ROS-based framework. The *franka_ros* metapackage includes:

- *franka_description*: provides the description (in urdf) of the robot and its end-effectors, including kinematics, joint limits, visual meshes and collision models.
- *franka_hw*: the `ros_control` hardware interface wrapping `libfranka` for real-time control;
- *franka_control*: provides the *franka_control_node* and *franka_combined_control_node*, which are hardware interface nodes built on *franka_hw*. These nodes expose the complete *libfranka* API to ROS through a set of ROS services.
- *franka_example_controllers*: a collection of example controllers that demonstrate how to command the robot from ROS.
- *franka_gripper*: action-based interface for the Franka Hand (grasp, move, stop).

4.2.2 Visualization and Simulation

RViz

RViz (ROS Visualization) [42] is a 3D visualization tool included in ROS. It can display robot models, coordinate frames and sensor data in real time by subscribing to ROS topics. RViz is commonly used to monitor and debug robotic systems during development.

Gazebo

Gazebo [43] is an open-source robotics simulator that provides a 3D environment for testing and developing robotic systems. It supports complex scenarios, with physics-based interactions, sensors simulation and realistic robot behavior, allowing algorithms and controllers to be tested before deploying them on real hardware. A key feature of Gazebo is its seamless integration with ROS, which allows simulated sensors, environments and robots to interact directly with ROS nodes, making it an essential tool for robotics research and development.

4.3 OptiTrack Motion Capture System

OptiTrack [44] is an optical motion capture system developed by **NaturalPoint Inc.** for high-precision tracking of objects and subjects in three dimensions. It relies on infrared cameras and reflective spherical markers to reconstruct the six DoF pose of rigid bodies in real time. The system is managed through the **Motive** [45] software, which handles camera calibration, marker tracking and data streaming to external applications. Thanks to its accuracy and robustness, OptiTrack is widely used in fields such as animation, virtual reality and robotics.

4.4 Dynamics, Motion Planning and Optimization Libraries

This section summarizes the main libraries used for kinematics and dynamics computation, time-optimal trajectory generation and quadratic programming.

4.4.1 Pinocchio

Pinocchio [46] is an open-source C++ library for efficient rigid-body kinematics and dynamics with analytical derivatives. Built on Featherstone's algorithms [47], it provides fast routines for forward (direct) and inverse kinematics(B), inverse dynamics, Jacobians, articulated-body inertias and centroidal quantities. The implementation is designed for real-time use and exposes a clear model/data API that integrates readily with ROS and common robotics toolchains.

4.4.2 Ruckig

Ruckig [48] is a real-time trajectory generator that computes joint motions subject to position, velocity, acceleration and jerk limits. Given an initial and a target state (position, velocity, acceleration), it returns a time-optimal, time-parameterized profile that can be evaluated at the control rate. This makes it suitable both for offline segments (approach, pre-grasp, lift) and for short on-line updates during alignment, where goals are refreshed as the human-held profile moves. Using Ruckig keeps trajectories dynamically feasible while avoiding ad-hoc smoothing or manual timing.

4.4.3 qpOASES

qpOASES [49] is an open-source C++ solver for convex quadratic programs based on an online active-set method. The formulation is well suited to control applications, such as MPC (Model predictive control) or bounded optimizations, where similar problems are solved repeatedly with warm starts. The solver handles challenging instances (e.g., near singular or ill-conditioned QPs) with robust performance and provides bindings to external tools like MATLAB/Simulink for validation.

Chapter 5

Experiments

This chapter describes the experimental context: the hardware and software setup for the assembly task, a state-machine breakdown of the task, with pseudocode[1] for the main steps, and the experiments conducted, including task conditions, initial poses and recorded data.

The task consists of a cooperative assembly in which a human operator and a robot assemble aluminium profiles to form L or T-shaped geometries (Figure 3.3). The desired geometry and role assignment are selected via the GUI by the operator and the sequence is initiated by the robot at the start of each session.

5.1 Experimental Setup

5.1.1 Workspace overview

The workspace (Figure 5.1) consists of a robot arm with gripper, aluminium profiles, OptiTrack cameras, screws and corner brackets, used to complete the assembly of the two profiles. The profiles are *Profile 5 20×20* extrusions (20 mm × 20 mm, Line 5 with four open grooves), cut to 330 mm (robot-held) and 350 mm (human-held). This scene (robot and aluminum profiles) is also visualized on RViz (Figure 5.2).

5.1.2 Robot and Gripper Integration in ROS

The system was integrated in ROS Noetic. A custom ROS package `hri_assembly_task` was developed, which contains the launch files, configuration parameters, custom messages, controller plugin and nodes developed for this work. The detailed tree structure of the package and selected codes are reported in Appendix A.
The framework was evaluated on a 7-DoF Franka Emika Panda equipped with a Robotiq 2F-85 parallel gripper. The Robotiq was preferred to the Franka hand for its larger opening (85 mm vs. 80 mm), broader usable force range (20–235 N) and robust, interchangeable fingertip pads

[1] Concise, readable description of algorithm steps without language-specific syntax.

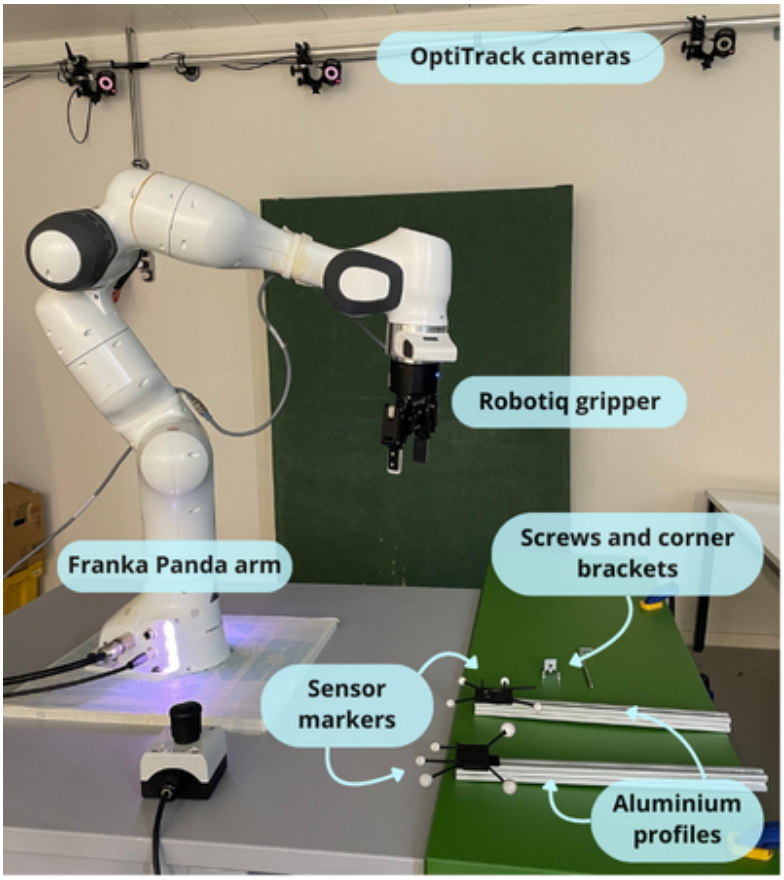


Figure 5.1: HRC workspace: Franka Emika Panda arm with a Robotiq gripper, OptiTrack motion-capture cameras, aluminium profiles instrumented with sensor markers and assembly hardware.

that improve friction and protect aluminium surfaces. It also integrates reliably with the ROS bridge for position and force commands, yielding consistent grasp behaviour during alignment. Standardizing on the Robotiq improved repeatability, reduced slip risk and minimized surface damage.

Following Siciliano's convention in *Robotics: Modelling, Planning and Control* [35], the end-effector frame $\{O_e; \mathbf{n}_e, \mathbf{s}_e, \mathbf{a}_e\}$ is defined at the fingertip of the Robotiq gripper:

$$\mathbf{n}_e \parallel +\mathbf{x}_{\texttt{panda\_link8}}, \qquad \mathbf{s}_e \parallel +\mathbf{y}_{\texttt{panda\_link8}}, \qquad \mathbf{a}_e \parallel +\mathbf{z}_{\texttt{panda\_link8}},$$

so that $(\mathbf{n}_e, \mathbf{s}_e, \mathbf{a}_e)$ is right-handed, with $\mathbf{a}_e$ the approach axis and $\mathbf{s}_e$ the finger-sliding axis. The gripper is mounted with a 45° rotation about the flange z-axis, and the fingertip origin O_e is obtained by translating 0.20 m along $+\mathbf{z}_{\texttt{panda\_link8}}$.

The Robotiq 2F-85 is interfaced through a thin Python driver and a C++ ROS bridge that embeds the Python runtime via `pybind11`. On the ROS side, the `gripper_node` loads the Python driver at launch and activates the gripper if needed. The desired width (in mm) is sent

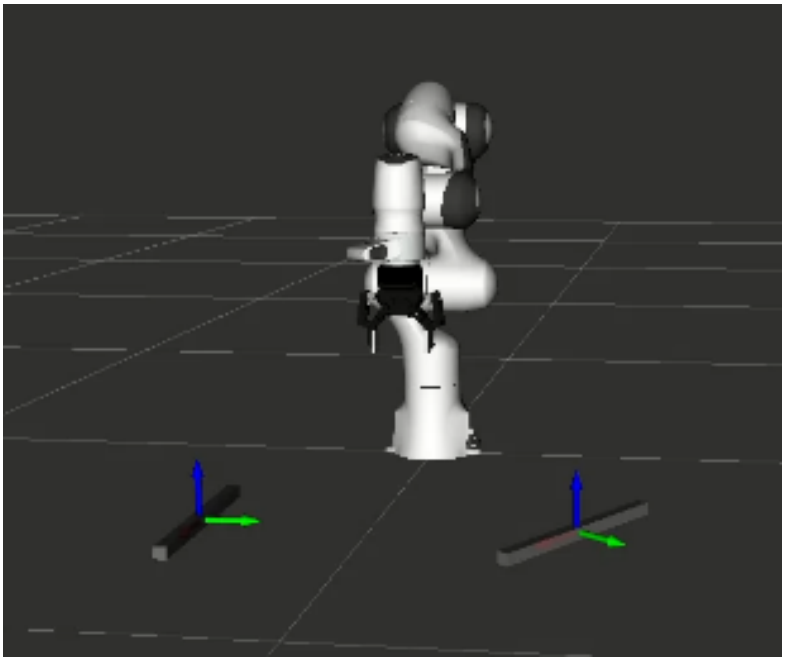


Figure 5.2: RViz snapshot of the assembly scene with the Panda robot, robot-held and human-held profiles, at beginning of Experiment 1.

by publishing on `/gripper_node/command` from the controller. When a new command arrives, the node calls the driver's `go_to(width, 100.0, 100.0)` function, with speed and force fixed to 100 mm/s and 100 N. The command is acknowledged on `/gripper_node/handshake` and completion feedback is later published on `/gripper_node/feedback` as one of `home_reached`, `close_reached`, `grasp_reached` or `gripper_failure`, based on whether the current opening matches the goal within 1 mm.

5.1.3 Motion Capture System and Frame Calibration

An OptiTrack motion-capture system was used to track the pose of the aluminium profiles. The OptiTrack system consisted of 18 OptiTrack s250e cameras, synchronized and operating real-time detections at 250 fps. A rigid body marker base was attached with Velcro at one end of each profile (Figure 3.3, 5.1). To capture the pose, the robot-held profile and the human-held profile have respectively 4 and 5 reflective spherical markers.

The profile frames were defined in the following way: the 3D workspace covered by the OptiTrack cameras was first calibrated to define the OptiTrack world frame. Since the robot base frame (`panda_link0`) in the OptiTrack world frame was unknown, reflective markers were rigidly attached to the robot base to create a temporary "OptiTrack base" (`panda_base_OptiTrack`). The transform between `panda_base_OptiTrack` and `panda_link0` was then computed by positioning the Franka hand fingertip over a reference marker and comparing the marker pose (with respect to `panda_base_OptiTrack`) with the gripper pose (`panda_hand_tcp`). Applying this transform, each OptiTrack body frame was expressed with respect to the `panda_link0`

frame. With RViz it was possible to verify if the transform was correct. After this calibration, both profiles were defined as rigid bodies in Motive, with their local frames adjusted so that the origin lay near the geometric center (Figure 3.3).
To clarify notation and axis conventions, the coordinate frames used throughout the chapter are depicted in Figure 5.3:

- world: $[\mathbf{x}_{\text{world}}\, \mathbf{y}_{\text{world}}\, \mathbf{z}_{\text{world}}]$ at `panda_link0`
- end-effector: $[\mathbf{x}_{ee}\, \mathbf{y}_{ee}\, \mathbf{z}_{ee}]$
- robot-held: $[\mathbf{x}_{r}\, \mathbf{y}_{r}\, \mathbf{z}_{r}]$
- human-held: $[\mathbf{x}_{h}\, \mathbf{y}_{h}\, \mathbf{z}_{h}]$

All frames are right-handed.

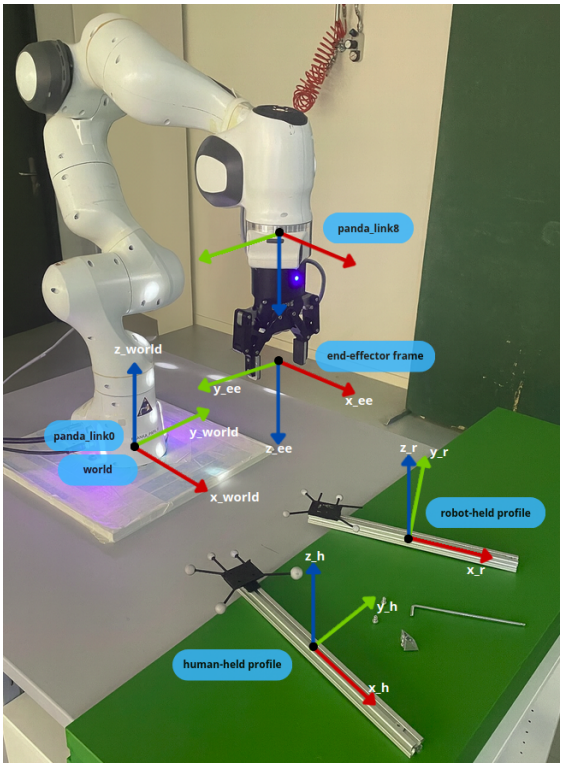


Figure 5.3: Coordinate frames used in the workspace: world (at panda_link0), flange (panda_-link8), end-effector and the local frames of the robot-held and human-held profiles.

5.1.4 Motion Planning

Motion planning paths are converted into joint space trajectories using Ruckig. Outside the alignment state, trajectories are computed offline and executed from start to end. During alignment, the goal pose (coming from motion capture human-held profile pose) is updated at 1 Hz.

At each update, a new trajectory segment is generated from the current robot state to the updated goal and the active trajectory is replaced accordingly. This process keeps the motion continuous, while respecting Franka's timing constraints.

5.2 State-Machine breakdown

The following sections describe each step of the sequence and provide pseudocode for the most relevant parts. The complete source code is reported in Appendix A.

Approach

In the APPROACH state, the robot moves its end-effector from its initial configuration to a pose above the robot-held profile (Figure 5.4).

The target pose is derived from the OptiTrack estimate of the profile: the grasp coordinate x_g is chosen along the profile axis $\mathbf{x}_r$ and the position is offset by 0.20 m along $\mathbf{z}_r$ (Algorithm 1).

The admissible grasp interval $[x_{\min}, x_{\max}]$ is set by adding safety margins at both ends; depending on the assignment, a preferred x_g is placed near $x_{\min}$ for a horizontal robot-held profile or near $x_{\max}$ for a vertical one, with a small δ to avoid sensor markers. The final x_g is then obtained by solving a bounded QP with `qpOASES` so it remains inside the graspable region (Algorithm 2).

The end-effector orientation for picking is pre-defined during this state. The approach axis of the end-effector $\mathbf{z}_{ee}$ is aligned with the negative direction of the world axis $\mathbf{z}_{world}$, while the normal axis $\mathbf{x}_{ee}$ is aligned with the robot-held profile axis $|\mathbf{x}_r|$. To avoid unnecessary rotations close to 180°, the alignment is chosen so that the signed angle θ between $\mathbf{x}_{ee}^{\text{start}}$ and $\mathbf{x}_r$, measured about the reference axis $\mathbf{r}$ (in this case $\mathbf{z}_{world}$), does not exceed 90° (Eq. 5.1).

$$\mathbf{x}_{ee} = \begin{cases} \mathbf{x}_r, & \text{if } \theta < \frac{\pi}{2}, \\ -\mathbf{x}_r, & \text{if } \theta \geq \frac{\pi}{2}, \end{cases} \qquad \mathbf{z}_{ee} = -\mathbf{z}_{\text{world}}, \qquad \mathbf{y}_{ee} = \mathbf{z}_{ee} \times \mathbf{x}_{ee}, \tag{5.1}$$

$$\theta = \angle\left(\mathbf{x}_{ee}^{\text{start}}, \mathbf{x}_r\right).$$

Algorithm 3 shows how the state is initialized: the initial joint state $\mathbf{q}_{\text{start}}$ is stored, the grasp coordinate is obtained with POSITIONGRASPQP (Algorithm 2) and the corresponding target configuration is computed with INITPICKOBJECT (Algorithm 1). If a valid inverse kinematics solution is not found ($\mathbf{q}_{\text{target}}$ is empty) or if a trajectory cannot be generated, the state transitions directly to ERROR. Otherwise, Ruckig is used to generate an offline joint-space trajectory from $\mathbf{q}_{\text{start}}$ to $\mathbf{q}_{\text{target}}$. While the trajectory is active, UPDATEOFFLINETRAJECTORY advances it at the control rate (1 kHz). When the trajectory finishes, the next state becomes PRE-GRASP.

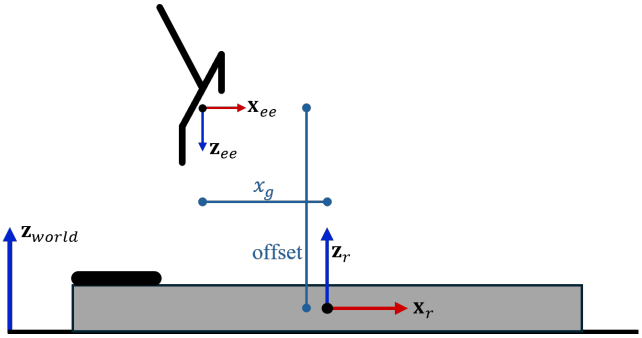


Figure 5.4: Target end-effector pose in APPROACH in the case: $\mathbf{x}_{ee} = +\mathbf{x}_r$.

Algorithm 1 INITPICKOBJECT pseudocode

$\mathbf{z}_{ee} \leftarrow -\mathbf{z}_{\text{world}}$
if $(\angle(\mathbf{x}_{ee}^{\text{start}}, \mathbf{x}_r) < \pi/2)$ **then**
    $\mathbf{x}_{ee} \leftarrow \mathbf{x}_r$
else
    $\mathbf{x}_{ee} \leftarrow -\mathbf{x}_r$
end if
$\mathbf{y}_{ee} \leftarrow \mathbf{z}_{ee} \times \mathbf{x}_{ee}$
$\mathbf{R}_{\text{approach}} \leftarrow \left[\mathbf{x}_{ee}\ \mathbf{y}_{ee}\ \mathbf{z}_{ee}\right]$
$\text{offset} \leftarrow (0.0, 0.0, 0.2)$
$\mathbf{p}_{\text{approach}} \leftarrow \mathbf{p}_r + \mathbf{R}_r\,[x_g, 0, 0]^T + \text{offset}$
Solve IK for $(\mathbf{R}_{\text{approach}}, \mathbf{p}_{\text{approach}})$ to get $\mathbf{q}_{\text{target}}$
return $[q]_{\text{target}}$

Algorithm 2 POSITIONGRASPQP pseudocode

$\delta \leftarrow 0.02$
$x_{\min} \leftarrow -x_{\text{tot}}/2 + 0.07$
$x_{\max} \leftarrow x_{\text{tot}}/2 - 0.08$
if *task_assignment* = HumV_RobH[2] **then**
    $x_g \leftarrow x_{\min} + \delta$
else if *task_assignment* = HumH_RobV[3] **then**
    $x_g \leftarrow x_{\max} - \delta$
end if
Solve bounded QP around x_g
return x_g

[2]HumV_RobH = human-held vertical profile, robot-held horizontal profile.

[3]HumH_RobV = human-held horizontal profile, robot-held vertical profile.

Algorithm 3 APPROACH STATE pseudocode

```
if approach_initialized is false then
    q_start ← q_current
    x_g ← POSITIONGRASPQP
    q_target ← INITPICKOBJECT
    if q_target is empty then
        next_state ← ERROR;
        return
    end if
    if trajectory_initialized is false then
        next_state ← ERROR;
        return
    end if
    approach_initialized ← true
else
    if advance offline trajectory then
        next_state ← PRE-GRASP
    end if
end if
```

Pre-Grasp

In the PRE-GRASP state, the end-effector moves vertically downward to position the gripper around the robot-held profile (Algorithm 4) (Figure 5.7). The target is defined by lowering the end-effector by 0.20 m in the world frame while maintaining the current orientation. The corresponding target configuration $\mathbf{q}_{\text{target}}$ is obtained through inverse kinematics and an offline trajectory is planned with Ruckig. If trajectory initialization fails, the state transitions to ERROR. Otherwise, the trajectory is executed to completion and the system then transitions to GRASP.

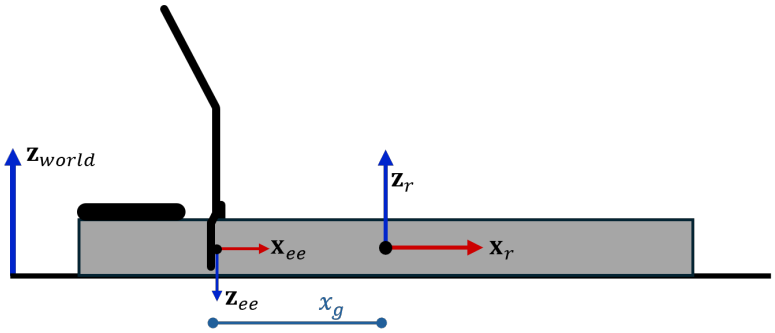


Figure 5.5: Target end-effector pose in PRE-GRASP in the case: $\mathbf{x}_{ee} = +\mathbf{x}_r$.

Algorithm 4 PRE-GRASP STATE pseudocode

if *pre_grasp_initialized* is **false then**
  $\mathbf{q}_{\text{start}} \leftarrow \mathbf{q}_{\text{current}}$
  $\mathbf{p}_{\text{target}} \leftarrow \mathbf{p}_{\text{current}} - (0,0,0.20)$
  Solve IK for $(\mathbf{R}_{\text{current}}, \mathbf{p}_{\text{target}})$ to get $\mathbf{q}_{\text{target}}$
  if *trajectory_initialized* is **false then**
    next_state $\leftarrow$ ERROR;
    return
  end if
  pre_grasp_initialized $\leftarrow$ **true**
else
  if update offline trajectory **then**
    next_state $\leftarrow$ GRASP
  end if
end if

Grasp

In the GRASP state, the gripper closes the fingers, grasping the profile. The controller publishes on topic `/gripper_node/command` the desired command and the `gripper_node` executes:

$$\begin{cases} \text{width} = 20 \text{ mm}, \\ \text{speed} = 100 \text{ mm/s}, \\ \text{force} = 100 \text{ N} \end{cases} \tag{5.2}$$

Grasp completion is then signaled via `/gripper_node/feedback` and the state transitions to LIFT. Implementation details of the gripper driver and ROS bridge are provided in Sec. 5.1.2.

Lift

In the LIFT state, the robot lifts the profile to a safe height before proceeding to alignment or reorientation (Algorithm 5) (Figure 5.6). The target pose is obtained by translating the current end-effector position upward by 0.15 m in the world space. The inverse kinematics is solved to obtain the target pose and Ruckig generates the corresponding trajectory. If the robot is assigned the horizontal profile, the next state is ALIGNMENT, while if it is assigned the vertical profile, the next state is REORIENTATION.

Reorientation

The REORIENTATION state is entered only when the robot is assigned the *vertical* profile (Figure 5.7). Its purpose is to introduce an intermediate, task-aware orientation (Algorithm 6) so that, during assembly, the robot does not need to perform large rotations. In practice, this predisposes the robot-held profile to minimize later motion and reduce the chance of kinematic dead-ends.

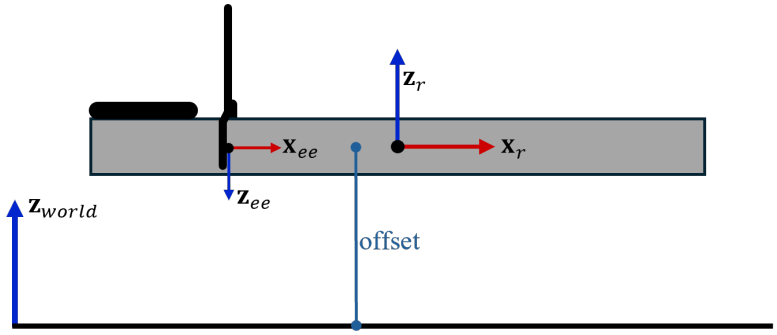


Figure 5.6: Target end-effector pose in LIFT in the case: $\mathbf{x}_{ee} = +\mathbf{x}_r$.

Algorithm 5 LIFT STATE pseudocode

```
1:  if lift_initialized is false then
2:      q_start ← q_current
3:      p_target ← p_current + (0,0,0.15)
4:      Solve IK for (R_current, p_target) to get q_target
5:      if trajectory_initialized is false then
6:          next_state ← ERROR;
7:          return
8:      end if
9:      lift_initialized ← true
10: else
11:     if update offline trajectory then
12:         if task_assignment = HumV_RobH then
13:             next_state ← ALIGNMENT
14:         else if task_assignment = HumH_RobV then
15:             next_state ← REORIENTATION
16:         end if
17:     end if
18: end if
```

Hand dominance is asked to determine on which side the contact edge is placed for screwing, to make the task more comfortable for the human operator. For example, on the front table side: if the operator is right-handed, the robot-held profile is brought to the left edge of the human-held profile. The operator is instructed, via GUI, to present their profile with the sensor edge on the right side. The same reasoning applies symmetrically for left-handed users and for the other table sides.

The table side is also used to pre-assign the end-effector approach axis before the ALIGNMENT state, so that the robot arrives already oriented with respect to the workspace (e.g., choosing $\mathbf{z}_{ee} = \mathbf{x}_{\text{world}}$ or $\pm\mathbf{y}_{\text{world}}$ depending on side and hand). This pre-alignment reduces the rotation demanded during ALIGNMENT, making the subsequent pose updates smoother and less aggressive.

Once the robot-held profile becomes vertical, the next state is ALIGNMENT.

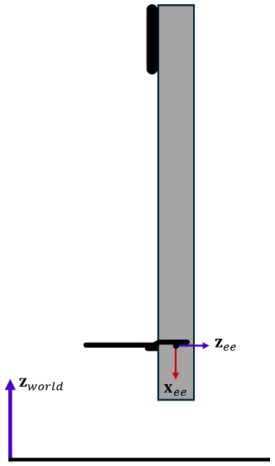


Figure 5.7: Target end-effector pose in REORIENTATION.

Algorithm 6 INITVERTICALOBJECT pseudocode

if $\angle(\mathbf{x}_{ee}^{\text{start}}, \mathbf{x}_r) < \pi/2$ **then**
    $\mathbf{x}_{ee} \leftarrow -\mathbf{z}_{\text{world}}$
else
    $\mathbf{x}_{ee} \leftarrow \mathbf{z}_{\text{world}}$
end if
if *hand* = `right` **then**
    if *table_side* = `front` **then**
        $\mathbf{z}_{ee} \leftarrow \mathbf{y}_{\text{world}}$
    else if *table_side* = `side` **then**
        $\mathbf{z}_{ee} \leftarrow -\mathbf{y}_{\text{world}}$
    else if (*table_side* = `angle`) **then**
        $\mathbf{z}_{ee} \leftarrow \mathbf{x}_{\text{world}}$
    end if
else if *hand* = `left` **then**
    $\mathbf{z}_{ee} \leftarrow \mathbf{x}_{\text{world}}$
end if
$\mathbf{y}_{ee} \leftarrow \mathbf{z}_{ee} \times \mathbf{x}_{ee}$
$\mathbf{R}_{\text{vertical}} \leftarrow [\mathbf{x}_{ee}\ \mathbf{y}_{ee}\ \mathbf{z}_{ee}]$
$\mathbf{p}_{\text{vertical}} \leftarrow \mathbf{p}_r$
Solve IK for $(\mathbf{R}_{\text{vertical}}, \mathbf{p}_{\text{vertical}})$ to get $\mathbf{q}_{\text{target}}$
return $\mathbf{q}_{\text{target}}$

Alignment

In the ALIGNMENT state, the robot approaches the human-held profile and continuously adjusts its pose until contact (Algorithm 7) (Figure 5.8). The target is *retargeted* at 1 Hz from the latest OptiTrack estimate of the human-held profile: at each update, the current robot state $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is taken as a new start, inverse kinematics is solved for the new alignment pose and a fresh Ruckig segment is generated that replaces the current one. This preserves continuous motion while restricting replanning to short segments. If IK or trajectory initialization fails, the state transitions to ERROR, otherwise the trajectory executes to completion and the system moves to DONE.

Algorithm 8 constructs the alignment pose according to the task assignment: if the vertical profile is held by the human, the end-effector axes are set $\mathbf{x}_{ee} \leftarrow -\mathbf{z}_h$ and $\mathbf{z}_{ee} \leftarrow \mathbf{x}_h$ (with $\mathbf{y}_{ee} = \mathbf{z}_{ee} \times \mathbf{x}_{ee}$). If the vertical profile is held by the robot, two approach variants are considered (`from_x_r`, `from_y_r`) as introduced in the REORIENTATION section. The choice of $\mathbf{z}_{ee}$ depends on table side and hand dominance.

The target alignment position of the robot-held profile is computed based on the human-held profile position, to which offsets based on the geometries of the profile are added:

$$\begin{aligned} \mathbf{p}_{\text{align}} &= \mathbf{p}_h + \text{off}_{z_h} + \text{off}_{x_h} \\ \text{off}_{z_h} &= \tfrac{x_r}{2} - x_g + \tfrac{z_h}{2} \\ \text{off}_{x_h} &= \tfrac{x_h}{2} - \tfrac{z_r}{2} \end{aligned} \tag{5.3}$$

$(\mathbf{R}_{\text{align}}, \mathbf{p}_{\text{align}})$ are then passed to IK. (Algorithm 7,8).

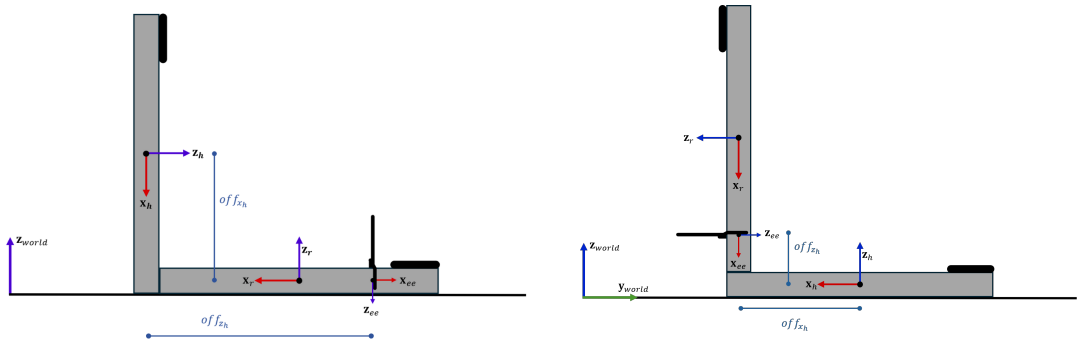


Figure 5.8: Target end-effector pose in ALIGNMENT. On the left is the human-held vertical profile case and on the right the human-held horizontal profile case.

Assembly

Once the two profiles are in contact, the ASSEMBLY state begins. In this state, the human operator can assemble them together, screwing the two parts. If there is a slight discrepancy between the two profiles, the operator can help the robot by adjusting the robot-held profile and make the assembly easier.

Algorithm 7 INITALIGNOBJECT pseudocode

approaches $\leftarrow$ {`from_x_r`, `from_y_r`}
for all $a \in$ approaches **do**
  if *task* = `HumV_RobH` **then**
    $\mathbf{x}_{ee} \leftarrow -\mathbf{z}_h$
    $\mathbf{z}_{ee} \leftarrow \mathbf{x}_h$
  else if *task* = `HumH_RobV` **then**
    if $\angle(\mathbf{x}_{ee}^{\text{start}}, \mathbf{z}_h) \leq \pi/2$ **then**
      $\mathbf{x}_{ee} \leftarrow \mathbf{z}_h$
    else
      $\mathbf{x}_{ee} \leftarrow -\mathbf{z}_h$
    end if
    if a = `from_y_r` **then**
      if *hand* = `left` **then**
        $\mathbf{z}_{ee} \leftarrow -\mathbf{y}_h$
      else
        $\mathbf{z}_{ee} \leftarrow \mathbf{y}_h$
      end if
    else
      $\mathbf{z}_{ee} \leftarrow -\mathbf{x}_h$
    end if
  end if
  $\mathbf{y}_{ee} \leftarrow \mathbf{z}_{ee} \times \mathbf{x}_{ee}$
  $\mathbf{R}_{\text{align}} \leftarrow [\mathbf{x}_{ee}\ \mathbf{y}_{ee}\ \mathbf{z}_{ee}]$
  $\text{off}_{z_h} \leftarrow$ offset to align along $\mathbf{z}_h$
  $\text{off}_{x_h} \leftarrow$ offset to align along $\mathbf{x}_h$
  $\mathbf{p}_{\text{align}} \leftarrow \mathbf{p}_h + \text{off}_{z_h} + \text{off}_{x_h}$
  Solve IK for $(\mathbf{R}_{\text{align}}, \mathbf{p}_{\text{align}})$ to get $\mathbf{q}_{\text{target}}$
end for
return $\mathbf{q}_{\text{target}}$

Error

The ERROR state is entered whenever a trajectory cannot be initialized, a valid inverse kinematics solution cannot be found or a critical failure occurs in another state. In this state, the robot stops and waits for user intervention as most of the problems come from the position of the profiles, because the robot trajectories to reach them are not feasible.

5.3 Experiments

Seven experiments (Table 5.1) were conducted under a single task family by one right-handed human operator: L-shape assembly with the robot assigned to the horizontal profile (the operator holds the vertical one).

Algorithm 8 ALIGNMENT STATE pseudocode

if *alignment_initialized* is **false** $\vee$ *retarget* is **true then**
  $\mathbf{q}_{\text{start}} \leftarrow \mathbf{q}_{\text{current}}$
  $\dot{\mathbf{q}}_{\text{start}} \leftarrow \dot{\mathbf{q}}_{\text{current}}$
  $\ddot{\mathbf{q}}_{\text{start}} \leftarrow \ddot{\mathbf{q}}_{\text{current}}$
  $\mathbf{q}_{\text{target}} \leftarrow$ INITALIGNOBJECT
  if $\mathbf{q}_{\text{target}}$ is empty **then**
    next_state $\leftarrow$ ERROR;
    return
  end if
  if *trajectory_initialized* is **false then**
    next_state $\leftarrow$ ERROR;
    return
  end if
  alignment_initialized $\leftarrow$ **true**
else if *alignment_initialized* is **true** $\wedge$ *retarget* is **false then**
  if update offline trajectory **then**
    next_state $\leftarrow$ DONE
  end if
end if

Two starting poses were defined for each profile at the beginning of the tracking. The second pose H2 for the human-held profile was intentionally tilted from the vertical position to assess the alignment stage under a mild orientation discrepancy. Four experiments covered all pairings of these choices so that each robot-held profile pose was tested once with each human-held profile pose.
The remaining three experiments fixed the profile pair at one of those arrangements (R1-H1) and varied the robot's initial configuration. In Table 5.1, the start configurations are summarized.

Exp	Robot pose	Robot-held profile pose	Human-held profile pose
1	A	R1	H1
2	A	R1	H2
3	A	R2	H1
4	A	R2	H2
5	B	R1	H1
6	C	R1	H1
7	D	R1	H1

Table 5.1: Start configurations used in the experiments: robot start configuration (A–D) and initial profile poses. Labels R1–R2 and H1–H2 denote the initial poses of the robot-held and human-held profiles, respectively.

All trials were executed by a single, right-handed operator.

In the case of the robot assigned to the vertical profile, when the robot-held profile was maintained strictly vertical, the arm was brought close to wrist singularities, stopping the robot. This case was therefore excluded from experiments with real robot.

Chapter 6

Results

This chapter reports the experimental results. Seven L-shape assembly experiments were executed with the robot assigned to the horizontal profile. The hardware evaluation examines each experiment with photographs of the main phases and five plots per experiment are reported: end-effector tracking, joint positions, joint velocities, commanded vs. measured joint torques and external wrench at the end-effector. The chapter closes with an overall analysis across experiments.

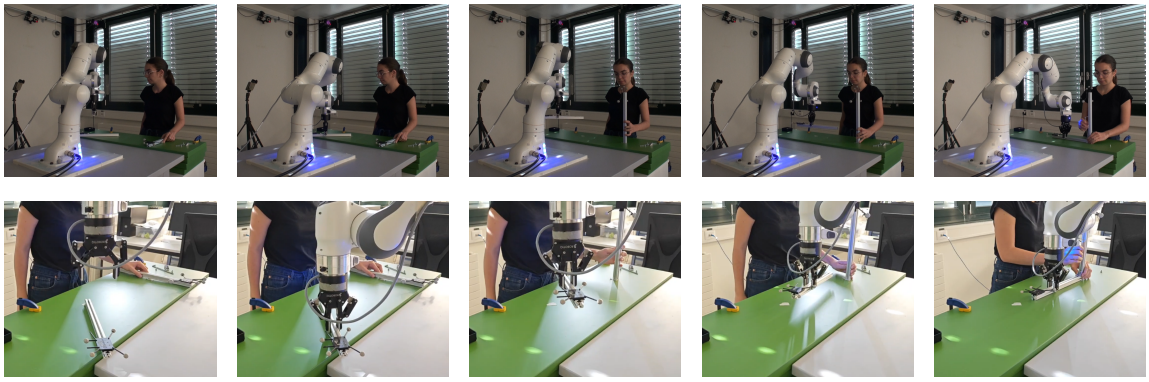
6.1 Hardware evaluation

The experiments analyzed in this section are the assembly experiments introduced in Section 5.3. For each experiment, telemetry was logged using `rosbag` at the robot control rate. The following signals were recorded:

- joint positions and velocities $(\mathbf{q}, \dot{\mathbf{q}})$ from the Franka state interface;
- desired joint positions and velocities $(\mathbf{q}_d, \dot{\mathbf{q}}_d)$ from the Ruckig trajectories;
- measured joint torques (τ) from the Franka state interface;
- commanded joint torques (τ_{cmd}) from the impedance controller;
- measured end-effector poses from the Franka state interface;
- desired end-effector poses from the Ruckig trajectories;
- estimated external wrench at the end-effector from the Franka state interface.

Experiment 1 - R1-H1, pose A

Experiment 1 corresponds to configuration R1–H1 with the robot in pose A (see Table 5.1). The robot is assigned to the horizontal profile, while the operator holds the vertical one. Figure 6.1 illustrates the important stages of Experiment 1 from two viewpoints.



(a) (b) (c) (d) (e)

Figure 6.1: Experiment 1 illustrated with photographs of the main stages: (a) APPROACH, (b) GRASP, (c) LIFT, (d) ALIGNMENT and (e) ASSEMBLY. For each stage, images from two viewpoints are shown.

Figure 6.2 (a) compares the desired vs. measured end-effector trajectory along the three Cartesian axes. The robot tracks the reference closely during the pre-contact phase (until t $\approx$ 9 s), whereas during LIFT and ALIGNMENT there are deviations below a few millimeters. The largest error occurs in x during the ALIGNMENT state and z during LIFT and ALIGNMENT, consistent with hand tremor of the human-held profile.

Figures 6.2 (b) and 6.2 (c) show, respectively, the joint space position and velocity trajectories.

The desired joint positions are not smooth polynomials but linear ramps: this comes from the Ruckig trajectory generator, which computes time-optimal trajectories under joint position, velocity, acceleration and jerk limits. As a result, the reference trajectories consist of concatenated linear segments. Joint positions track the references closely, with errors well below one degree on all joints. Within each state, joint velocities are nearly constant, showing only small oscillations typical of the Franka controller in practice. During ALIGNMENT, the velocities decay smoothly to zero, indicating that the robot has reached the final pose relative to the human-held profile.

Figure 6.2 (d) compares commanded and measured joint torques. Commanded torques do not consider the gravity compensation of the Franka arm and of the Robotiq gripper, as it is done automatically by Franka interface. The second joint carries most of the load, consistent with its role in supporting gravity during LIFT. Torque demands remain below 50 Nm, well inside actuator limits.

Figure 6.2 (e) shows the measured external wrench at the end-effector (forces F_x, F_y, F_z and torques T_x, T_y, T_z). With the Robotiq gripper configured in Franka Desk, tool gravity is compensated. The lateral force F_x remains small ($\approx$1–2 N) even during ALIGNMENT, consistent with light side contact, whereas F_z rises by $\approx$5–6 N during LIFT due to the weight of the profile.

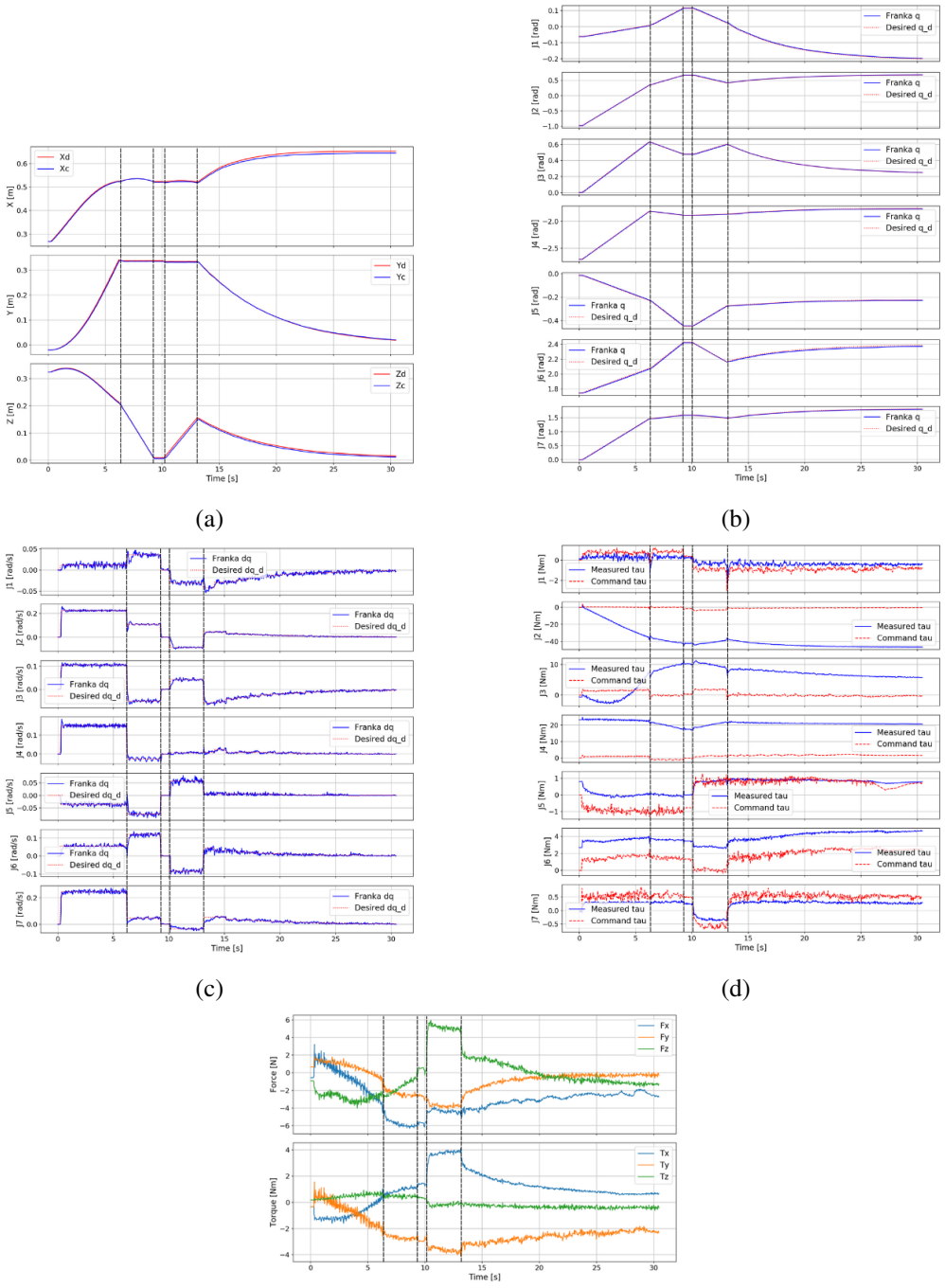
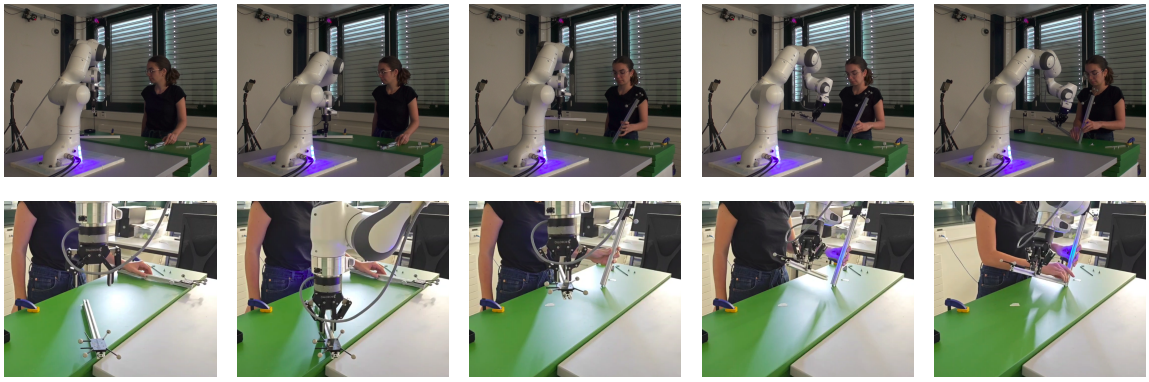


Figure 6.2: Experiment 1 results. (a) Desired vs. real end-effector positions. (b) Joint positions q vs. q_d. (c) Joint velocities $\dot{q}$ vs. $\dot{q}_d$. (d) Commanded vs. measured joint torques. (e) Estimated external wrench at the end-effector. Vertical dashed lines mark phase boundaries: APPROACH (0–6 s), PRE-GRASP (6–9 s), GRASP (9–10 s), LIFT (10–13 s), ALIGNMENT ($>$ 13 s).



(a) (b) (c) (d) (e)

Figure 6.3: Experiment 2 illustrated with photographs of the main stages: (a) APPROACH, (b) GRASP, (c) LIFT, (d) ALIGNMENT and (e) ASSEMBLY. For each stage, images from two viewpoints are shown.

Experiment 2 - R1-H2, pose A

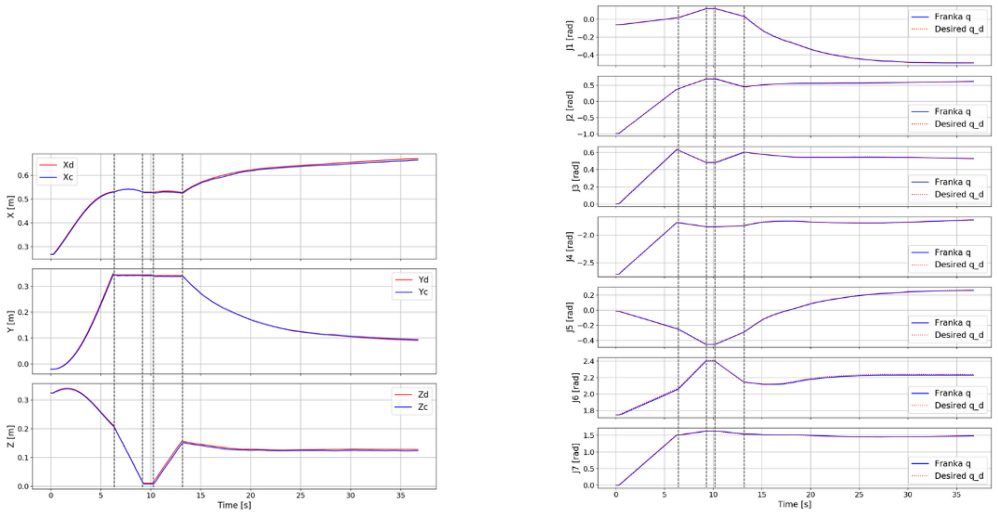
Experiment 2 corresponds to the configuration R1–H2 with the robot in pose A (Table 5.1). The robot is assigned to the horizontal profile, while the operator holds the vertical one, this time tilting it slightly to test the robot's behavior when the human-held profile is not perfectly vertical.

The two viewpoints of the different states are shown in Figure 6.3.

Figure 6.4(a) compares the desired and measured end-effector motion along the three Cartesian axes. Tracking is tight in the pre-contact phase (up to $t \approx 9\,\mathrm{s}$), while the largest deviations appear on z during LIFT and ALIGNMENT.

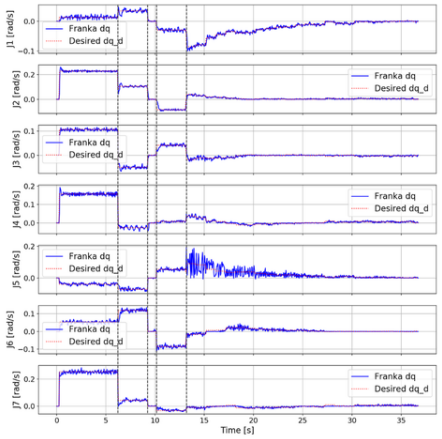
Joint positions (Figure 6.4(b)) follow their references with high precision. Joint velocities (Figure 6.4(c)) show more pronounced oscillations in ALIGNMENT, particularly on joint 5, likely reflecting hand tremor as the operator maintains the tilted profile. Velocities decay smoothly to zero after $t \approx 27\,\mathrm{s}$, consistent with the robot reaching the final relative pose.

Joint torques (Figure 6.4(d)) exhibit the same pattern, with larger oscillations on joint 5 attributable to the same source. The external wrench (Figure 6.4(e)) shows a rise in F_x during Alignment that decreases as contact stabilizes.

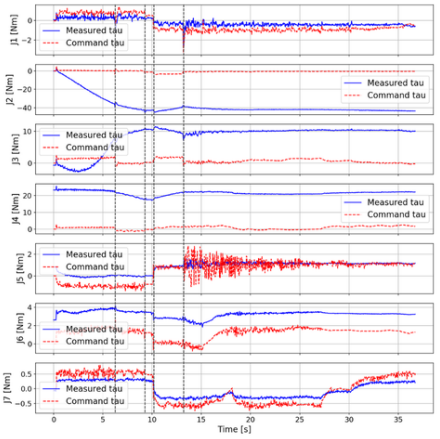


(a)

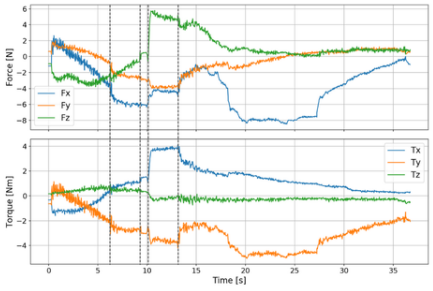
(b)



(c)

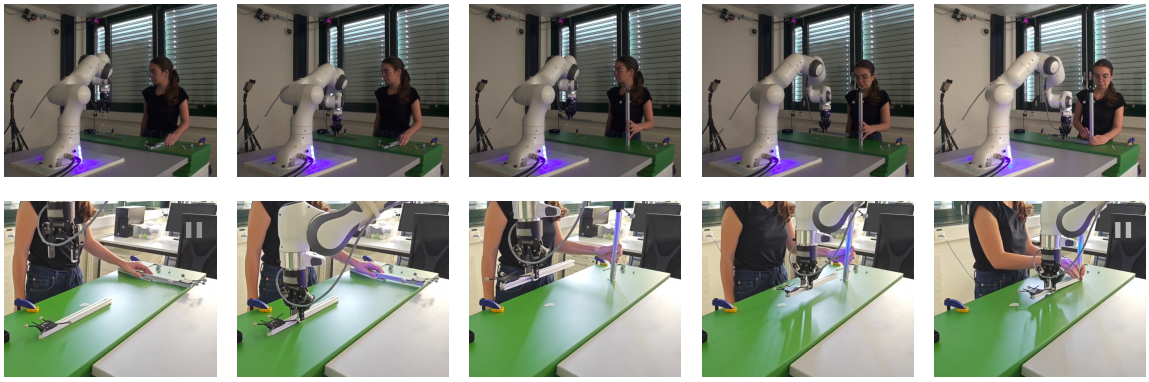


(d)



(e)

Figure 6.4: Experiment 2 results. (a) Desired vs. real end-effector positions. (b) Joint positions q vs. q_d. (c) Joint velocities $\dot{q}$ vs. $\dot{q}_d$. (d) Commanded vs. measured joint torques. (e) Estimated external wrench at the end-effector. Vertical dashed lines mark phase boundaries: APPROACH (0–6 s), PRE-GRASP (6–9 s), GRASP (9–10 s), LIFT (10–13 s), ALIGNMENT ($>$ 13 s).



(a) (b) (c) (d) (e)

Figure 6.5: Experiment 3 illustrated with photographs of the main stages: (a) APPROACH, (b) GRASP, (c) LIFT, (d) ALIGNMENT and (e) ASSEMBLY. For each stage, images from two viewpoints are shown.

Experiment 3 - R2-H1, pose A

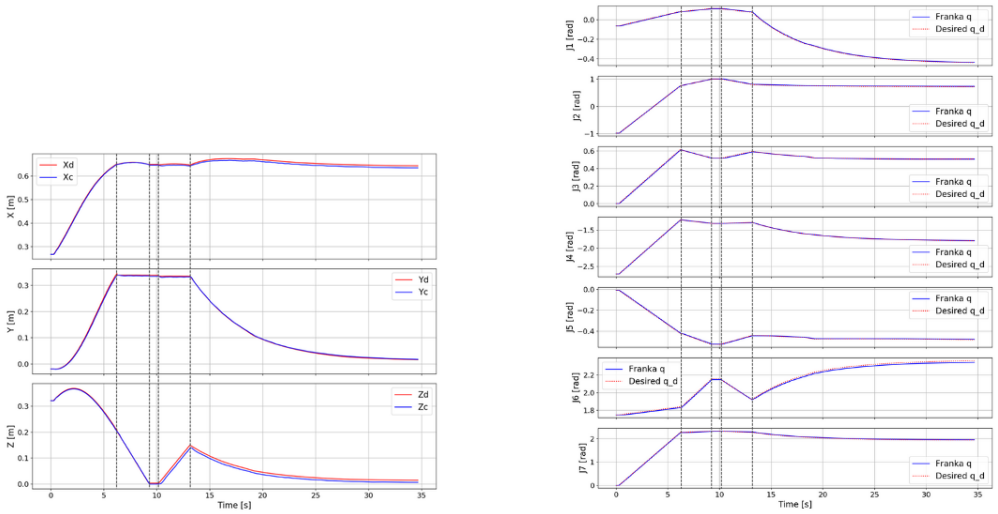
Experiment 3 corresponds to the configuration R2–H1 with the robot in pose A (Table 5.1). The robot is assigned to the horizontal profile, with a different starting position, while the operator holds the vertical one.

Figure 6.5 shows the different task phases captured from two viewpoints.

The end-effector (Figure 6.6(a)) trajectory has small deviations in the x and z axis post grasping. Regarding the joint position (Figure6.6(b)), a small error can be found during the ALIGNMENT state on joint 6, while the rest of the joints follow the desired trajectories with high precision. Joint velocities (Figure 6.6(c)) exhibit larger oscillations during the ALIGNMENT phase, again in joint 6. All joint velocities converge smoothly to zero after $t \approx 30\,\text{s}$, when the robot reaches the final pose relative to the human-held profile.

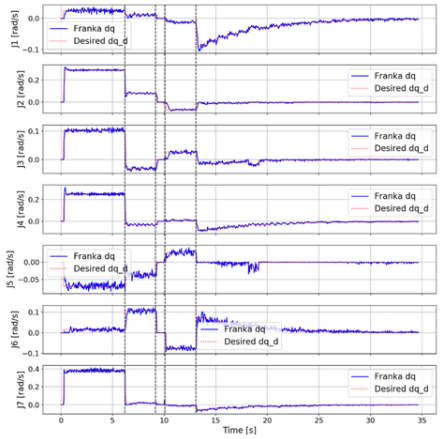
For Experiment 3 (Figure 6.6(d)), joint 2 again carries most of the load, reaching about 50 Nm during LIFT and ALIGNMENT, consistent with supporting the horizontal profile against gravity, while wrist joints (J5J7) show mild oscillations during ALIGNMENT. Overall, torque demands remain moderate and safely within actuator limits.

The external wrench (Figure 6.6(e)) shows an increase in F_x during ALIGNMENT and seems to reach -15N. Similarly, F_z remains constant to 2-3 N after LIFT.

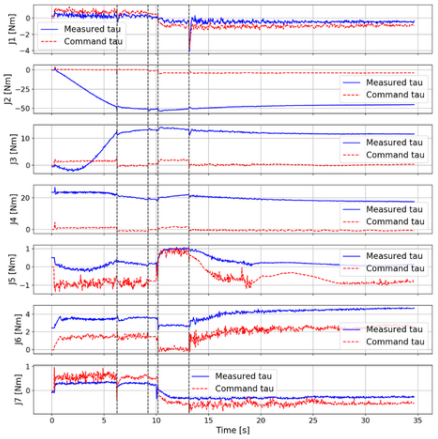


(a)

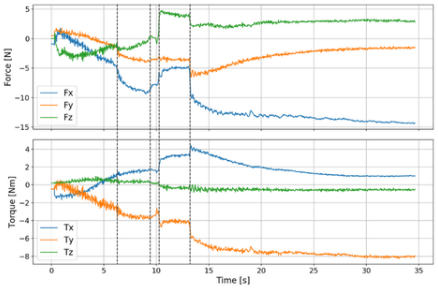
(b)



(c)

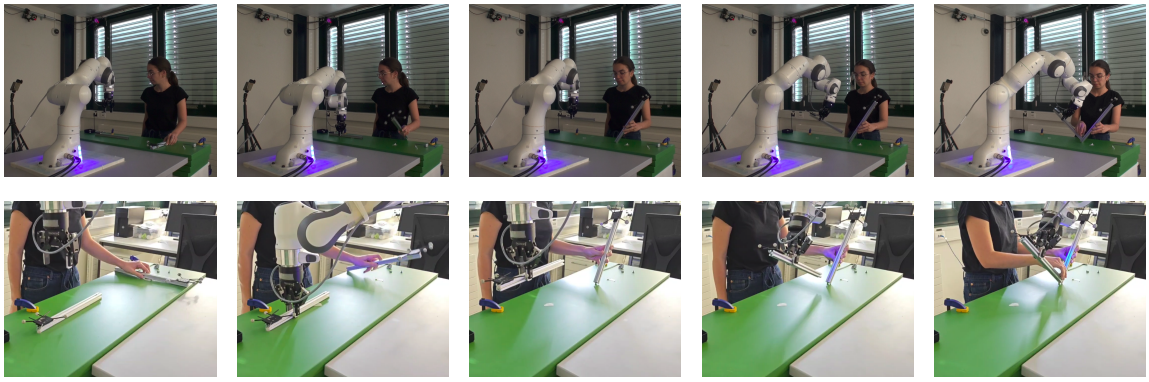


(d)



(e)

Figure 6.6: Experiment 3 results. (a) Desired vs. real end-effector positions. (b) Joint positions q vs. q_d. (c) Joint velocities $\dot{q}$ vs. $\dot{q}_d$. (d) Commanded vs. measured joint torques. (e) Estimated external wrench at the end-effector. Vertical dashed lines mark phase boundaries: APPROACH (0–6 s), PRE-GRASP (6–9 s), GRASP (9–10 s), LIFT (10–13 s), ALIGNMENT ($>$ 13 s).



(a) (b) (c) (d) (e)

Figure 6.7: Experiment 4 illustrated with photographs of the main stages: (a) APPROACH, (b) GRASP, (c) LIFT, (d) ALIGNMENT and (e) ASSEMBLY. For each stage, images from two viewpoints are shown.

Experiment 4 - R2-H2, pose A

Experiment 4 corresponds to the configuration R2–H2 with the robot still in pose A (Table 5.1). The robot is assigned to the horizontal profile, with the same starting position as Experiment 3, while the operator holds the vertical one, tilted like Experiment 2.

Figure 6.7 illustrates the task phases from the same two perspectives.

Small deviations can be seen on the end-effector (Figure 6.8(a)) trajectory in the x and z axis post grasping, and also in the y axis at the end of the ALIGNMENT state.

Regarding the joint positions (Figure6.8(b)), a small error is visible in joint 6, while the remaining joints track the desired trajectories with high accuracy. Higher oscillations in joint velocities (Figure 6.8(c)) can be seen in joints 3, 5 and 6 during the ALIGNMENT phase. All joint velocities converge smoothly to zero after $t \approx 30$ s, when the robot reaches the final pose.

Due to hand tremor, the commanded torque (Figure 6.8(d)), shows pronounced oscillations especially in the wrist joints.

The external wrench (Figure 6.8(e)) again shows an increase in F_x during ALIGNMENT, it reaches -15N and after a while it seems to converge to 0. The same can be noticed with F_z, that after lifting remains around to 2-3 N.

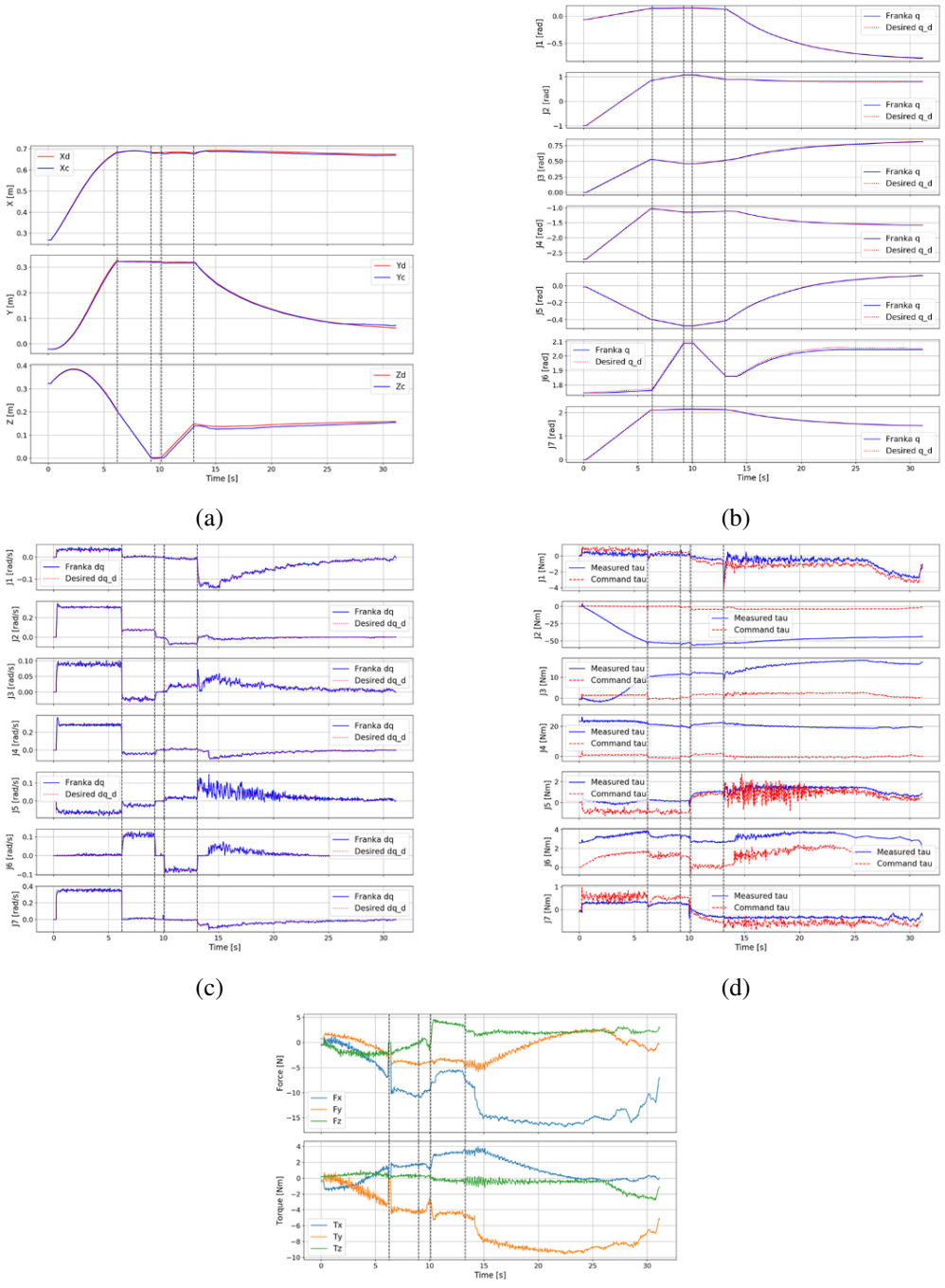
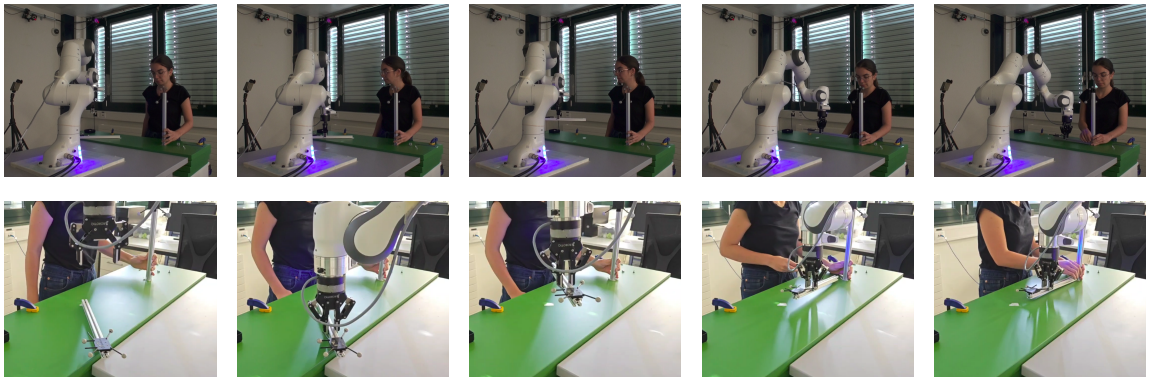


Figure 6.8: Experiment 4 results. (a) Desired vs. real end-effector positions. (b) Joint positions q vs. q_d. (c) Joint velocities $\dot{q}$ vs. $\dot{q}_d$. (d) Commanded vs. measured joint torques. (e) Estimated external wrench at the end-effector. Vertical dashed lines mark phase boundaries: APPROACH (0–6 s), PRE-GRASP (6–9 s), GRASP (9–10 s), LIFT (10–13 s), ALIGNMENT ($>$ 13 s).



(a) (b) (c) (d) (e)

Figure 6.9: Experiment 5 illustrated with photographs of the main stages: (a) APPROACH, (b) GRASP, (c) LIFT, (d) ALIGNMENT and (e) ASSEMBLY. For each stage, images from two viewpoints are shown.

Experiment 5 - R1-H1, pose B

Experiment 5 starts from the configuration R1–H1 of experiment 1 with the robot in pose B (Table 5.1). The robot is assigned to the horizontal profile, while the operator holds the vertical one.

Figure 6.9 illustrates the task phases from the same two perspectives.

Small deviations can be seen on the end-effector (Figure 6.10(a)) trajectory in the x and z axis during APPROACH, LIFT and ALIGNMENT states. Joint 6 and 7 show small errors in the joint positions (Figure6.10(b)), while the remaining joints still track the desired position with high precision. Higher oscillations in joint velocities (Figure 6.10(c)) can be seen in most of the joints, but they are still acceptable. The joint velocities converge to zero after $t \approx 30$ s.

Joint 5 commanded torque shows marked oscillations (Figure 6.10(d)).

The external wrench (Figure 6.10(e)) F_x reaches -2 N during ALIGNMENT and F_z after lifting (when it reaches 4 N) decreases below 0 N.

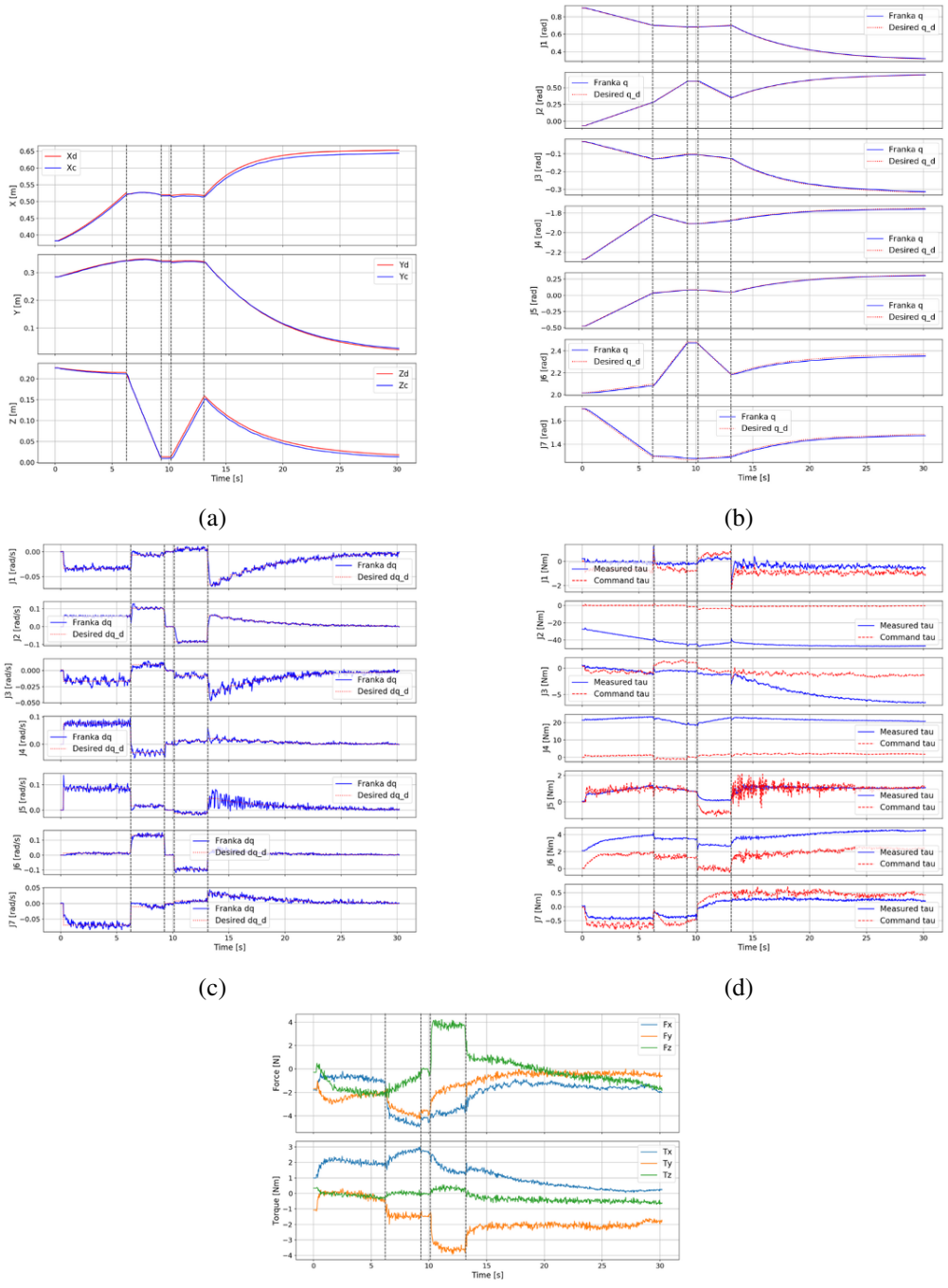
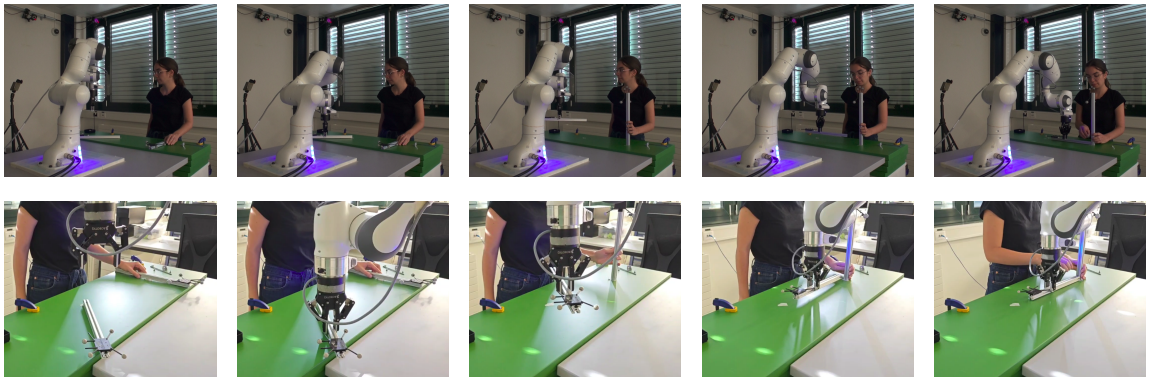


Figure 6.10: Experiment 5 results. (a) Desired vs. real end-effector positions. (b) Joint positions q vs. q_d. (c) Joint velocities $\dot{q}$ vs. $\dot{q}_d$. (d) Commanded vs. measured joint torques. (e) Estimated external wrench at the end-effector. Vertical dashed lines mark phase boundaries: APPROACH (0–6 s), PRE-GRASP (6–9 s), GRASP (9–10 s), LIFT (10–13 s), ALIGNMENT ($>$ 13 s).



(a) (b) (c) (d) (e)

Figure 6.11: Experiment 6 illustrated with photographs of the main stages: (a) APPROACH, (b) GRASP, (c) LIFT, (d) ALIGNMENT and (e) ASSEMBLY. For each stage, images from two viewpoints are shown.

Experiment 6 - R1-H1, pose C

Experiment 6 begins from the same configuration R1–H1 of experiment 1 and 5, but with the robot in pose C (Table 5.1).

Figure 6.11 shows the task phases observed from the two viewpoints.

Small deviations can be seen on the end-effector (Figure 6.12(a)) trajectory in the x and z axis during LIFT and ALIGNMENT states. The joint positions (Figure6.12(b)) track the desired position with high accuracy. Lower oscillations can be seen in joint velocities (Figure 6.12(c)) with respect to Experiment 5, track the desired velocities well. Torque oscillations (Figure 6.12(d)) also decrease, with joint 2 carrying the majority of gravity compensation.

The external wrench (Figure 6.12(e)) F_x reaches -7 N during ALIGNMENT and F_z after lifting (when it reaches 4 N) decreases toward -2.5 N.

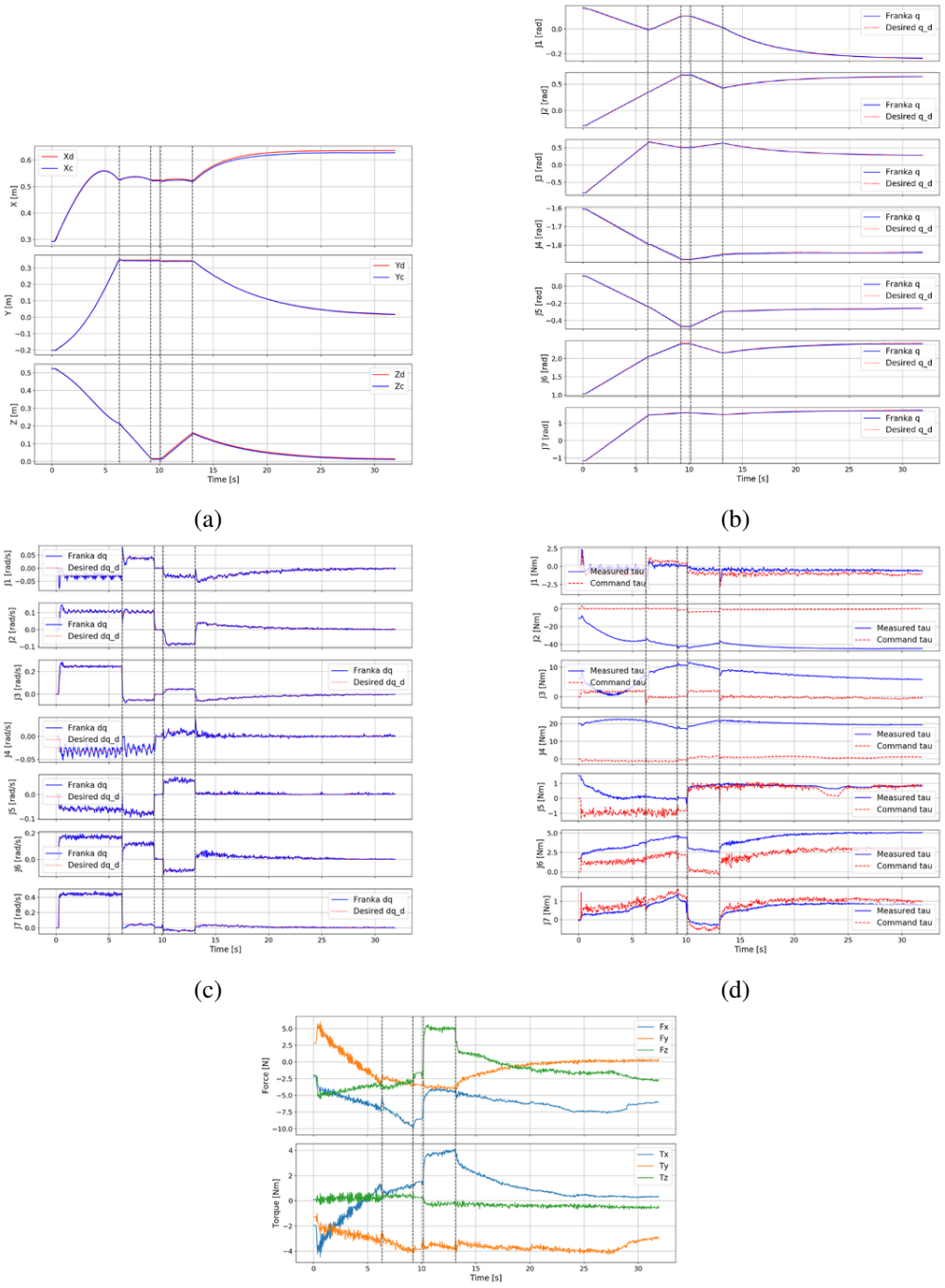
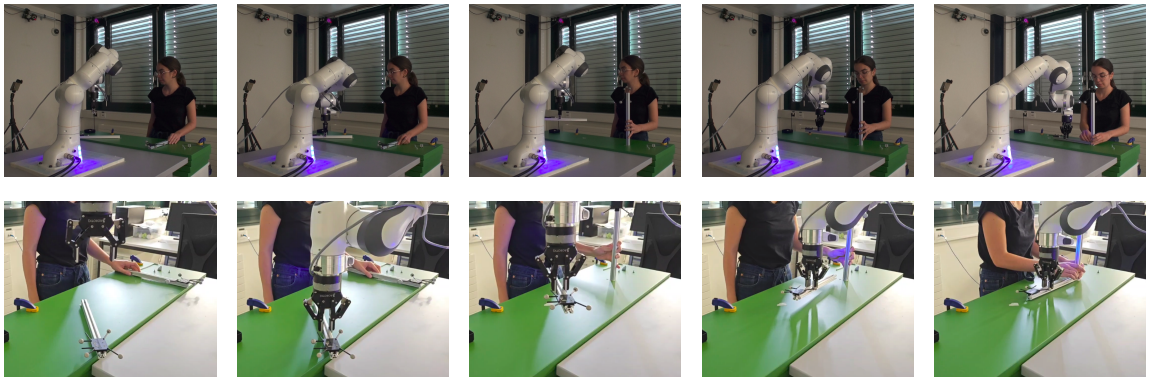


Figure 6.12: Experiment 6 results. (a) Desired vs. real end-effector positions. (b) Joint positions q vs. q_d. (c) Joint velocities $\dot{q}$ vs. $\dot{q}_d$. (d) Commanded vs. measured joint torques. (e) Estimated external wrench at the end-effector. Vertical dashed lines mark phase boundaries: APPROACH (0–6 s), PRE-GRASP (6–9 s), GRASP (9–10 s), LIFT (10–13 s), ALIGNMENT ($>$ 13 s).



(a) (b) (c) (d) (e)

Figure 6.13: Experiment 7 illustrated with photographs of the main stages: (a) APPROACH, (b) GRASP, (c) LIFT, (d) ALIGNMENT and (e) ASSEMBLY. For each stage, images from two viewpoints are shown.

Experiment 7 - R1-H1, pose D

Experiment 7 begins from the same configuration R1–H1 of experiment 1, 5 and 6, but with the robot in pose D (Table 5.1).

Figure 6.13 shows the task phases observed from the two viewpoints.

End-effector trajectory (Figure 6.14(a)) along the x axis shows small oscillations in the PRE-GRASP phase and higher error during LIFT and ALIGNMENT. Small deviations can also be seen during LIFT and ALIGNMENT of the z axis. Joint 2 and Joint 7 show large oscillations from the desired joint position(Figure6.14(b)). Oscillations are still present in the measured joint velocities (Figure 6.14(c)), but they are negligible. Also oscillations (Figure 6.14(d)) in the commanded torques decreased, with joint 2 having the majority of gravity compensation.

The external wrench (Figure 6.14(e)) show high oscillations in the APPROACH phase, probably because of the initial configuration of the robot. F_x is around -4 N during ALIGNMENT and F_z after lifting (when it reaches 4 N) decreases going toward -1 N.

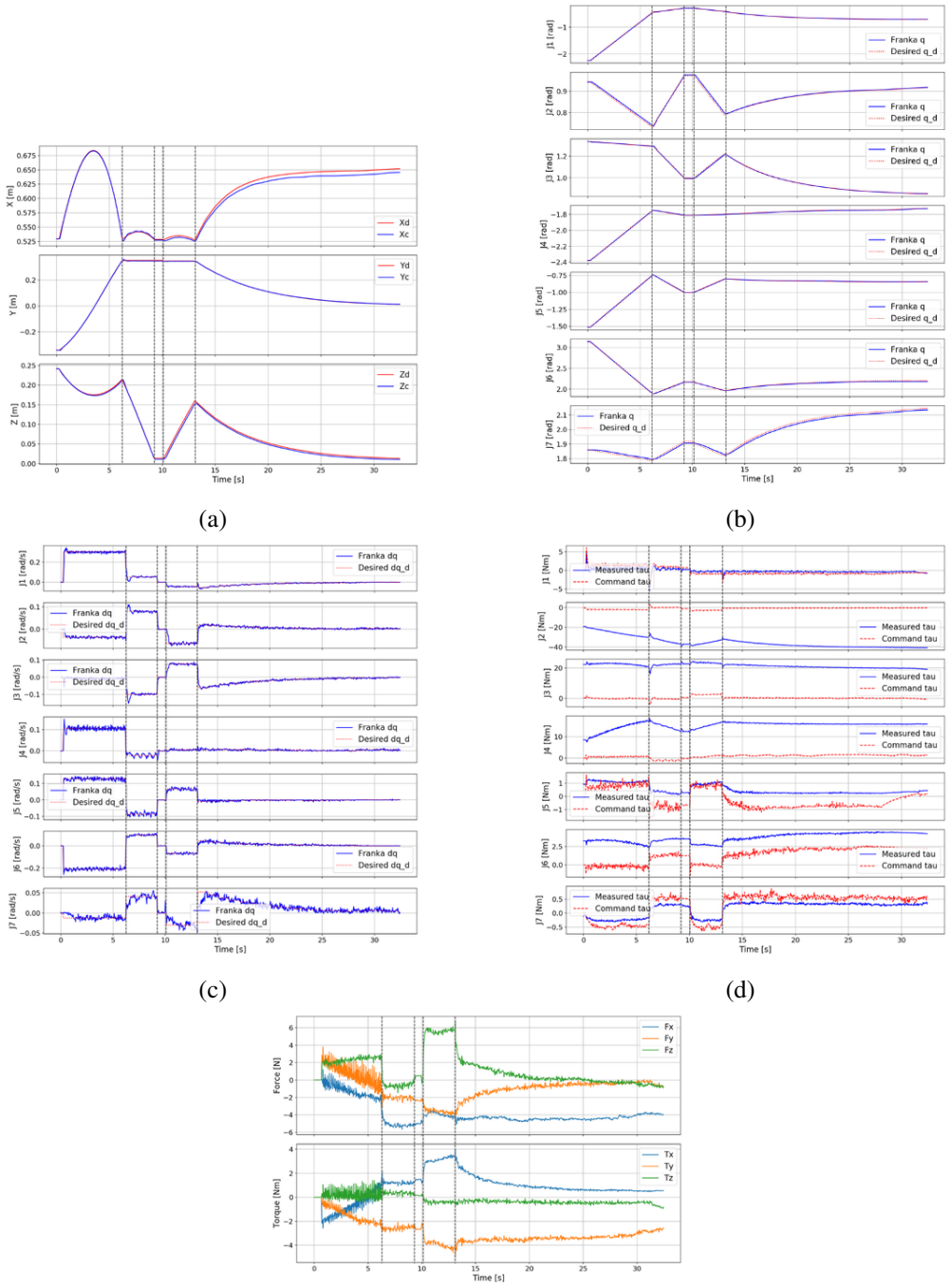


Figure 6.14: Experiment 7 results. (a) Desired vs. real end-effector positions. (b) Joint positions q vs. q_d. (c) Joint velocities $\dot{q}$ vs. $\dot{q}_d$. (d) Commanded vs. measured joint torques. (e) Estimated external wrench at the end-effector. Vertical dashed lines mark phase boundaries: APPROACH (0–6 s), PRE-GRASP (6–9 s), GRASP (9–10 s), LIFT (10–13 s), ALIGNMENT ($>$ 13 s).

6.2 Simulation evaluation

The same planner–controller stack (grasp optimization, constrained trajectory segments and joint-impedance control with torque-rate limits) was verified in simulation on Gazebo (Franka + Franka gripper model, rigid environment, perfect state estimation). The simulations covered both the postures tested on hardware and additional configurations that were not executed on the physical robot, including cases where the robot held the vertical profile. Under the same limits, both L- and T-shape assemblies completed successfully with the robot in that role. This outcome points to a posture related constraint in the hardware runs (proximity to a wrist singularity) rather than a limitation of the method itself. The simulation was used as an internal check and debugging aid and is not assessed further in this thesis.

6.3 Summary

Across the seven assemblies, execution remained consistent and within constraints. Before GRASP, the end-effector closely followed the commanded path in every run. The only exception was Experiment 7, where a small oscillation appeared on the x trace during PRE-GRASP without causing instability. After grasp detection and during LIFT, all trials showed a similar pattern: small, repeatable deviations on z as the arm took the load and moved toward the human-held profile, accompanied by only minor drift on x and tight tracking on y. During ALIGNMENT the behavior depends on the human-held profile and the starting arm posture. When the operator holds the vertical profile with a deliberate tilt (H2-R1), wrist motion increases and a more pronounced F_x appears once side contact forms, yet the contact stabilizes cleanly. In the upright cases (H1), posture matters most: with starting pose C the wrist stays away from near-singular configurations and the velocity/torque traces are smoother. Starting from poses B or D there is more wrist motion, but the outcome does not change. In all experiments the joint velocities drop to zero by the end of ALIGNMENT and the final relative pose is reached cleanly.

The measured forces and torques follow the same picture. At the start of ALIGNMENT F_x is near zero, rising only after lifting the object and then drops as the contact redistributes. The normal force F_z shows a small peak during LIFT in all experiments ($\sim$ 4–6 N) and then drops back once the LIFT height is reached. On the actuation side, J2 carries most of the gravity during GRASP and LIFT, while the other joints add smaller adjustments. Wrist activity depends on the condition: J5/J6 are more active in tilted trials and in starts that place the wrist nearer less favorable angles. Overall, torque levels remain moderate and within limits across all experiments.

Results in Table 6.1 line up with these qualitative observations. The duration of each experiment is measured from the first non-zero motion to the time at which all joint angular velocities drop below 0.005 rad/s, after applying a 7-sample moving average to joint velocities. This time corresponds to the robot-held profile reaching the human-held profile, which is kept still in the same position; minor residual motion at the end is attributed to natural hand tremor and is not task-relevant. Across the seven experiments, end-effector accuracy is stable: RMSE ranges from 6.57 mm (Exp. 2) to 10.31 mm (Exp. 3), with a mean of 8.4 mm. The mean joint-position RMSE is ≈ 0.0066 rad ($\approx 0.38^\circ$), with consistently higher values at J6–J7 in more demanding postures. Commanded torques are moderate overall: τ_{RMS} ranges from 10.5 to 12.3 Nm (overall

Table 6.1: Experiment metrics over the full trajectory.

Exp	Duration [s]	End-effector RMSE [m]	Joint position RMSE [rad]	τ_{RMS} mean [Nm]
1	30.440	0.0083	0.0062	10.50
2	36.719	0.0066	0.0056	10.68
3	34.640	0.0103	0.0071	11.64
4	31.126	0.0101	0.0074	12.32
5	30.168	0.0094	0.0062	10.59
6	31.874	0.0076	0.0077	10.62
7	32.428	0.0068	0.0060	11.12
Mean $\pm$ SD	**32.485 $\pm$ 2.393**	**0.0084 $\pm$ 0.0015**	**0.0066 $\pm$ 0.0008**	**11.07 $\pm$ 0.68**

mean $\approx$ 11.1 Nm). J2 carries most of the load (35–46 Nm) because it supports gravity during GRASP and LIFT; J3 is higher in Exp. 7 due to the initial posture. Exp. 2 and Exp. 7 show the best balance between accuracy and effort, whereas Exp. 4 requires the most torque and exhibits slightly larger errors. Differences in the joint-space traces during LIFT and ALIGNMENT for runs with the same task (e.g., R1–H1) are explained by redundancy[1] resolution: the 7-DoF arm legitimately adopts different joint configurations for the same Cartesian pose to stay within limits and away from near-singular postures.

[1] A 7-DOF manipulator is redundant because it has more joints than the six degrees of freedom needed to define the end-effector pose, allowing multiple joint configurations for the same task.

Chapter 7

Concluding Remarks

This thesis develops and evaluates a practical framework for human-robot assembly that integrates a GUI for task selection and role assignment with grasp optimization, motion-capture tracking of both profiles, constrained trajectory segments and a joint impedance controller with torque-rate limitation. The execution sequence consists of APPROACH, PRE-GRASP, GRASP, LIFT, optional REORIENTATION, ALIGNMENT and ASSEMBLY under periodic target updates, providing a smooth and constraint-compliant behavior.

On hardware, the L-shape assemblies with the robot holding the horizontal profile showed consistent execution across varied initial profile poses and arm postures. Tracking before contact was accurate, GRASP and LIFT completed reliably and ALIGNMENT proceeded smoothly without noticeable oscillations. Commanded torques remained within expected bounds. As anticipated, the shoulder joint (Joint 2) bore most of the load to counter gravity, while the contribution of the remaining joints depended on posture. Within the tested range, the starting posture did not consistently change tracking error or mean torque. Instead, it mainly redistributed effort among joints without degrading accuracy or total effort. During and after LIFT, small, repeatable deviations appeared primarily along z as the arm took the load and moved toward the human-held profile, while lateral drift on x stayed minor and tracking on y remained tight. In ALIGNMENT, the onset of side contact increased wrist activity and produced a temporary rise in F_x, which diminished as contact stabilized.

The execution traces are compatible with redundancy resolution in a 7-DoF arm: trials aimed at the same Cartesian goal sometimes followed different joint-space paths as joint limits and proximity to singularities allowed multiple feasible postures. In H2 cases, choosing initial configurations farther from near singular wrist poses yielded visibly calmer velocity and torque profiles while achieving the same final pose. Across experiments, velocities decayed to zero by the end of ALIGNMENT and the target pose was reached cleanly, indicating predictable behavior at contact.

A practical consideration concerns assigning the vertical profile to the robot. On hardware, a strictly vertical approach can drive the wrist close to a singularity and halt motion, whereas the same controller and limits succeed in simulation. This points to a posture-related constraint rather than a limitation of the method. A sensible direction for future work is to adopt a slightly tilted approach and posture-aware trajectory generation that penalizes low manipulability and steers away from wrist singularities.

Overall, the contribution is a compact, hardware-tested stack that behaves predictably with modest assumptions and minimal tuning. The pipeline runs from operator input on a GUI to grasp selection, pose tracking, bounded trajectory updates and compliant execution on a Franka Emika Panda with a Robotiq 2F-85. The method achieves contact-rich profile assembly with smooth execution, stable accuracy and consistent effort across the starting poses tested.

The framework has clear limits. During the contact phase, trajectories are refreshed at 1 Hz, which is adequate for steady interaction but reduces responsiveness: rapid changes by the operator are only incorporated at the next update, up to one second later, so visible lag and larger corrective moves can occur. Forces are only monitored, not exploited for guarded moves or active search. Feasible postures are sometimes restricted by proximity to wrist singularities, particularly in the robot-holds-vertical assignment. The operator is not perceived explicitly and safety is enforced through conservative bounds rather than a dedicated layer. Posture quality and force transmission are not analyzed along the sequence, and compliance is kept moderate to preserve tracking.

These limits suggest future work that builds on the current stack. Planning can prefer postures that keep the wrist away from near singular configurations and the loop can accept more frequent trajectory updates so small human operator changes are taken up promptly while updates remain bounded. Contact phases can be made more forgiving by increasing compliance where needed and by tightening the link between the tracked poses and the updates, so that small mismatches at the profiles do not lead to sharp corrections. A simple perception channel for the operator would allow the robot to keep distance and speed margins around the human and would provide a clearer basis for safety than conservative limits alone. Studying how posture affects manipulation and how forces are routed along the sequence would inform both goal choice and posture choice, adjusting the grasp coordinate online when margins shrink would help keep the task reachable as the hold position changes. Basic voice commands (pause, resume and small adjustments) could keep the operator hands on the workpiece and make the interaction smoother.

In summary, a compact, integrated stack is sufficient for this class of collaborative assembly. By coupling planning and control to signals that the operator can observe and influence, refreshing goals in short segments and executing with joint impedance control and torque rate limiting, the system runs smoothly, maintains stable accuracy and keeps effort moderate. The implementation serves as a practical baseline on real hardware and leaves clear room for incremental extensions.

Appendix A

ROS package

A.1 ROS custom package: `hri_assembly_task`

This appendix summarizes the structure and key files of the custom ROS package `hri_assembly_task`. The complete package is available at: https://github.com/carladevittorio/hri_assembly_task.git. In this appendix, only the most relevant files are shown.

Package Layout

The ROS package `hri_assembly_task` follows the standard ROS structure, extended with custom source code, configuration files and robot models. Below is a description of the purpose of each directory and the role of the files inside it.

Root Directory (hri_assembly_task) Contains the two core files that every ROS package needs:

- `package.xml`: Defines the package metadata (name, version, dependencies, maintainers).
- `CMakeLists.txt`: Build system instructions (CMake macros to compile nodes, generate messages and link libraries).
- `hri_assembly_task_plugin.xml`:Registers the shared library `libhri_assembly_task` and exposes `hri_assembly_task/Controller` (`hri_assembly_task::Controller`) as a `controller_interface::ControllerBase` plugin for position/impedance control.

launch This folder stores launch files, which automate the startup of multiple ROS nodes and parameters.

- `hri_assembly_task.launch`: it brings up the full system and switches between real hardware and Gazebo via the `use_real_robot` argument. It starts the OptiTrack interface and publishes a static TF from the `panda_base_OptiTrack` to `panda_link0`. For the real robot, it launches `franka_control` and generates `/robot_description` from the Panda+Robotiq xacro, while for simulation, it launches `franka_gazebo` and loads the

Gazebo-ready Panda description. It then loads runtime parameters from `config/hri_assembly_task.yaml`, sets key links (`base_link`, `ee_link`) and controller mode (`impedance`), spawns the controller via `controller_manager`, starts the package's task UI (`GUI.py`) and gripper node, and opens RViz with the provided config.

config Configuration files in YAML format that define parameters loaded at runtime.

- `hri_assembly_task.yaml`: Stores controller gains, planner options, or task-specific parameters.

msg Custom message definitions used for communication between nodes. ROS automatically generates header files for these.

- `Data.msg`: Defines numerical data for plots.
- `TaskInfo.msg`: Contains information about the assembly task published by the GUI and the status published by the controller .

src Source code implementing the package functionality. Each `.cpp` file corresponds to a ROS node or helper class:

- `controller.cpp`: Main control logic.
- `ImpedanceController.cpp`, `PositionController.cpp`: Different control strategies.
- `KinematicsSolver.cpp`: Handles forward/inverse kinematics using Pinocchio library.
- `RuckigTrajectory.cpp`: Implements trajectory generation using Ruckig library.
- `gripper_node.cpp`: Node to control the Robotiq gripper.
- `logger.cpp`: Data logging functionality.
- `solver_qpOASES.cpp`: Optimization solver interface for control tasks.

include Header files corresponding to the source code in `src/`. This separation follows the standard C++ practice and makes the code reusable across nodes. Files include class definitions for controllers, solvers and utilities.

robots URDF models *and* meshes for the Franka Panda arm and the Robotiq 2F-85 gripper. These assets define geometry, kinematics, dynamics and visuals used across the package (e.g., for `/robot_description`, RViz and simulation).

worlds Gazebo world files for simulation scenarios used by the launch files.

scripts Python executables (e.g., GUI and Robotiq helpers) launched as ROS nodes.

rviz Preconfigured RViz visualization files for the assembly task (fixed frames, displays and topics).

```
hri_assembly_task/
├── package.xml
├── CMakeLists.txt
├── hri_assembly_task_plugin.xml
├── launch/
│   └── hri_assembly_task.launch
├── config/
│   └── hri_assembly_task.yaml
├── msg/
│   ├── Data.msg
│   └── TaskInfo.msg
├── src/
│   ├── controller.cpp
│   ├── ImpedanceController.cpp
│   ├── PositionController.cpp
│   ├── KinematicsSolver.cpp
│   ├── RuckigTrajectory.cpp
│   ├── gripper_node.cpp
│   ├── logger.cpp
│   └── solver_qpOASES.cpp
├── include/
│   ├── controller.h
│   ├── ImpedanceController.h
│   ├── PositionController.h
│   ├── KinematicsSolver.h
│   ├── RuckigTrajectory.h
│   ├── gripper_node.h
│   ├── logger.h
│   └── solver_qpOASES.h
├── robots/
│   ├── panda.urdf
│   └── robotiq_2f85.urdf
├── worlds/
│   └── world_with_object.world
├── scripts/
│   ├── GUI.py
│   └── Robotiq2F85Driver.py
└── rviz/
    └── hri_assembly_task
```

Controller State Machine Implementation

The following code shows the state machine logic of the `Controller` class, responsible for handling the robot's task execution:

Listing A.1: Approach state in `Controller`

```
1     void Controller::approachState(const ros::Duration& period) {
2     publishState();
3     if (!approach_initialized) {
4         q_d_i = arm_q;
5         kinematic_solver_.set_q(q_d_i);
6         pinocchio::SE3 pose_grasp_pick = kinematic_solver_.
              compute_forward_kinematics(kinematic_solver_.get_frame_id
              ());
7         initPickObject(robot_profile, pose_grasp_pick, x_g, q_d_f,
              arm_q, kinematic_solver_, sign_xee_xobj, task_assignment);
8         x_g = positionGraspQP(robot_profile.x_length, task_assignment
              );
9
10        if (q_d_f.size() == 0) {
11          previous_state_ = current_state_;
12          current_state_ = State::Error;
13          return;
14        }
15
16        if (!trajectory_.init_offline(q_d_i, q_d_f, dq_d_i, dq_d_f,
              ddq_d_i, ddq_d_f, 6)) {
17            ROS_ERROR("Trajectory init failed     error state");
18            current_state_ = State::Error;
19            return;
20        }
21        approach_initialized = true;
22        elapsed_time = ros::Duration(0.0);
23    } else {
24        if (updateOfflineTrajectory(period)) {
25            current_state_ = State::preGraspState;
26        }
27    }
28 }
```

Listing A.2: Pre-grasp state in `Controller`

```
1
2 void Controller::preGraspState(const ros::Duration& period) {
3   publishState();
4   if (!pre_grasp_initialized) {
5     q_d_i = arm_q;
6     Eigen::Vector3d position_d_f(0.0,0.0,-0.20);
7     Eigen::Quaterniond quat_d_f;
```

```
8       q_d_f = fromCartesianToJointFinalPosition(q_d_i, "relative",
            position_d_f, "fixed", quat_d_f);
9
10      if (!trajectory_.init_offline(q_d_i, q_d_f, dq_d_i, dq_d_f,
            ddq_d_i, ddq_d_f, 3)) {
11        ROS_ERROR("Trajectory init failed     error state");
12        current_state_ = State::Error;
13        return;
14      }
15
16      pre_grasp_initialized = true;
17      elapsed_time = ros::Duration(0.0);
18    } else {
19      if (updateOfflineTrajectory(period)) {
20        lift_initialized = false;
21        current_state_ = State::Grasp;
22      }
23    }
24  }
```

Listing A.3: Grasp state in Controller

```
1  void Controller::grasp() {
2    publishState();
3    gripper_cmd = "grasp";
4    if (gripper_feedback=="grasp_reached") {
5      current_state_ = State::liftProfile;
6    }
7  }
```

Listing A.4: Lift state in Controller

```
1  void Controller::liftState(const ros::Duration& period) {
2    publishState();
3    if (!lift_initialized) {
4      q_d_i = arm_q;
5      Eigen::Vector3d position_d_f(0.0, 0.0, 0.15);
6      Eigen::Quaterniond quat_d_f;
7      q_d_f = fromCartesianToJointFinalPosition(q_d_i, "relative",
           position_d_f, "fixed", quat_d_f);
8
9      if (!trajectory_.init_offline(q_d_i, q_d_f, dq_d_i, dq_d_f,
           ddq_d_i, ddq_d_f, 3)) {
10       ROS_ERROR("Trajectory init failed     error state");
11       current_state_ = State::Error;
12       return;
13     }
14
15     lift_initialized = true;
```

```
16      elapsed_time = ros::Duration(0.0);
17    } else {
18      if (updateOfflineTrajectory(period)) {
19        if (task_assignment == "HumV_RobH") {
20          current_state_ = State::alignmentState;
21        } else if (task_assignment == "HumH_RobV") {
22          current_state_ = State::reorientationState;
23        }
24      }
25    }
26  }
```

Listing A.5: Reorientation state in Controller

```
1   void Controller::reorientation(const ros::Duration& period) {
2     publishState();
3     if (!vertical_initialized) {
4       q_d_i = arm_q;
5
6       kinematic_solver_.set_q(q_d_i);
7       pinocchio::SE3 pose_grasp_lift = kinematic_solver_.
            compute_forward_kinematics(kinematic_solver_.get_frame_id());
8
9       initVerticalObject(robot_profile, pose_grasp_lift, q_d_f, arm_q,
            kinematic_solver_, sign_xee_xobj, main_hand_human, table_side)
            ;
10
11      if (!trajectory_.init_offline(q_d_i, q_d_f, dq_d_i, dq_d_f,
            ddq_d_i, ddq_d_f, 7)) {
12        ROS_ERROR("Trajectory init failed     error state");
13        current_state_ = State::Error;
14        return;
15      }
16      vertical_initialized = true;
17      elapsed_time = ros::Duration(0.0);
18    } else {
19      if (updateOfflineTrajectory(period)) {
20        current_state_ = State::alignment;
21      }
22    }
23  }
```

Listing A.6: Alignment state in Controller

```
1   void Controller::alignment(const ros::Duration& period) {
2     publishState();
3     ++retarget_elapsed;
4     if (!alignment_initialized || retarget_elapsed >=
          retarget_time_ticks ) {
```

```
5       q_d_i = q_d;
6       dq_d_i = dq_d;
7       ddq_d_i = ddq_d;
8
9       kinematic_solver_.set_q(arm_q);
10      pinocchio::SE3 pose_grasp_vertical = kinematic_solver_.
            compute_forward_kinematics(kinematic_solver_.get_frame_id());
11      initAlignObject(robot_profile, human_profile, pose_grasp_vertical
            , q_d_f, arm_q, kinematic_solver_, x_g, main_hand_human,
            table_side, T_percentage, task_assignment);
12      if (q_d_f.size() == 0) {
13        last_human_position = current_human_pos;
14        previous_state_ = current_state_;
15        current_state_ = State::Error;
16        return;
17      }
18
19      if (!trajectory_.init_offline(q_d_i, q_d_f, dq_d_i, dq_d_f,
            ddq_d_i, ddq_d_f, 6)) {
20        ROS_ERROR("Trajectory init failed     error state");
21        current_state_ = State::Error;
22        return;
23      }
24      alignment_initialized = true;
25      elapsed_time = ros::Duration(0.0);
26      retarget_elapsed = 0;
27    } else {
28       if (updateOfflineTrajectory(period)) {
29        current_state_ = State::Done;
30      }
31    }
32  }
```

Listing A.7: PositionGraspQP in SubtaskExecutor.cpp

```
1   double positionGrasp(double& x_tot, std::string& task_assignment) {
2     double delta = 0.02;
3     double x_min = -x_tot/2 + 0.07;
4     double x_max = x_tot/2 - 0.08;
5     double x_g;
6
7     if (task_assignment == "HumV_RobH") {
8       x_g = x_min + delta;
9     } else if (task_assignment == "HumH_RobV") {
10      x_g = x_max - delta;
11    }
12
13    Eigen::MatrixXd H(1,1);
14    Eigen::VectorXd g(1);
```

```
15    Eigen::MatrixXd A(0,1);
16    Eigen::VectorXd lb(1), ub(1);
17    Eigen::VectorXd lbA(0), ubA(0);
18
19    H << 2.0;
20    g << -2.0*x_g;
21    lb << x_min;
22    ub << x_max;
23
24    if (!qpOASES_solver_.solve(H, g, A, lb, ub, lbA, ubA)) {
25      ROS_ERROR("QP setup failed");
26    }
27    return qpOASES_solver_.get_optimal_solution()(0);
28  }
```

Listing A.8: InitPickObject in SubtaskExecutor.cpp

```
1   void initPickObject(Profile& robot_profile, pinocchio::SE3&
      pose_grasp, double& z_g, Eigen::VectorXd& q_d_f, Eigen::VectorXd&
      arm_q, KinematicsSolver& kinematic_solver_, int sign_xee_xobj, std
      ::string& task_assignment) {
2     Eigen::Vector3d p_r = robot_profile.position;
3     Eigen::Matrix3d R_r = robot_profile.orientation.toRotationMatrix();
4
5     Eigen::Matrix3d R_ee_start = pose_grasp.rotation();
6
7     Eigen::Vector3d z_world = Eigen::Vector3d::UnitZ();
8     Eigen::Vector3d z_ee = -(z_world).normalized();
9     Eigen::Vector3d x_r = R_obj.col(0).normalized();
10
11    sign_xee_xobj = checkAngleRad(R_grasp.col(0), x_obj, z_world);
12
13    Eigen::Vector3d x_ee;
14
15
16    if (sign_xee_xobj > 0) {
17        x_ee = x_r;
18    } else {
19        x_ee = -x_r;
20    }
21
22    Eigen::Vector3d y_ee = z_ee.cross(x_ee);
23    x_ee.normalized();
24
25    Eigen::Matrix3d R_approach;
26    R_approach.col(0) = x_ee;
27    R_approach.col(1) = y_ee;
28    R_approach.col(2) = z_ee;
29
```

```
30    Eigen::Vector3d local(z_g,0,0);
31    Eigen::Vector3d p_approach = p_r + R_r*local;
32    Eigen::Vector3d  z_offset_over_pick(0.0, 0.0, 0.2);
33    Eigen::Vector3d offset_zee_w = (gripper_z_offset(2)) * z_ee;
34
35
36    pinocchio::SE3 pose_d_f(R_approach, p_approach-offset_zee_w+
          z_offset_over_pick);
37
38    kinematic_solver_.set_q(arm_q);
39    q_d_f = kinematic_solver_.compute_inverse_kinematics(
40        kinematic_solver_.get_frame_id(),
41        pinocchio::ReferenceFrame::LOCAL,
42        "pseudoinverse",
43        pose_d_f);
44
45    if (q_d_f.size() == 0) {
46        ROS_WARN("IK failed for pose");
47    }
48  }
```

Listing A.9: InitVerticalObject in SubtaskExectur.cpp

```
1
2   void initVerticalObject(Profile& robot_profile, pinocchio::SE3&
        pose_ee_start, Eigen::VectorXd& q_d_f, Eigen::VectorXd& arm_q,
        KinematicsSolver& kinematic_solver_, int& sign_xee_xobj, std::
        string& hand, std::string& side) {
3     Eigen::Matrix3d R_ee_start = pose_ee_start.rotation();
4     Eigen::Vector3d p_ee_start = pose_ee_start.translation();
5
6     Eigen::Vector3d z_world = Eigen::Vector3d::UnitZ();
7     Eigen::Matrix3d R_r = robot_profile.orientation.toRotationMatrix();
8
9     sign_xee_xobj = checkAngleRad(R_ee_start.col(0), R_r.col(0),
          z_world);
10
11    Eigen::Vector3d x_ee;
12    if (sign_xee_xobj > 0) {
13        x_ee = -z_world;
14    } else {
15        x_ee = z_world;
16    }
17
18    Eigen::Vector3d z_ee;
19    if (hand=="right") {
20      if (side=="front") {
21        z_ee = -Eigen::Vector3d::UnitY();
22      } else if (side=="side") {
```

```
23        z_ee = Eigen::Vector3d::UnitY();
24      } else if (side=="angle") {
25        z_ee = Eigen::Vector3d::UnitX();
26      }
27    } else if (hand=="left") {
28        z_ee = Eigen::Vector3d::UnitX();
29    }
30
31    x_ee.normalize();
32    z_ee.normalize();
33    Eigen::Vector3d y_ee = z_ee.cross(x_ee).normalized();
34    Eigen::Matrix3d R_vertical;
35    R_vertical.col(0) = x_ee;
36    R_vertical.col(1) = y_ee;
37    R_vertical.col(2) = z_ee;
38
39    pinocchio::SE3 pose_d_f(R_vertical, p_ee_start);
40
41    kinematic_solver_.set_q(arm_q);
42    q_d_f = kinematic_solver_.compute_inverse_kinematics(
43      kinematic_solver_.get_frame_id(),
44      pinocchio::ReferenceFrame::LOCAL,
45      "pseudoinverse",
46      pose_d_f);
47
48    if (q_d_f.size() == 0) {
49        ROS_WARN("IK failed for pose");
50    }
51  }
```

Listing A.10: initAlignObject in SubtaskExecutor.cpp

```
1  void initAlignObject(Profile& robot_profile, Profile& human_profile,
       pinocchio::SE3& pose_ee_start, Eigen::VectorXd& q_d_f, Eigen::
       VectorXd& arm_q, KinematicsSolver& kinematic_solver_, double& x_g,
        std::string& hand, std::string& side, double& T_percentage, std::
       string& task_assignment) {
2     bool success = false;
3     std::vector<std::string> approaches = {"from_x_obj", "from_y_obj"};
4
5     for (const auto& approach : approaches) {
6
7         Eigen::Matrix3d R_ee_start = pose_grasp.rotation();
8         Eigen::Vector3d p_ee_start = pose_grasp.translation();
9
10        Eigen::Vector3d p_r = robot_profile.position;
11        Eigen::Matrix3d R_r = robot_profile.orientation.
              toRotationMatrix();
12        Eigen::Vector3d p_h = human_profile.position;
```

```
13        Eigen::Matrix3d R_h = human_profile.orientation.
              toRotationMatrix();
14        Eigen::Vector3d x_ee;
15        Eigen::Vector3d z_ee;
16        Eigen::Vector3d y_ee;
17        Eigen::Vector3d p_align;
18
19        if (task_assignment == "HumV_RobH") {
20          x_ee = -R_h.col(2);
21          z_ee = R_h.col(0);
22          x_ee.normalize();
23          z_ee.normalize();
24          y_ee = z_ee.cross(x_ee).normalized();
25
26          Eigen::Vector3d z_world = Eigen::Vector3d::UnitZ();
27          Eigen::Vector3d offset_zh_w = (robot_profile.x_length/2-x_g+
                human_profile.y_length/2) * R_h.col(2);
28          Eigen::Vector3d offset_yh_w =R_h.col(1);
29
30          Eigen::Vector3d offset_xh_w = (human_profile.x_length/2 -
                robot_profile.y_length/2) * R_h.col(0);
31
32          Eigen::Vector3d offset_zee_w = (gripper_z_offset(2)) * z_ee;
33
34          p_align = p_h+offset_zh_w+offset_xh_w-offset_zee_w+
                offset_yh_w;
35
36        } else if (task_assignment == "HumH_RobV") {
37          int sign_xee_z_h = checkAngleRad(R_ee_start.col(0), R_h.col
                (2), R_h.col(1));
38          if (sign_xee_z_h > 0) {
39              x_ee = R_h.col(2);
40          } else {
41              x_ee = -R_h.col(2);
42          }
43
44          if (approach=="from_y_obj") {
45            if (hand=="left") {
46              z_ee = -R_h.col(1);
47            } else {
48              z_ee = R_h.col(1);
49            }
50
51          } else if (approach=="from_x_obj")  {
52            z_ee = -R_h.col(0);
53          }
54
55          x_ee.normalize();
56          z_ee.normalize();
```

```
57
58          y_ee = z_ee.cross(x_ee).normalized();
59
60          Eigen::Vector3d z_world = Eigen::Vector3d::UnitZ();
61          Eigen::Vector3d offset_zh_w = (x_g+human_profile.y_length/2)
               * R_h.col(2);
62          double start = human_profile.x_length / 2.0 - 0.01;
63          double offset_value = start * (1.0 - T_percentage / 100.0);
64
65          Eigen::Vector3d offset_xh_w = offset_value * R_h.col(0);
66          Eigen::Vector3d offset_zee_w = (gripper_z_offset(2)) * z_ee;
67
68          p_align = p_h+offset_zh_w+offset_xh_w-offset_zee_w;
69
70        }
71
72        Eigen::Matrix3d R_align;
73        R_align.col(0) = x_ee;
74        R_align.col(1) = y_ee;
75        R_align.col(2) = z_ee;
76
77        pinocchio::SE3 pose_d_f(R_align, p_align;
78
79        q_d_f = kinematic_solver_.compute_inverse_kinematics(
80            kinematic_solver_.get_frame_id(),
81            pinocchio::ReferenceFrame::LOCAL,
82            "pseudoinverse",
83            pose_d_f);
84
85        if (q_d_f.size() > 0) {
86            success = true;
87        }
88    }
89 }
```

Appendix B

Robotics

This appendix summarizes the main mathematical tools used throughout this work to describe and analyze robotic manipulators. The focus is on the kinematic modeling of serial robots, starting from the representation of positions and orientations (rotation matrices, homogeneous transformations and quaternions) and proceeding to the formulation of forward and inverse kinematics. Differential kinematics is then introduced through the manipulator Jacobian, which also provides the basis for the analysis of singular configurations.

These concepts are classical in robotics and follow the notation and derivations in *Robotics: Modelling, Planning and Control* by Siciliano et al. [35]. The goal of this appendix is to provide a compact yet comprehensive reference for the theoretical background that supports the implementation of the `hri_assembly_task` package, including trajectory generation, inverse kinematics solvers and control strategies.

B.1 Pose of a rigid body

The pose of a rigid body in space is defined by its *position* and *orientation*. The position is described by a vector $\mathbf{p} \in \mathbb{R}^3$, while the orientation can be represented in several mathematically equivalent ways, each with different advantages. The most common representations are rotation matrices and quaternions. The pose is represented by a homogeneous transformation matrix, which combines position and orientation in a single object.

Homogeneous Transformation Matrix

The pose of a body frame $\{B\}$ with respect to a reference frame $\{A\}$ is

$$\mathbf{T}_B^A = \begin{bmatrix} \mathbf{R}_B^A & \mathbf{p}_B^A \\ 0^\top & 1 \end{bmatrix} \in SE(3),$$

where $\mathbf{R}_B^A \in SO(3)$ is the rotation matrix describing the orientation of $\{B\}$ relative to $\{A\}$ and $\mathbf{p}_B^A \in \mathbb{R}^3$ is the position of the origin of $\{B\}$ expressed in $\{A\}$. This matrix compactly describes the rigid-body motion and compositions of motions are obtained by matrix multiplication.

Rotation Matrices

The orientation part of the pose, $\mathbf{R} \in SO(3)$, is a 3×3 orthogonal matrix expressed as three column vectors,

$$\mathbf{R} = \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{bmatrix},$$

where $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^3$ are the unit vectors of the rotated frame's axes, expressed in the reference frame.

These vectors satisfy the orthonormality conditions:

$$\mathbf{x}^\top \mathbf{y} = \mathbf{y}^\top \mathbf{z} = \mathbf{z}^\top \mathbf{x} = 0, \qquad \|\mathbf{x}\| = \|\mathbf{y}\| = \|\mathbf{z}\| = 1.$$

The rotation matrix has determinant $+1$:

$$\mathbf{R}^\top \mathbf{R} = I, \quad \det \mathbf{R} = 1.$$

It transforms coordinates of vectors between frames.

Quaternions

An alternative to rotation matrices is the unit quaternion

$$Q = \{\eta, \varepsilon\}, \quad \eta \in \mathbb{R},\ \varepsilon \in \mathbb{R}^3,$$

with $\eta = \cos(\frac{\theta}{2})$, $\varepsilon = \sin(\frac{\theta}{2})\, r$, where r is the unit rotation axis and θ the rotation angle. η is called the *scalar part* of the quaternion, while $\varepsilon = [\varepsilon_x\ \varepsilon_y\ \varepsilon_z]^\top$ is the *vector part*. For a unit quaternion they are constrained by

$$\eta^2 + \varepsilon_x^2 + \varepsilon_y^2 + \varepsilon_z^2 = 1$$

B.2 Forward Kinematics

Forward kinematics describes how the configuration of the robot joints uniquely determines the position and orientation (pose) of the end-effector with respect to a reference frame. The problem consists in computing the mapping from the joint vector $\mathbf{q}$ to the homogeneous transformation $\mathbf{T}_e^b(\mathbf{q})$:

$$\mathbf{T}_e^b(\mathbf{q}) = \begin{bmatrix} \mathbf{R}_e^b(\mathbf{q}) & \mathbf{p}_e^b(\mathbf{q}) \\ \mathbf{0}^T & 1 \end{bmatrix} \in SE(3),$$

where $\mathbf{R}_e^b(\mathbf{q}) \in SO(3)$ represents the orientation of the end-effector and $\mathbf{p}_e^b(\mathbf{q}) \in \mathbb{R}^3$ represents its position.

Implementation in `hri_assembly_task`

In this work, forward kinematics is computed using the Pinocchio library, which provides efficient algorithms for rigid-body kinematics. Given a joint configuration q, the forward kinematics is computed as:

Listing B.1: Forward kinematics computation using Pinocchio

```
kinematic_solver_.set_q(q);
pinocchio::SE3 end_effector_pose =
    kinematic_solver_.compute_forward_kinematics(
        kinematic_solver_.get_frame_id());
```

The resulting `end_effector_pose` contains both the position and orientation of the end-effector. This information is used in trajectory planning and visualization.

B.3 Inverse Kinematics

Inverse kinematics (IK) determines the joint variables q that realize a desired end–effector pose x. In contrast to direct kinematics $\mathbf{x} = k(\mathbf{q})$, IK solves $k(\mathbf{q}) = \mathbf{x}$ and is central to specifying motions in Cartesian space. Solutions may be multiple, infinite (with redundancy) or nonexistent (outside the workspace), and closed-form solutions are not always available.

The direct kinematics of a manipulator is expressed as

$$\mathbf{x} = k(\mathbf{q}),$$

where $\mathbf{q} \in \mathbb{R}^n$ are the joint variables and $\mathbf{x} \in SE(3)$ the end–effector pose. The IK problem requires solving

$$k(q) = x_d.$$

When a closed–form solution is not available, differential IK is used. Defining the task–space error $\mathbf{e} = \mathbf{x_d} - \mathbf{x}$, the Jacobian $\mathbf{J}(\mathbf{q})$ relates velocities as

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}.$$

A common solution is based on the pseudo–inverse:

$$\dot{\mathbf{q}} = \mathbf{J}^{\dagger}(\mathbf{q})\,\dot{\mathbf{x}}_{\mathbf{d}},$$

possibly with damping to handle singularities.

Implementation in `hri_assembly_task`

Inverse kinematics is solved iteratively with a damped least–squares (DLS) scheme. At each iteration: (i) the current end–effector pose via FK is computed, (ii) a pose error is formed, (iii) the local frame Jacobian is evaluated J, (iv) compute $\mathbf{v} = -\mathbf{J}^{\top}(\mathbf{J}\mathbf{J}^{\top} + \lambda\mathbf{I})^{-1}\mathbf{e}$ and (v) integrate $q_{k+1} = \mathrm{clamp}\big(q_k + v\Delta t\big)$. with joint–limit clamping.

Listing B.2: Iterative Jacobian-based IK with Pinocchio (DLS)

```
Eigen::VectorXd KinematicsSolver::compute_inverse_kinematics(const
    pinocchio::Model::FrameIndex frame_id, const pinocchio::
    ReferenceFrame frame_reference, const std::string computation_mode
    , const pinocchio::SE3 pose_target)
{
```

```
3       bool success = false;
4       for (int i = 0; i < ik_max_iteration_; i++)
5       {
6           pinocchio::SE3 pose = compute_forward_kinematics(frame_id);
7           pinocchio::SE3 error_SE3 = pose_target.actInv(pose);
8           Eigen::VectorXd error = pinocchio::log6(error_SE3).toVector()
                ;
9
10          if (error.norm() < ik_tolerance_)
11          {
12              success = true;
13              break;
14          }
15          if (i >= ik_max_iteration_ - 1)
16          {
17              std::cout << ">> Max Iteration Reached (IK Solver)!" <<
                    std::endl;
18              break;
19          }
20
21          Eigen::MatrixXd J, JJT, damped;
22          Eigen::VectorXd v;
23
24          J.setZero();
25          JJT.setZero();
26          damped.setZero();
27          v.setZero();
28          J = compute_frame_Jacobian(frame_id, frame_reference);
29          Eigen::JacobiSVD<Eigen::MatrixXd> svd(J);
30
31          switch (get_ik_computation_mode(computation_mode))
32          {
33          case ComputationMode::PSEUDOINVERSE:
34              JJT = J * J.transpose();
35              damped = JJT + ik_damping_factor_ * Eigen::MatrixXd::
                    Identity(J.rows(), J.rows());
36              v = -J.transpose() * damped.ldlt().solve(error);
37              break;
38          case ComputationMode::COD:
39              v = -ik_alpha_ * J.completeOrthogonalDecomposition().
                    solve(error);
40              break;
41          default:
42              break;
43          }
44
45          Eigen::VectorXd q_new = integrate(model_, q_, v *
                ik_time_period_);
46          q_ = clamp_in_range_q(q_new, model_);
```

```
47      }
48
49      if (!success)
50      {
51          return Eigen::VectorXd();
52      }
53      return q_;
54  }
```

Bibliography

[1] Przemyslaw A. Lasota, Terrence Song, and Julie A. Shah. *A Survey of Methods for Safe Human-Robot Interaction*. 2017.

[2] Ales Vysocky and Petr Novak. Human - robot collaboration in industry. *MM Science Journal*, 2016:903–906, 06 2016.

[3] Abdelfetah Hentout, Mustapha Aouache, Abderraouf Maoudj, and Isma Akli. Human–robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017. *Advanced Robotics*, 33(15-16):764–799, 2019.

[4] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248–266, 2018.

[5] Jörg Krüger, Terje Kristoffer Lien, and Alexander W. Verl. Cooperation of human and machines in assembly lines. *CIRP Annals*, 58(2):628–646, 2009. ISSN 0007-8506.

[6] Arash Ajoudani, Andrea Maria Zanchettin, Serena Ivaldi, Alin Albu-Schäffer, Kazuhiro Kosuge, and Oussama Khatib. Progress and prospects of the human–robot collaboration. *Autonomous Robots*, 42:957–975, 2018.

[7] Swapnil Patil, Vishwa Vasu, and K. Srinadh. Advances and perspectives in collaborative robotics: a review of key technologies and emerging trends. *Discover Mechanical Engineering*, 2, 08 2023.

[8] Iso 10218-1:2025 robotics — safety requirements — part 1: Industrial robots, 2025.

[9] Iso 10218-2:2025 robotics — safety requirements — part 2: Industrial robot applications and robot cells, 2025.

[10] Iso/ts 15066:2016 robots and robotic devices — collaborative robots, 2016.

[11] J. E. Domínguez-Vidal, Nicolás Rodríguez, and Alberto Sanfeliu. Perception-intention-action cycle as a human acceptable way for improving human-robot collaborative tasks. In *Companion of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, page 567–571, New York, NY, USA, 2023. Association for Computing Machinery.

[12] Mica Endsley. Theoretical underpinnings of situation awareness: A critical review. *Situation awareness analysis and measurement*, pages 3–32, 01 2000.

[13] Dylan P. Losey, Craig G. McDonald, Edoardo Battaglia, and Marcia K. O'Malley. A review of intent detection, arbitration, and communication aspects of shared control for physical human–robot interaction. *Applied Mechanics Reviews*, 70(1):010804, 02 2018.

[14] Alexander Mörtl, Martin Lawitzky, Ayse Kucukyilmaz, Metin Sezgin, Cagatay Basdogan, and Sandra Hirche. The role of roles: Physical cooperation between humans and robots. *The International Journal of Robotics Research*, 31(13):1656–1674, 2012.

[15] Panagiota Tsarouchi, George Michalos, Sotiris Makris, Thanasis Athanasatos, Konstantinos Dimoulas, and George Chryssolouris. On a human–robot workplace design and task allocation system. *International Journal of Computer Integrated Manufacturing*, 30(12): 1272–1279, 2017.

[16] Roland Sandor, Clara Wiederschwinger-Fischer, Christina Schmidbauer, and Sebastian Schlund. Decision support model for implementing a human-robot collaboration. 04 2023.

[17] Nikolaos Nikolakis, Niki Kousi, George Michalos, and Sotiris Makris. Dynamic scheduling of shared human-robot manufacturing operations. *Procedia CIRP*, 72:9–14, 2018. 51st CIRP Conference on Manufacturing Systems.

[18] Shozo Takata and Takeo Hirano. Human and robot allocation method for hybrid assembly systems. *CIRP Annals*, 60(1):9–12, 2011.

[19] Yee Yeng Liau and Kwangyeol Ryu. Task allocation in human-robot collaboration (hrc) based on task characteristics and agent capability for mold assembly. *Procedia Manufacturing*, 51:179–186, 2020. 30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2021).

[20] George Evangelou, Nikos Dimitropoulos, George Michalos, and Sotiris Makris. An approach for task and action planning in human–robot collaborative cells using ai. *Procedia CIRP*, 97:476–481, 2021. URL https://www.sciencedirect.com/science/article/pii/S2212827120314931.

[21] Ali Ahmad Malik and Arne Bilberg. Complexity-based task allocation in human-robot collaborative assembly. *Industrial Robot*, 46(4):471–480, 2019.

[22] George Michalos, Niki Kousi, Panagiotis Karagiannis, Christos Gkournelos, Konstantinos Dimoulas, Spyridon Koukas, Konstantinos Mparis, Apostolis Papavasileiou, and Sotiris Makris. Seamless human robot collaborative assembly – an automotive case study. *Mechatronics*, 55:194–211, 2018.

[23] Seemal Asif, Tiziana C. Callari, Fahad Khan, Iveta Eimontaite, Ella-Mae Hubbard, Masoud S. Bahraini, Phil Webb, and Niels Lohse. Exploring tasks and challenges in human-robot collaborative systems: A review. *Robotics and Computer-Integrated Manufacturing*, 97:103102, 2026.

[24] Sandra Robla-Gómez, Victor M. Becerra, J. R. Llata, E. González-Sarabia, C. Torre-Ferrero, and J. Pérez-Oria. Working together: A review on safe human-robot collaboration in industrial environments. *IEEE Access*, 5:26754–26773, 2017.

[25] Andrea Bonci, Pangcheng David Cen Cheng, Marina Indri, Giacomo Nabissi, and Fiorella Sibona. Human-robot perception in industrial environments: A survey. *Sensors*, 21(5), 2021.

[26] Loris Roveda, Palaniappan Veerappan, Marco Maccarini, Giuseppe Bucca, Arash Ajoudani, and Dario Piga. A human-centric framework for robotic task learning and optimization. *Journal of Manufacturing Systems*, 67:68–79, 2023.

[27] Matteo Meregalli Falerni, Vincenzo Pomponi, Hamid Reza Karimi, Matteo Lavit Nicora, Le Anh Dao, Matteo Malosio, and Loris Roveda. A framework for human–robot collaboration enhanced by preference learning and ergonomics. *Robotics and Computer-Integrated Manufacturing*, 89:102781, 2024.

[28] Francesco Iodice, Elena De Momi, and Arash Ajoudani. Intelligent framework for human-robot collaboration: Dynamic ergonomics and adaptive decision-making, 2025.

[29] Shuvo Kumar Paul, Mircea Nicolescu, and Monica Nicolescu. Enhancing human–robot collaboration through a multi-module interaction framework with sensor fusion: Object recognition, verbal communication, user of interest detection, gesture and gaze recognition. *Sensors*, 23(13), 2023.

[30] Jonghan Lim, Sujani Patel, Alex Evans, John Pimley, Yifei Li, and Ilya Kovalenko. Enhancing human-robot collaborative assembly in manufacturing systems using large language models. In *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, pages 2581–2587. IEEE, 2024.

[31] Fabrizio Flacco, Torsten Kroeger, Alessandro De Luca, and Oussama Khatib. A depth space approach for evaluating distance to objects. *Journal of Intelligent & Robotic Systems*, 80(1):7–22, 2015.

[32] Benjamin Navarro, Andrea Cherubini, Aïcha Fonte, Robin Passama, Gérard Poisson, and Philippe Fraisse. An iso10218-compliant adaptive damping controller for safe physical human-robot interaction. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3043–3048, 2016.

[33] Carlos Morato, Krishnanand Kaipa, Boxuan Zhao, and Satyandra Gupta. Toward safe human robot collaboration by using multiple kinects based real-time human tracking. *Journal of Computing and Information Science in Engineering*, 14:011006, 03 2014.

[34] tkinter. https://docs.python.org/3/library/tkinter.html.

[35] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. Springer, 2009.

[36] Franka robotics. https://franka.de/.

[37] Franka Robotics GmbH (formerly Franka Emika GmbH). *Franka Emika Robot's Instruction Handbook*, 2021. https://download.franka.de/documents/100010_Product%20Manual%20Franka%20Emika%20Robot_10.21_EN.pdf.

[38] Franka Robotics GmbH. *Franka Control Interface (FCI) Documentation*, 2023. https://frankarobotics.github.io/docs/index.html.

[39] Robotiq. https://robotiq.com/products/adaptive-grippers.

[40] Robotiq Inc. *2F-85 & 2F-140 General Instruction Manual*, 2020. https://assets.robotiq.com/website-assets/support_documents/document/2F-85_2F-140_General_PDF_20200211.pdf.

[41] Ros. https://www.ros.org/.

[42] RViz. GitHub repository:https://github.com/ros-visualization/rviz.

[43] gazebo. https://gazebosim.org/home.

[44] Optitrack. https://optitrack.com/.

[45] Motive. https://optitrack.com/software/motive/.

[46] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE International Symposium on System Integrations (SII)*, 2019.

[47] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2008.

[48] Ruckig. https://ruckig.com/.

[49] qpoases. GitHub repository:https://github.com/coin-or/qpOASES.