WP4 – Robot arm programming

D4.2 – Robot control functionalities demonstration

Due date: M33 - 31/07/2022

Delivery date: M33 - 30/07/2022

Version number: V1

CONFIDENTIAL UNTIL PROJECT FINALIZATION

Responsible partner

CEA : B. GRADOUSSOFF

Contributing partners

CEA : G. ACHER, J. DUMORA, F. GEFFARD, F. LEGENDRE, V. MOLINA, N. BRUCKMANN, M. NDIAYE

CASP : A. SARDELIS

Summary

This deliverable shows the functionalities of the robot arm control technologies described in deliverable D4.1 - Report on robot arm control technologies, through experimental demonstration of elementary use from user's side.

The first part focuses on the skill-based programming framework and the associated learning toolbox for skills optimization. It details the various available workflows, and the elementary experiments done for the Thimonnier and Selmark use-cases.

The second part focuses on the toolbox of on-line robot control laws dedicated to human-machine interaction (robustified intention detection). It details the experiments done in the frame of the VDL use-case, for large flexible fabrics co-manipulation on one side, and for large fragile parts co-manipulation on the other side.

Table of content

1. Skill Programming Interface for Robotic Environments (T4.1 & T4.3)

This part presents the available building blocks and associated experiments, both for skill-based programming (T4.1), and for the associated AI-based optimization toolbox (T4.3).

1.1. S.P.I.R.E Framework overview

Robotic manufacturing poses many challenges related to the ease of programming, versatility and deployment of automated tasks. For the Merging project, CEA developed a skill-based programming interface for robotic environments (SPIRE) with the aim of easing the programming and deployment of flexible, complex robotic tasks in order to tackle these challenges. The framework integrates a reusable skill concept, an associated library, a hardware abstraction layer to ensure interoperability through standardized APIs, and a multi-modal demonstration module based on motion capture, with the goal of simplifying the workflows of operators, integrators and roboticists. We evaluate the capability of the architecture on fabric manipulation and precise assembly, and conclude that our prototype provides adequate tools for developing complex operations, then easily adapting and sequencing them.

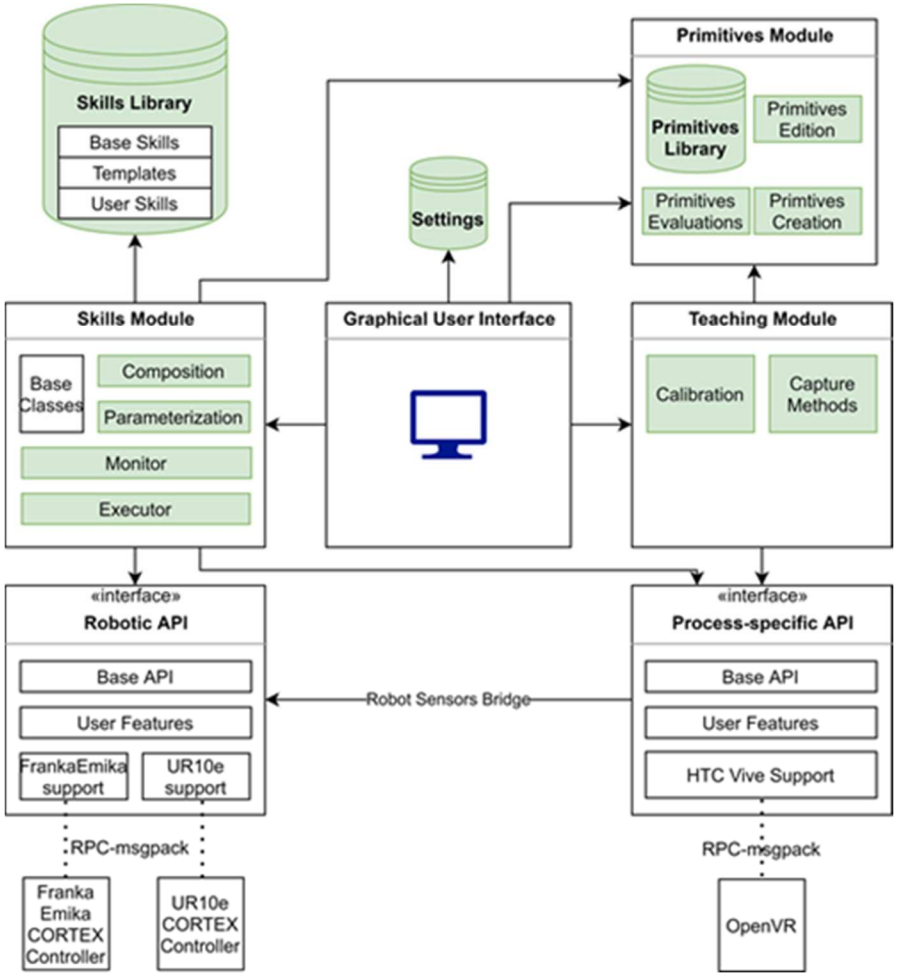


Figure 1 - Features of the SPIRE framework

1.2. Overview of the SPIRE Workflows

During the design of the framework, CEA kept in mind the specific needs associated to the various user roles involved in the lifecycle of a robotic task.

The next figure summarizes the different workflows anticipated for the project:

- Researchers design new high-level algorithm or low-level control patterns.
- Integrators extend the support to new robots and sensor hardware.
- Process engineers perform minor skill developments either with the GUI or by scripting.
- Operators create the final skill by composing existing skills using the GUI, configure the skills with demonstrations, perform adjustments as well the skill parameterization, and monitor the execution (and autonomous parameters optimization if needed).

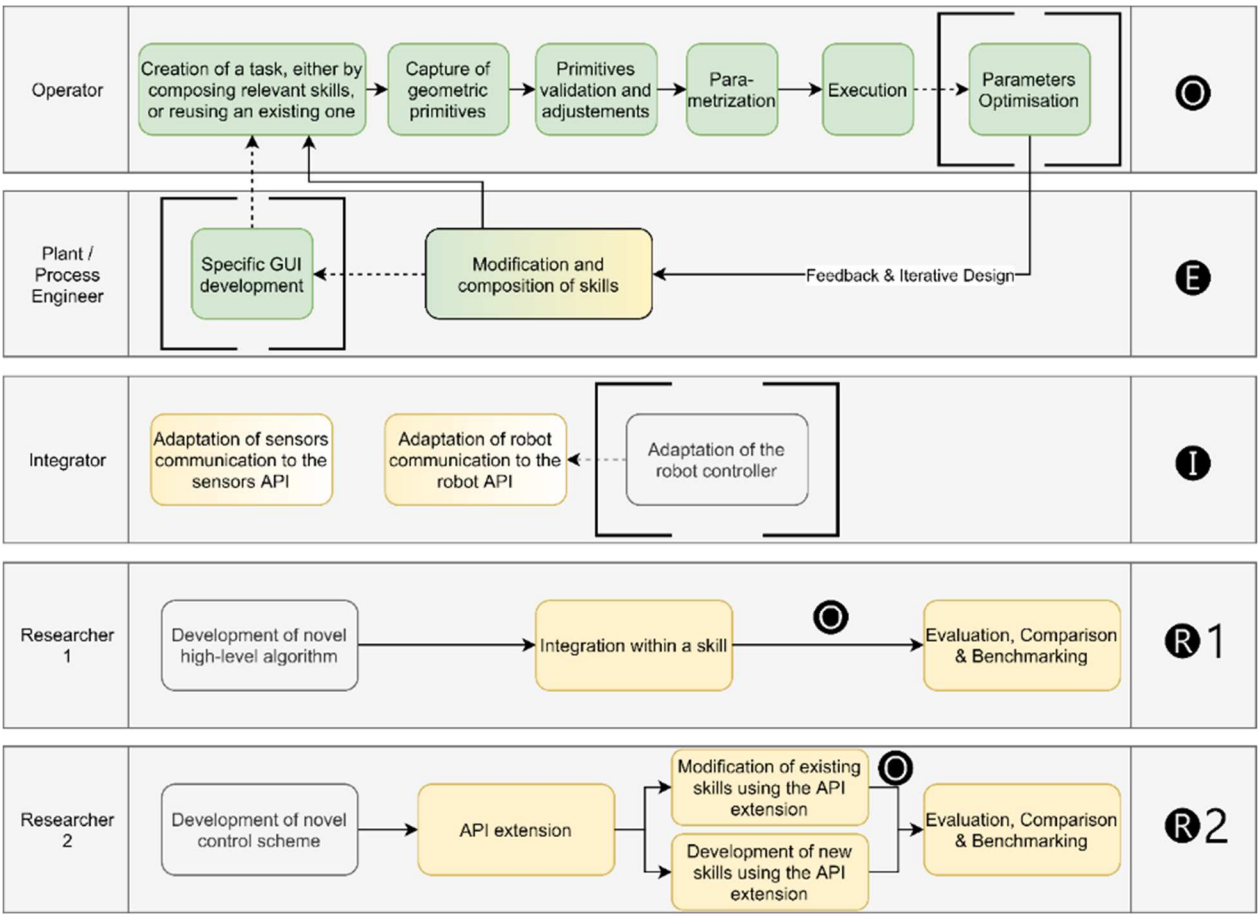


Figure 2- Overview of the different workflows (green steps are performed through the GUI, yellow steps are performed by coding/scripting within the framework, and grey steps depend on external software/hardware. Bracketed steps are optional)

Even though the operator workflow was the main focus in this project, CEA anticipated that the further continuation and deployment of the framework would be significantly facilitated if designed by taking into account these different roles and uses.

1.3. Workflow for operators

The core philosophy of our proposed workflow is illustrated by next figure. Skills are high-level, task-related functions, easily understandable by the operator. A skill is applied by a resource (such as a robotic

arm), onto a target (such as a piece of fabric). Most skills need geometric primitives (points, trajectories, etc.) as inputs, these primitives are teached by the operator using one of the available teaching modalities (co-manipulation, motion-capture, etc.). AI-based optimization algorithms can help the operator to fine-tune the skill configuration parameters. The correct setup of these parameters is usually necessary for the skill to perform adequately for a given use-case.

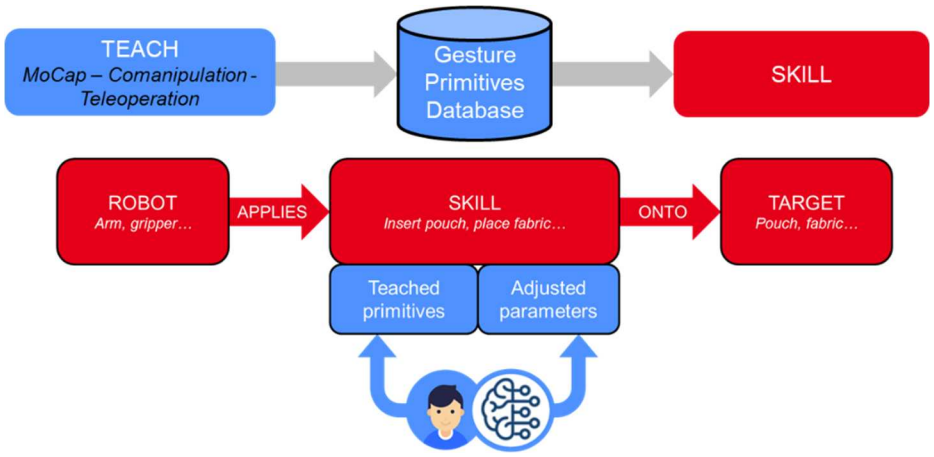


Figure 3 - Core workflow of SPIRE

1.3.1. Setup

1.3.1.1. SPIRE launch and configuration

The initial setup required minimal intervention. The main software is launched though a single executable, although the robots controllers and sensor servers have to be started manually.

The user has to define the devices used for demonstrations. *Figure 4* illustrates the case of bimanual motion capture teleoperation where two distinct controllers are associated.

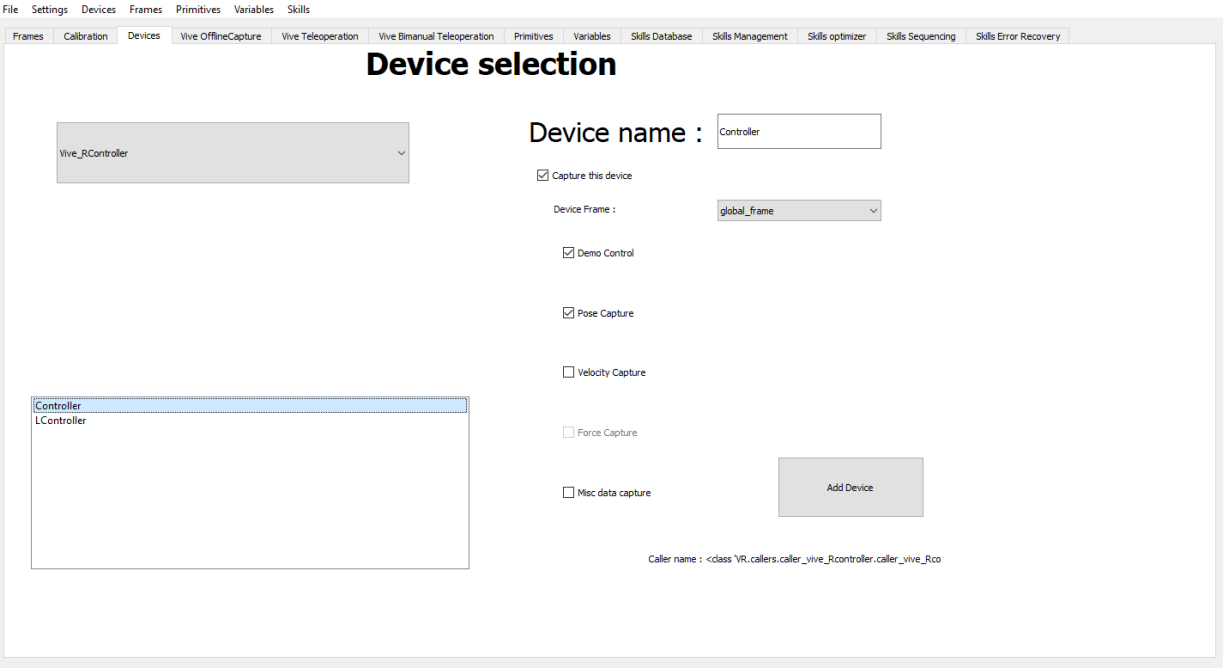


Figure 4 - Demonstration device definition

1.3.1.2. Motion capture setup

The hardware setup required for using motion capture consists in 2 or 4 'lighthouses' mounted around the robotic cell, as to minimize the likeliness of occlusion. The various trackers and controllers can be paired through the existing steamVR interface. CEA uses an HTC VivePro 2 system, which requires the use of a Windows system, thus the motion capture communication software is executed on a different computer than the main framework (SPIRE currently runs on Linux for practical reasons).

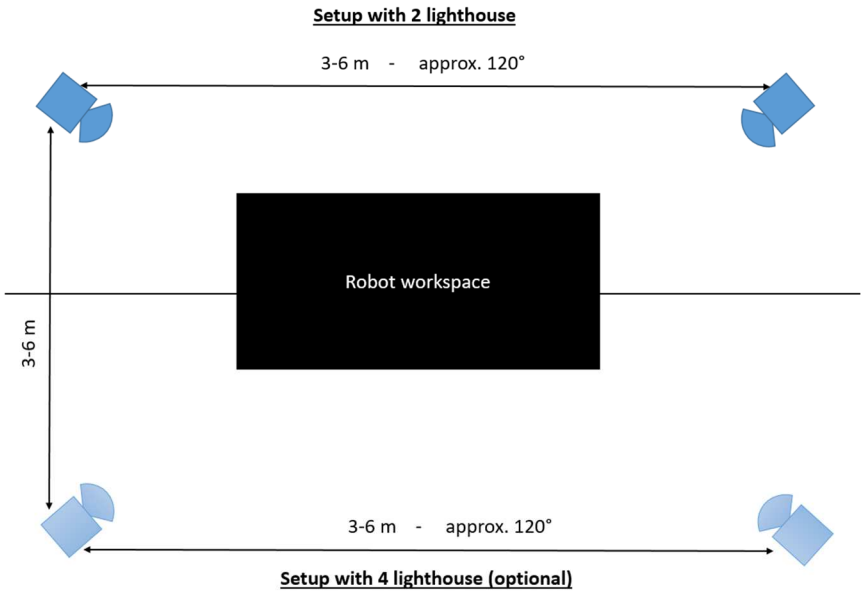


Figure 5 - HTC Vive motion capture setup

A dedicated server makes available the OpenVR through RPC calls, and needs to be started before the next steps.

In order to perform the preliminary calibration phase, the robot gripper has to be fitted with custom tips so the tracker can be mounted on the robot, as illustrated by *Figure 6*. Related CAD files are available within the SPIRE directory if adaptations to new grippers are needed.

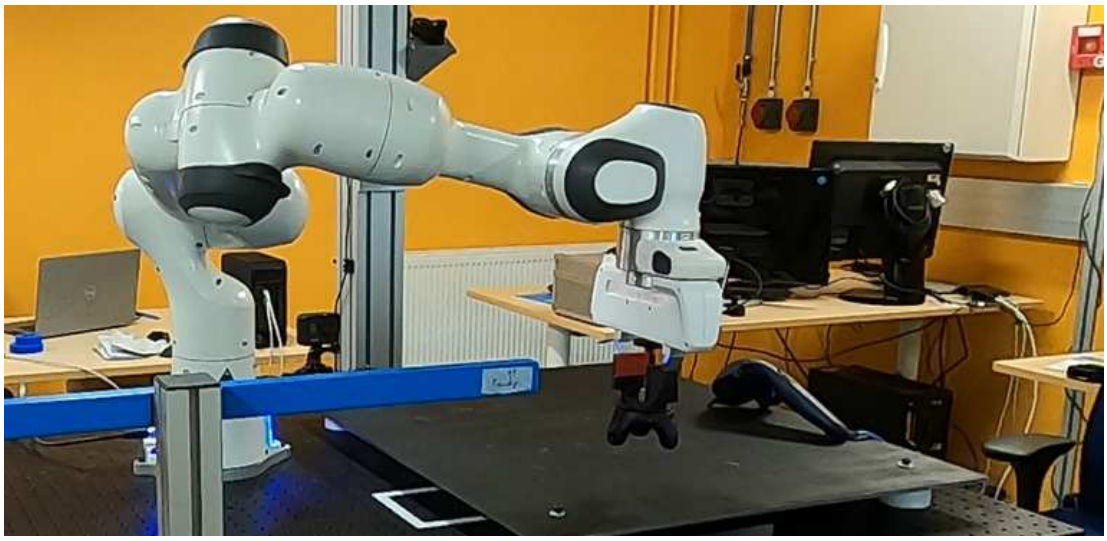


Figure 6 - Calibration with the motion capture tracker fitted on the robot

Using the GUI (*Figure 7*), the user can then select the algorithm and calibration trajectory, before starting the process. The interface offers the possibility to open the gripper so that the operator fits the MoCap trackers onto the robot, before closing it to lock the position. Two methods out of four available offer the possibility of gathering data in a compliant mode so the operator can focus a specific area to perform the calibration. Otherwise, the robot moves automatically to predefined waypoints in order to cover a larger volume.

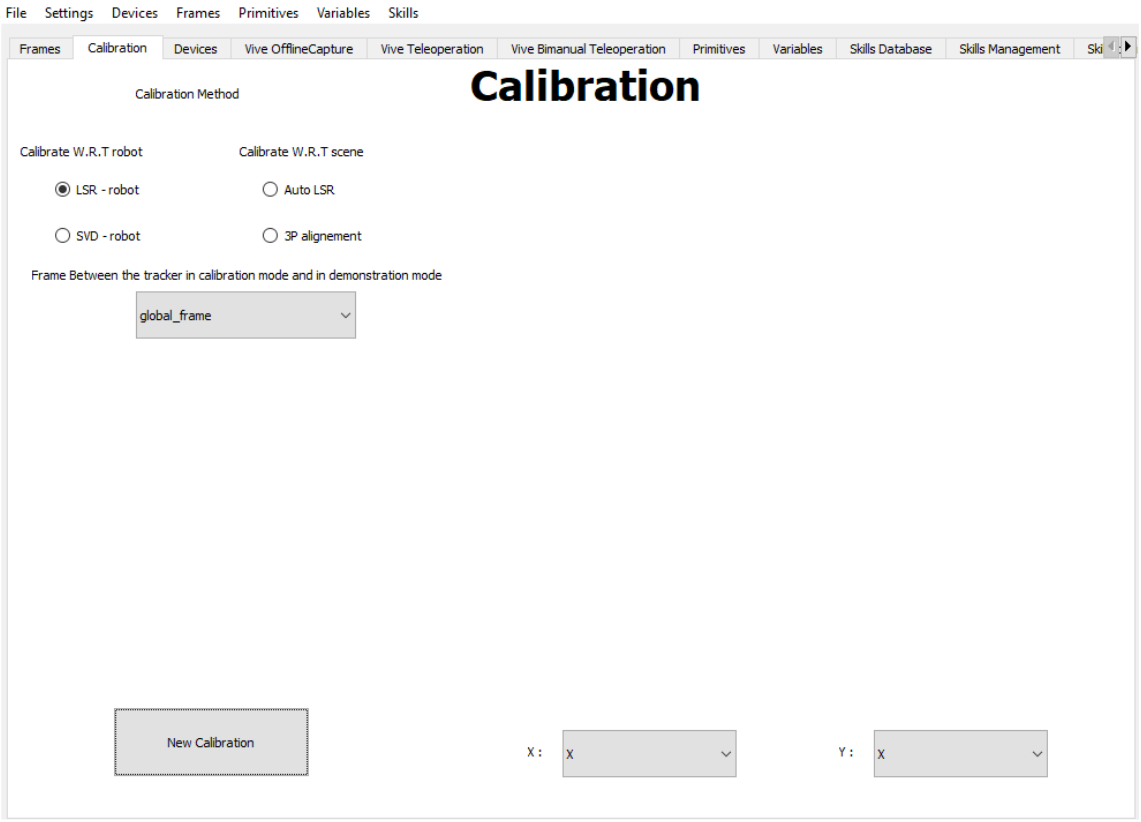


Figure 7 – Calibration interface

After the execution is completed, the robot switches to a compliant mode so that the operator is able to move it throughout the workspace with the tracker still attached during a few dozen seconds. A synthetic plot of this validation step is displayed, enabling the operator to check the quality of the calibration.

The operator can save and load calibration data through the GUI. However, it is recommended to perform a new calibration when starting a new capture session, as the MoCap firmwares tends to update its internal calibrations, sometimes leading to unwanted drifts in the arbitrary Vive reference frame.

1.3.2. Task-level composition

If needed, operators can perform high-level skills compositions to adapt the task sequence to the context. SPIRE presents a list of the available skills and their hierarchy, as seen in *Figure 8*.

The robotic task can be made of skills picked among the available skills:

- The standard library, which encompasses both basic functionalities and more complex generic behaviours
- User-specific skills that were developed or adapted specifically for the current task.

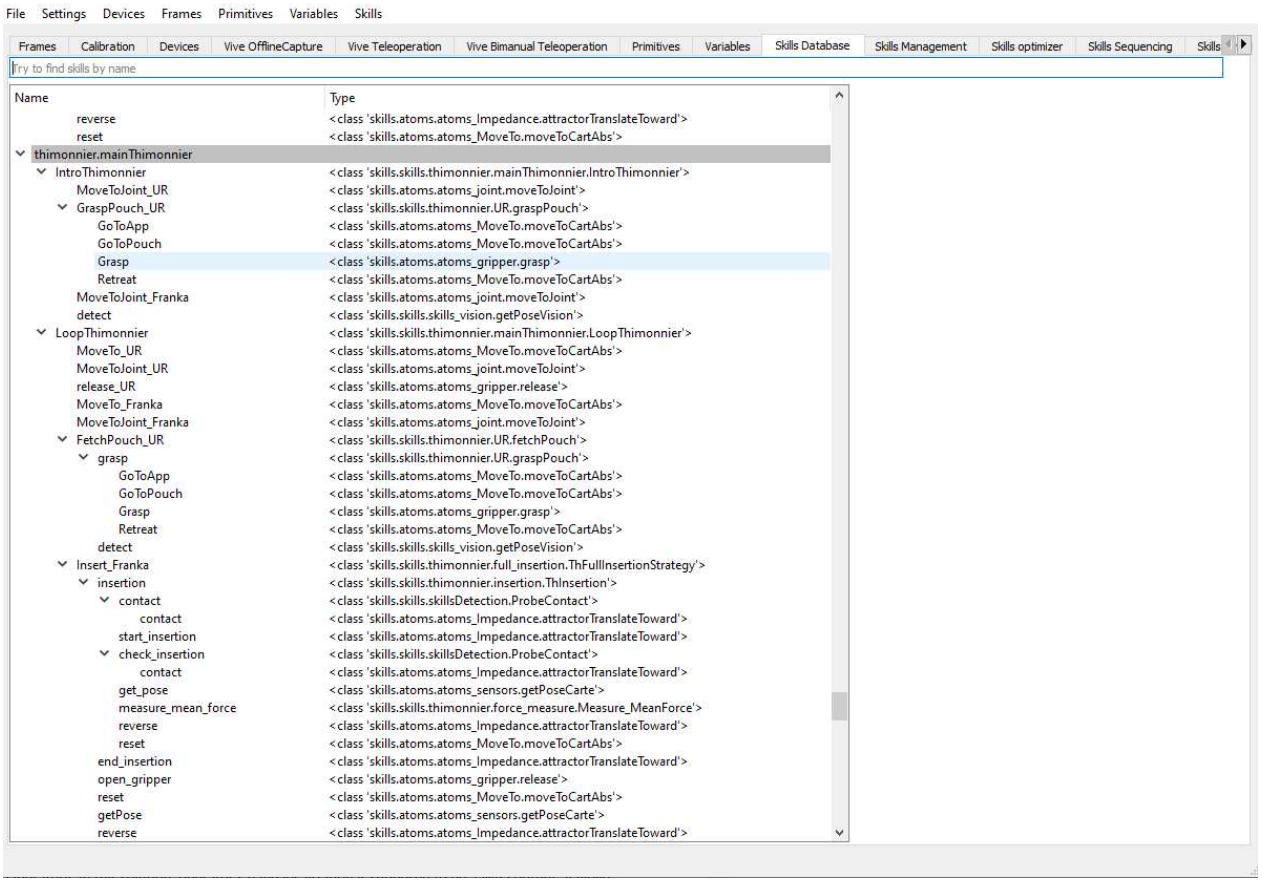


Figure 8 - Skills list and hierarchy

SPIRE presents dedicated graphical interfaces for skill composition and sequencing, following an operator-focused no-code approach. The most basic composition consists in a simple sequence, as illustrated by next figure.

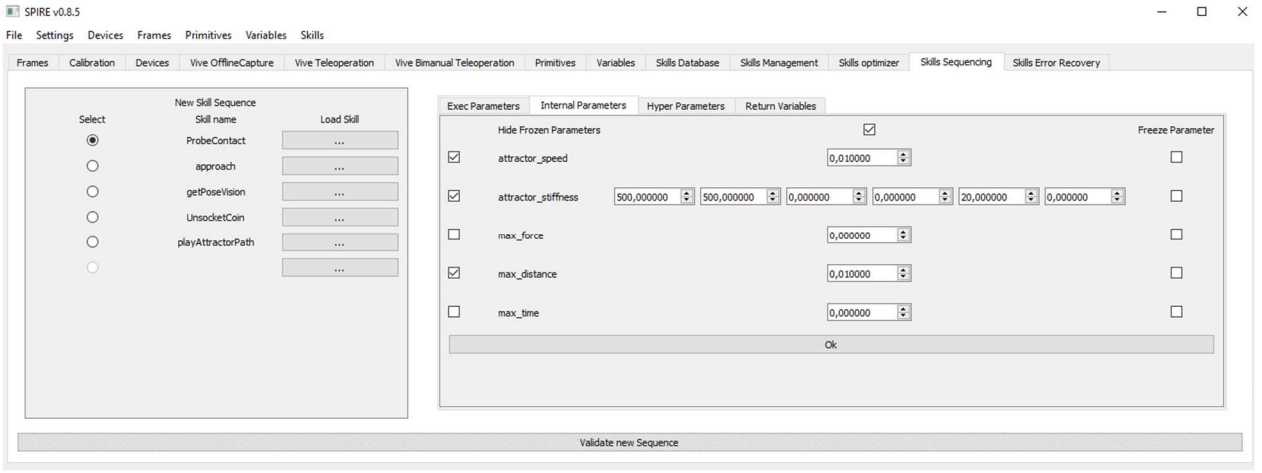


Figure 9 - Screenshot of GUI for skill composition (sequential execution)

It is also possible to integrate recovery behaviours. For instance, the user can switch the robot to a compliant mode whenever a force threshold is reached (i.e. a collision occurs).

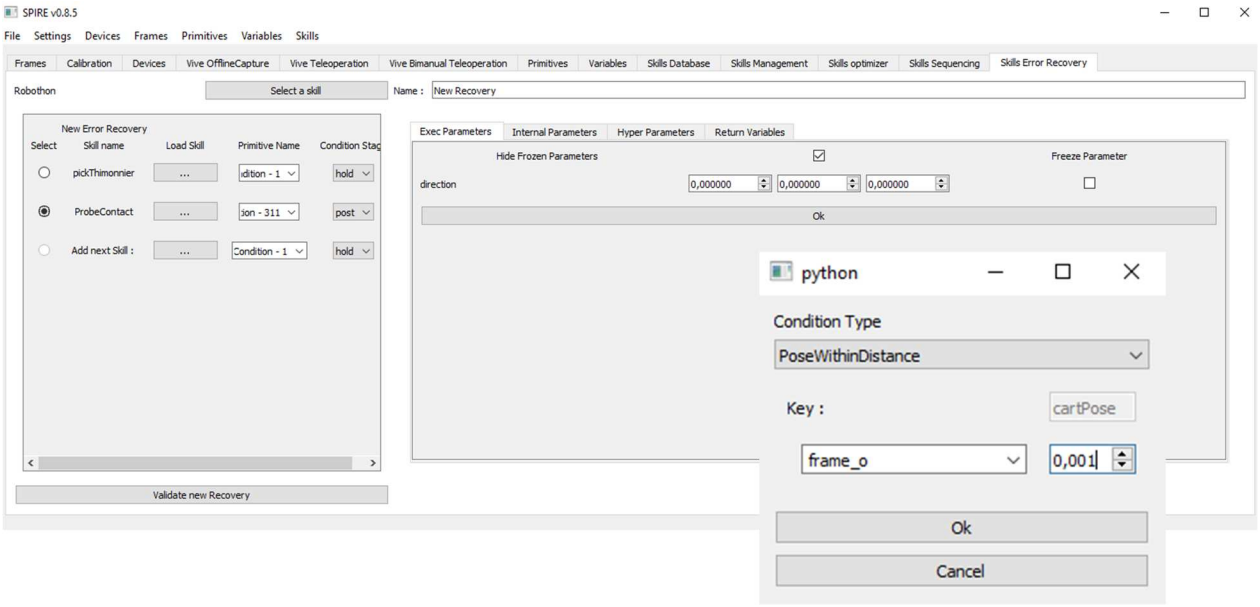


Figure 10 - Example of GUI for Skills error recovery management

The newly defined composition can then be saved and loaded.

For now, it is preferable to compose skills with the GUI only in the case of the highest level task. In the absence of code generation, workers with knowledge of the scripting mechanisms are not able to perform in-depth script modifications at the highest level. However, the current system has the advantage of not requiring any further work when lower levels skills undergo modifications: when loading a composition, both the code and parameters signatures of the subskills are directly extracted, meaning that any modification of the base subtasks is taken into account.

Once this step is completed, the operator can gather parameters and primitives, as shown in the next part. Note that this step is optional, as the main structure can be directly scripted by an engineer.

1.3.3. Demonstrations

Once the skill sequence is defined, the operator has to configure each skill with its needed input data. Geometric data, which we call (geometric) primitives, are naturally fit to be teached by demonstration. SPIRE supports multiple ways of performing demonstrations, depending on the requirements of the task at hand.

1.3.3.1. Offline motion capture

CEA fitted an HTC Vive controller with a 3D printed pointing tip. The trigger is used to perform demonstrations :

- A short click records a pose.
- A long press records a trajectory until the trigger release.

Using these controls, the user is able to gather multiple primitives in a row in the robotic cell, without having to go back to the main computer and interact with the interface.

One of the advantages of this demonstration modality is that the pointing part enables demonstrations of geometrical features independently of the robot, and most particularly the gripper. For instance, next figure shows the demonstration of the rail insertion site for the Thimonnier use-case as a completely abstract frame. The insertion skill then uses this abstract information, along with the demonstration of the other end of the rail, to define the rail axis and provide the suitable gripper orientation for the insertion.

Figure 11 - Offline MoCap demonstration of the rail insertion site

Higher-level features like lines and planes can be inferred using these abstract poses.

Offline mocap also enables demonstrating trajectories with excellent ease and dexterity. Combined with trajectory adaptation features, transport movements can be programmed quickly to adapt to contextual constraints and obstacles.

This approach has limitations:

- The absolute accuracy of the MoCap system does not allow very accurate operations to be directly executed. This can be solved with parameter optimization, trajectory adaptation using accurate waypoints, or contact-probing skills for instance.
- The user must consider reachability, kinematics and collisions aspects, since captures are completely independent from the robot. This can also be seen as an advantage, as it allows defining the task needs in terms of reachability, which can be used to specify the robot choice or its positioning within the scene.

1.3.3.2. MoCap-based teleoperation

This is an interactive mode: as one moves the controller, the robot imitates the trajectory. The coupling is done in cartesian space: the user controls the movements of the robot end-effector. The user activates the coupling by pressing the trigger. By releasing the trigger, the operator can move around the workcell, and place its arm back in a more convenient configuration.

Compared to conventional force-feedback teleoperation, using a dedicated master arm which gives a proportional feedback of the interaction forces between the slave arm and its environment, this solution doesn't provide force sensing. However, it provides a very useful haptic feedback by forwarding the norm

of the interaction forces to the integrated vibration motor of the MoCap controller. This way, the user knows if the robot applied forces during contact, and has an appreciation of their magnitude, although one has to infer the direction of these forces visually. During contact, a click on the trigger nulls out all external forces applied by resetting the followed attractor to the current position. Furthermore, to allow for a stable and comfortable use, the cartesian controller of the robot is less stiff/rigid than in most common applications, so that contacts and interactions remain manageable without reaching the robots limits.

Various user assistances have been implemented to provide a better control of the robot, some of them are directly mapped on the MoCap controller. Next figure presents the layout of these user assistances, which include :

- A position homothety switch (1:1, 1:2, 1:4) for finer control. At 1:2 position, a 10cm displacement of the operator results in a 5cm displacement of the robot.
- Gripper open/close controls.
- Various axis locks using virtual guides (in the local gripper frame or global reference frame). Notably, the “Reach Vertical Orientation” is a very useful feature for many common applications where the gripper has to be set perpendicular to the (horizontal) table.
- Remaining sectors of the central wheel can be assigned to triggering pre-defined skills.

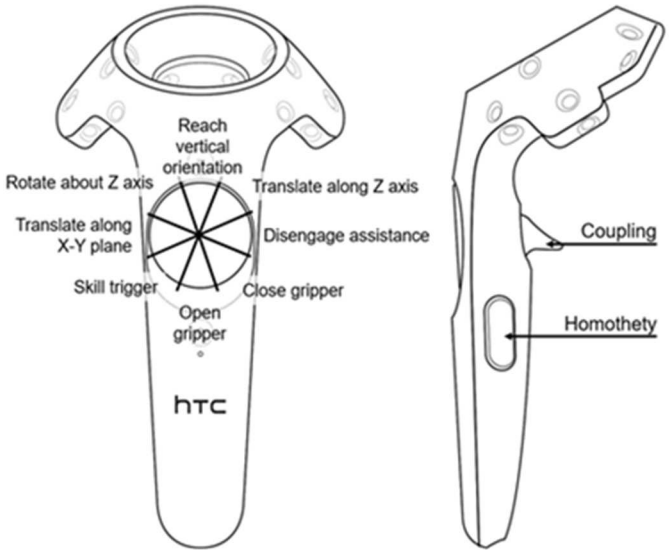


Figure 12 - Teleoperation controls layout

As depicted in the next figure, the current selection of user assistance is displayed live on the user interface. GUI buttons are used to trigger pose captures, as well as starting and stopping trajectory captures. Since coupling is only active when the trigger is pressed, it is easy for the operator to lock the robot in position by releasing the trigger, and safely capture the pose on the GUI. Similarly, when capturing trajectories, data is only recorded while the coupling is activated. This way, the operator can freely reposition itself during the demonstration.

This demonstration modality fixes the accuracy and robot kinematics issues of offline motion capture, since it’s the real robot position that is teached, not the MoCap controller position. Moreover, teleoperation provides a solution when the actual robotic cell is not easily accessible (for safety reasons, or because it stands in a cramped space for instance). However, compared to offline motion-capture, captures take slightly longer to trigger, and complex trajectories are less convenient to perform.

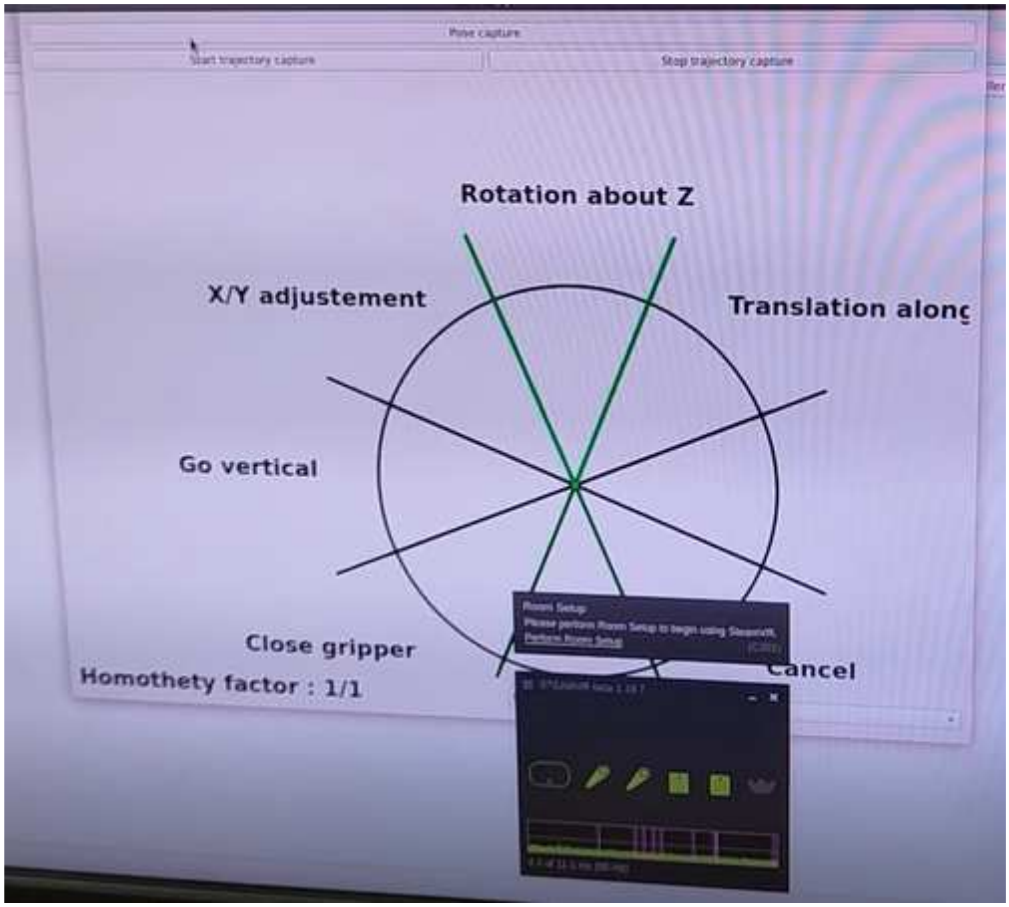


Figure 13 - Teleoperation demonstration interface

1.3.3.3. Bi-manual teleoperation

Mocap teleoperation was extended to bimanual capability. Controllers can be assigned to the robots via the GUI.

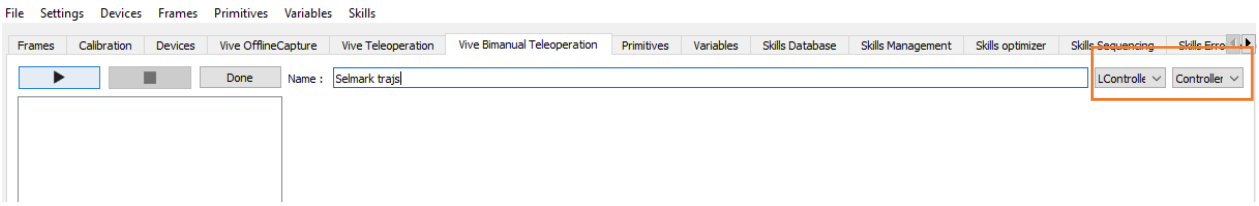


Figure 14 - Controller selection for bimanual teleoperation

The rest of the workflow is similar to the previous teleoperation section, with the exception that each capture generates two primitives, each labelled with an ‘R’ or ‘L’ after its base name to make a distinction between both robots.

Figure 15 - Bimanual teleoperation for Selmark ply manipulation

1.3.3.4. Cobot

A cobot can be used for demonstration, similarly as offline mocap. The cobot must be defined as a capture device in the setup interface, as pictured in *Figure 16*.

Handling the cobot is significantly more tedious than the previous methods, especially since the compliance is not perfect. This proved to be especially problematic when demonstrating trajectories. Moreover, demonstrating points precisely is inconvenient, as fine movements require a lot a focus and patience from the user.

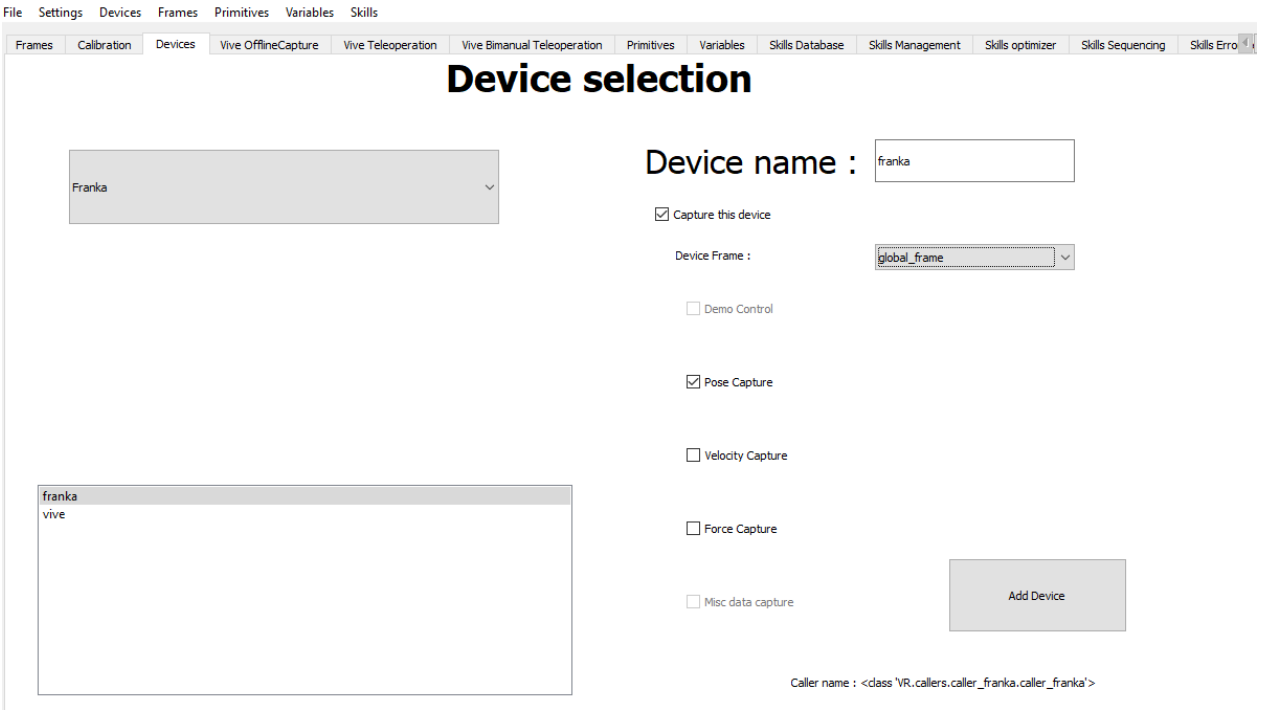


Figure 16 - Cobot device definition, similar to mocap devices

1.3.3.5. Teaching modalities comparison

The table below summarizes the strengths and weaknesses of each modality:

Capture Mode	Ease of deployment	Accuracy	Ease of use	Force-aware
Offline MoCap	=	-	**++**	No
MoCap Teleoperation	+	**++**	+	**Partially (haptic feedback)**
Cobot	**++**	**+**	=	**Yes (direct feedback)**

The MoCap teleoperation modality is more precise than offline motion capture (as it records the real position of the robot), ensures that demonstration positions are located within the workspace of the robot, and prevents unanticipated gripper collisions in the vicinity of obstacles. However, it is less manoeuvrable that offline motion capture, since the operator has to consider the risk of collision while manipulating. It remains very intuitive when manipulating two robots at once and when demonstrating trajectories, which is definitely not the case for traditional cobotic manipulation.

Overall, MoCap-based teaching modalities are very user-friendly, and provide consistent user-experience, which is definitely not the case for direct co-manipulation. Co-manipulation performance is very dependant on the cobot models. In many cases, poor transparency (inertia, friction) make the cobot difficult and cumbersome to manipulate, which can affect effective accuracy of the system, due to the difficulty to position finely the cobot.

1.3.4. Primitives post-processing and testing

Once primitives are captured, the user can manage them through the GUI. All primitives are listed in a panel, where they can be grouped in a tree-like structure. By selecting a primitive on this list, the user is able to edit the name and description. The data is also displayed and editable, and expressed within a selectable workspace frame, which can be convenient is the user uses a reference related to the workcell instead of the robot frame.

Each type of primitive comes with a set of operations, processes and transformations made available via dedicated buttons. This is especially convenient for trajectories (*Figure 18*), which often require cropping, smoothing or other methods to enhance the quality of the raw data. Multi-selection (*Figure 19*) enables to perform transformations on multiple primitives at the same time, as well as fitting a set of poses to a line or plane. This latter feature can be used to mitigate imprecisions when demonstrating rectilinear or planar features.

Certain primitives can be tested using a button:

- In the case of poses, the robot moves to the specified point
- In the case of trajectories, the robot moves to the initial point then plays the primitive. The trajectory can be adapted between starting and ending pose primitives beforehand, via the GUI.

High-level primitives can be created with specific methods with the graphical interface, as pictured in *Figure 20*, which illustrates frame creation possibilities, from one or several poses.

Overall, SPIRE provides a set of tools that enable the user to create, monitor, edit, and test primitives before using them as skills parameters.

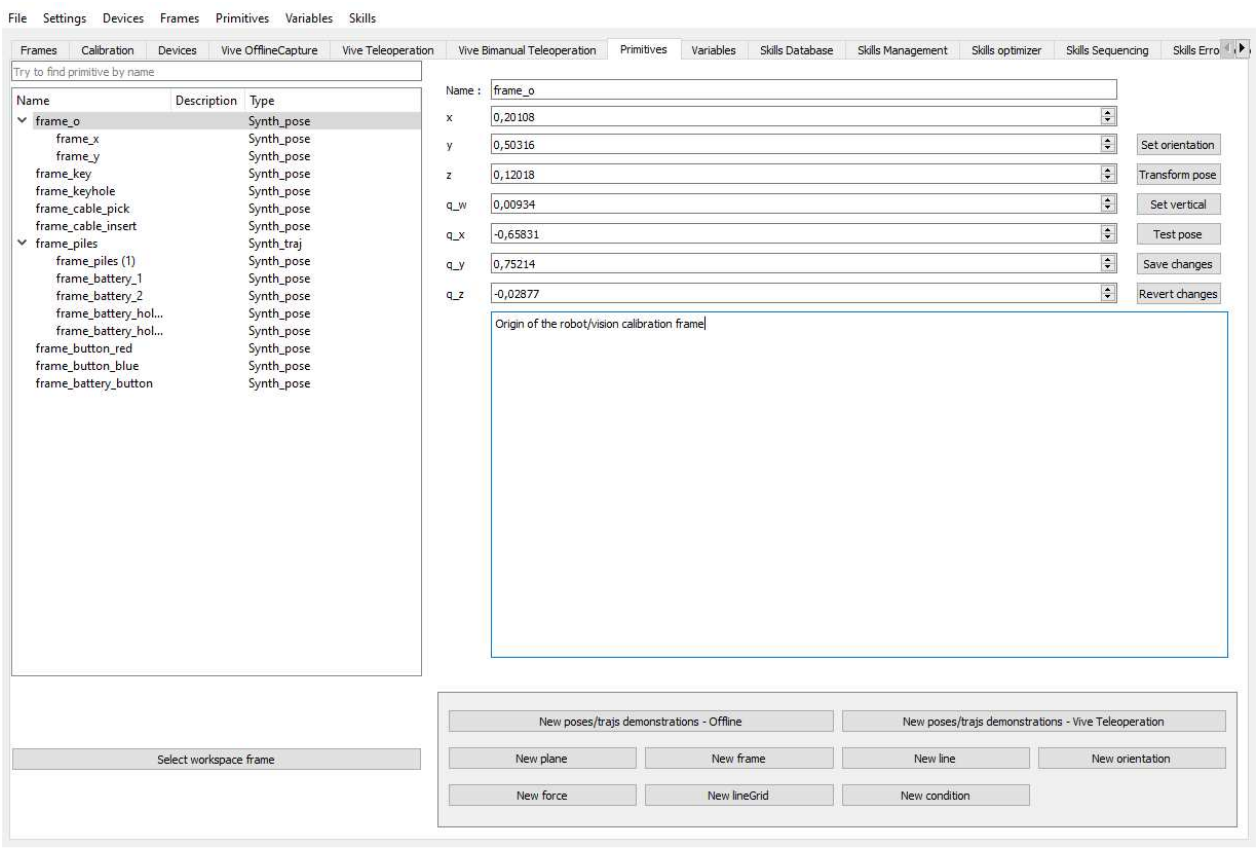


Figure 17 - Primitive interface for a pose

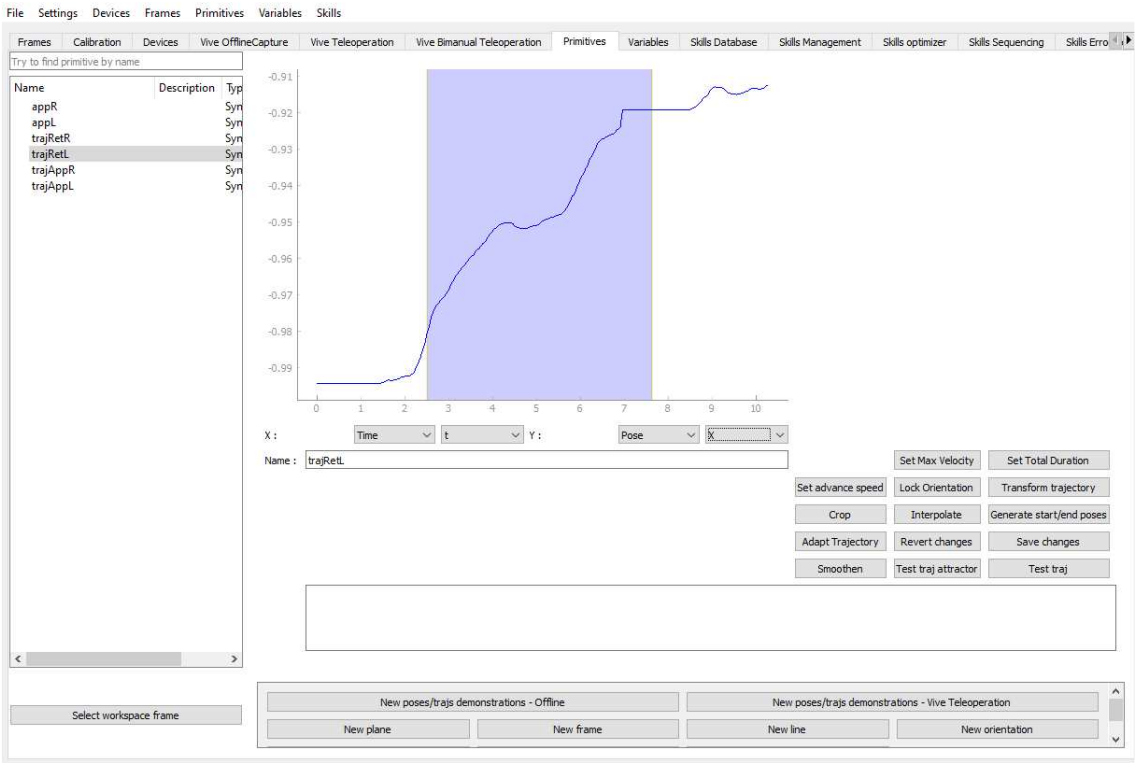


Figure 18 - Primitive interface for a trajectory

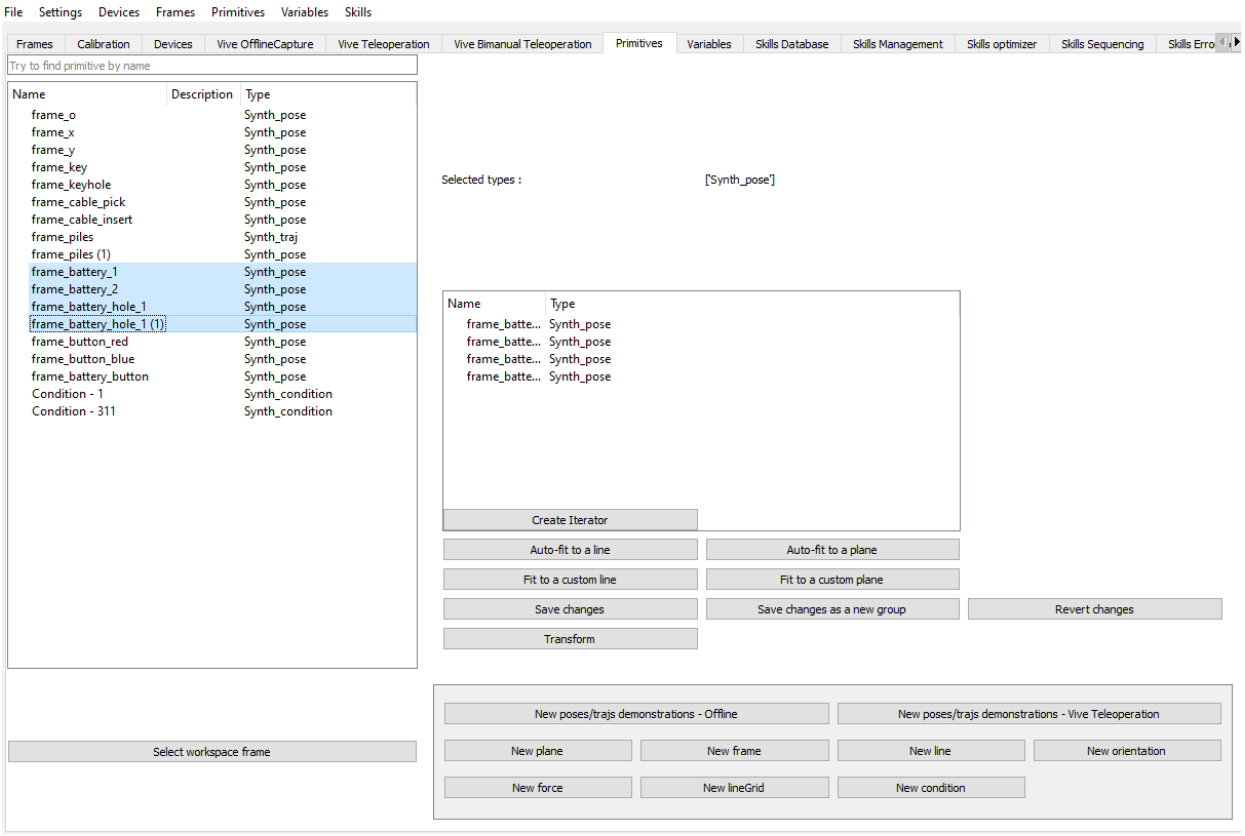


Figure 19 - Primitive interface for a multi-selection

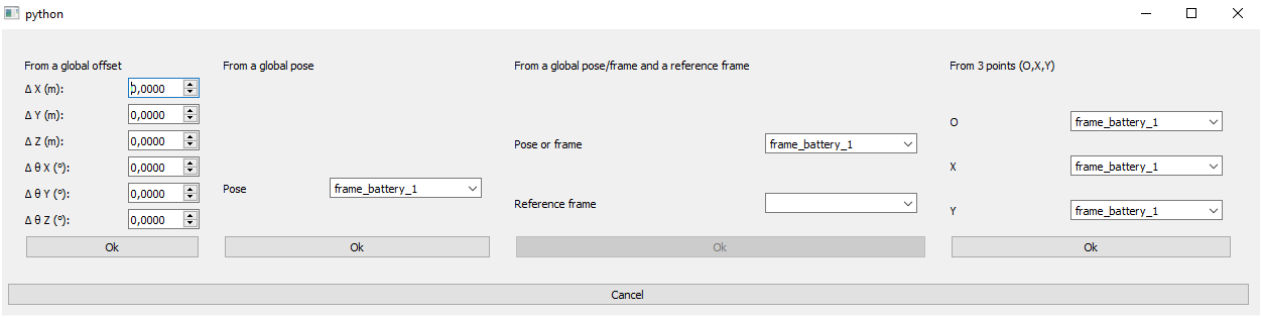


Figure 20 - Primitive creation interface for a frame

1.3.5. Skills parameterization

Now that skills were defined and that primitives were captured and processed, the parameterization stage starts. The interface is directly generated from the skill metadata. For each parameter, primitives are filtered according to the required type. Values can be entered manually in the case of numerical parameters. Parameters can be either mandatory (*Figure 21*) or optional (*Figure 22*). In that case, the user can select whether one wishes to specify a value or use the default one (for optional parameters, the skill designer has defined a default value).

Parameterized skills can be saved and loaded.

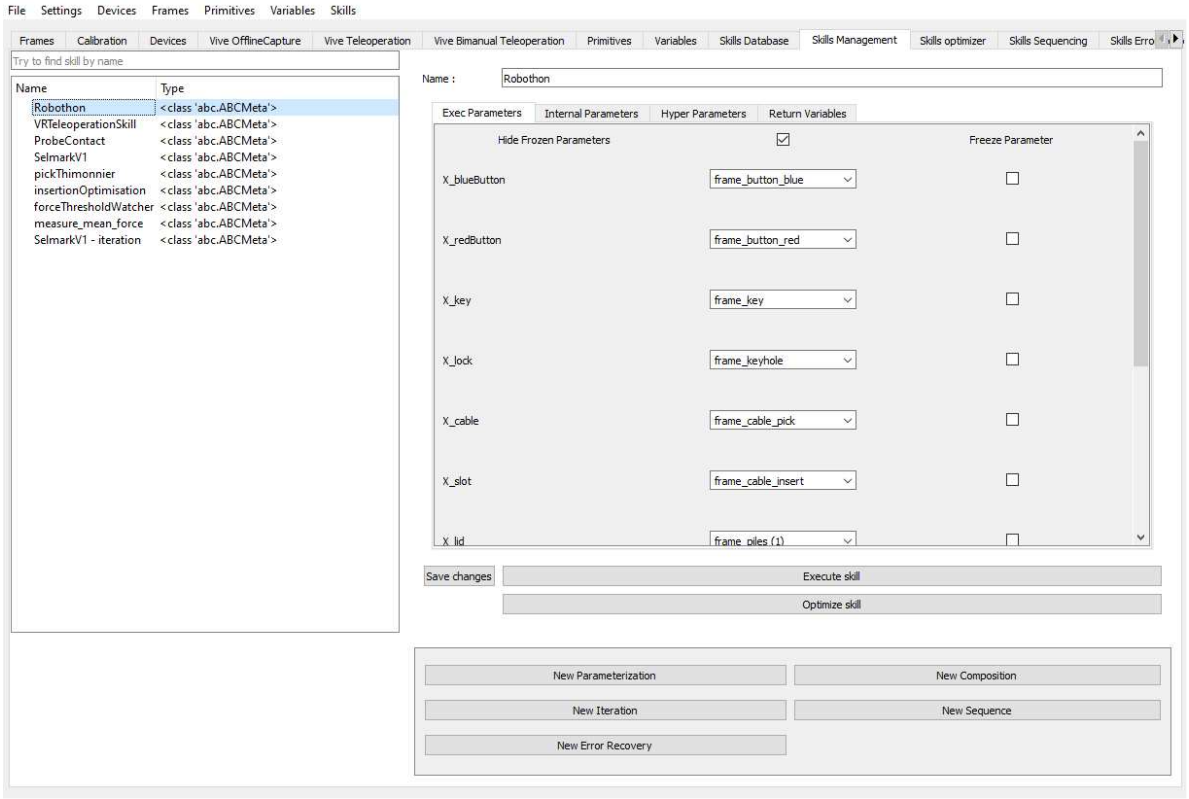


Figure 21 - Skill parameterization interface for mandatory parameters

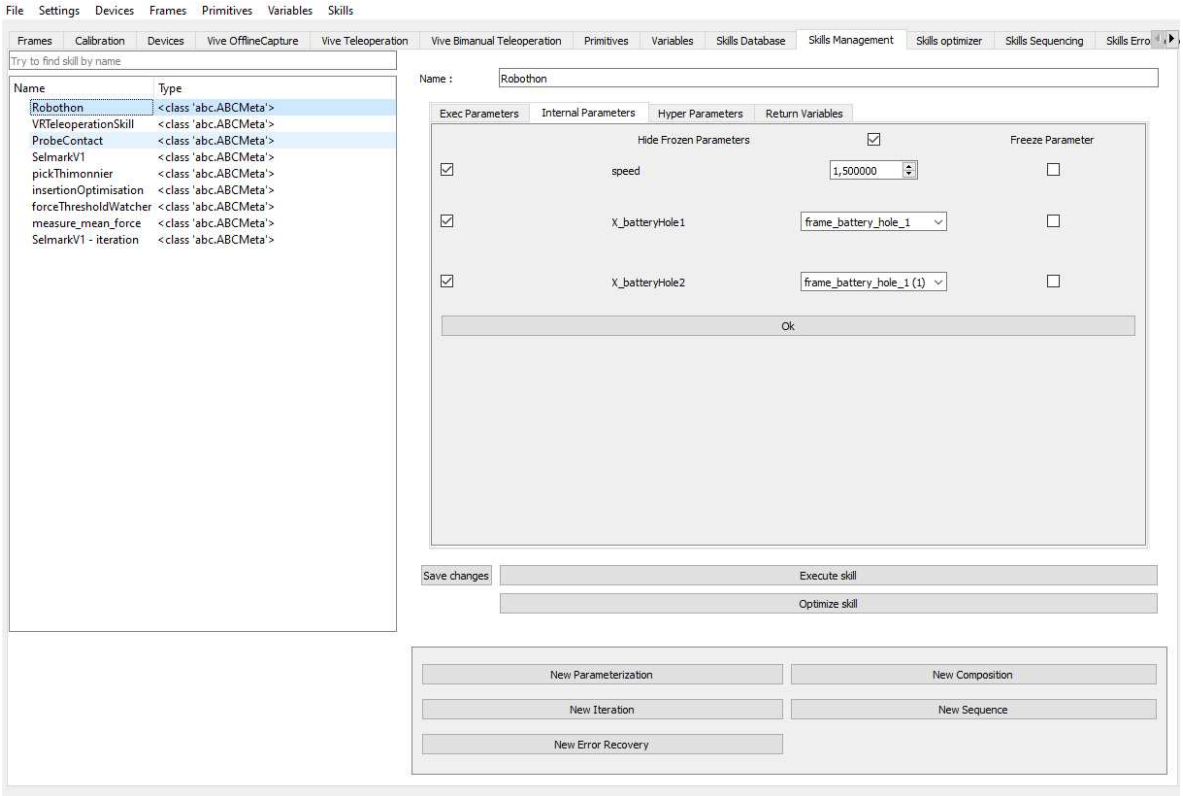


Figure 22 - Skill parameterization interface for optional parameters

1.3.6. Execution

While a skill is being executed, the interface displays the sensor data and the current state of the skill hierarchy.

Live control of the execution (speed coefficient and pause) via the GUI is in development.

1.3.7. Optimization

1.3.7.1. Method

In many cases, skills can benefit from having some of their parameters optimized. For instance, the Thimonnier use-case required to perform an optimization of the insertion site location, as the acceptable clearance was far smaller than the precision of the primitive captured with motion capture. The choice of these hyper-parameters and their range is left to the skill designer, as well as the optimization plan and algorithm choice. For the details about optimization algorithms, please refer to deliverable D4.1.

The user starts by starting an optimization server and loading a new optimization session via the GUI. An existing session can also be loaded in order to resume an optimization. The optimization is similar to a regular execution, with the exception that at each trial, the agent communicates a set of values to the skill to be evaluated, as it investigates the best performing combinations. While the optimization itself is autonomous, assistance of the user might be required, if the object slips within the gripper for instance.

The progress and results of the optimization is plotted at each iteration (cf. next figures, which provide an example of such results).

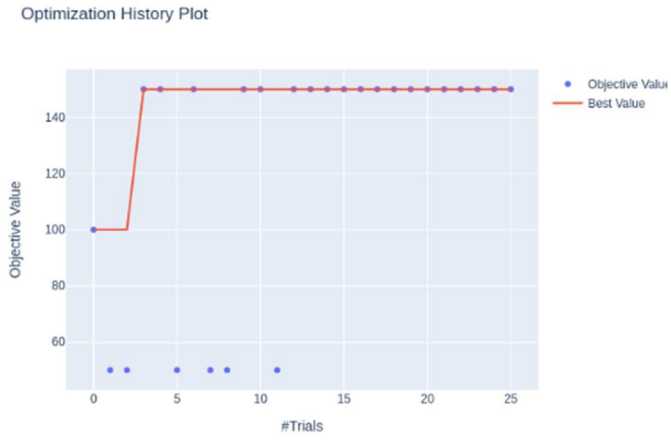


Figure 23 - Optimization history plot

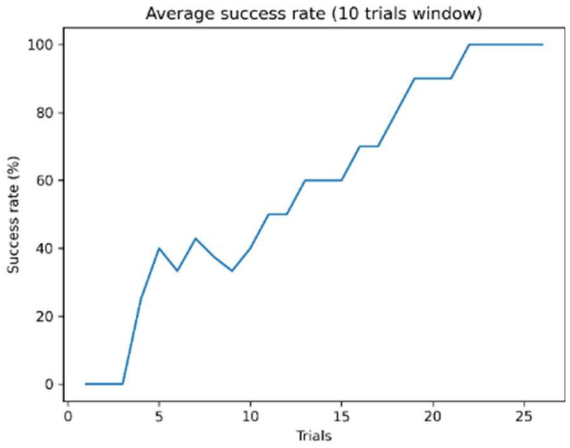


Figure 24 - Optimization performance plot over the course of the training, filtered with a 10 trials moving average

A typical two-parameters optimization usually needs a few tens of trials to converge. Applied using an automatic trial and error sequence, this optimization generally needs only a few minutes to realize those trials on the real system.

In order to find an optimal set of parameters, the operator can either use the output of the integrated clustering algorithm, or infer it visually on convergence plots. Next figure shows the convergence plot for the current example.

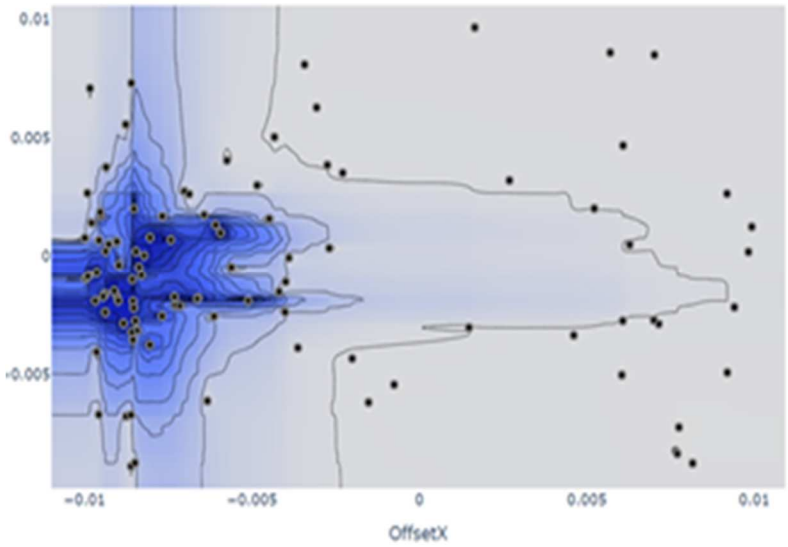
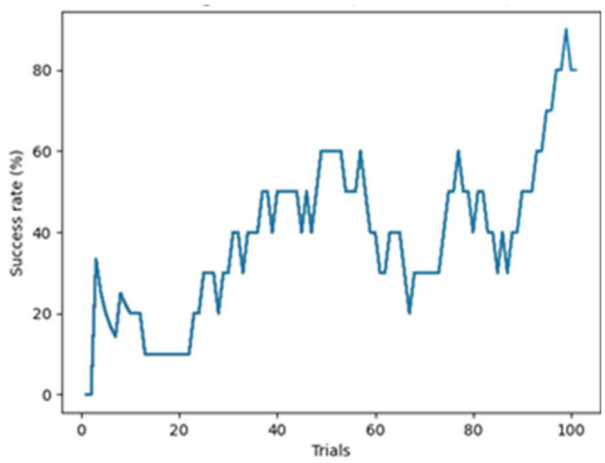
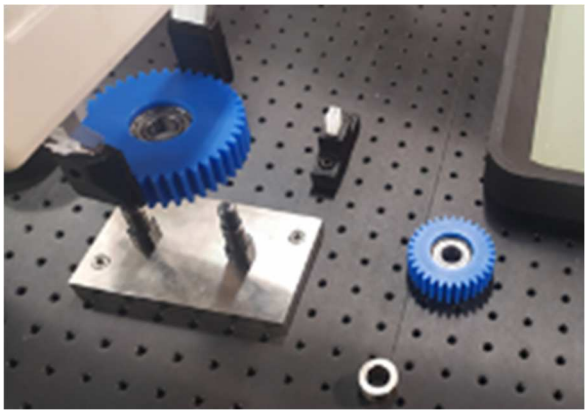


Figure 25 - Convergence plot for a geometrical X-Y offset (blue is better)

1.3.7.2. Optimization example

The first experiments on optimization were performed on a seemingly simple task: inserting a gear on an axis. Due to the critically low clearance (<10µm), this insertion was performed using hybrid force-position control with independent Cartesian stiffness components and Lissajous oscillations. The location of the target axis was demonstrated with offline motion capture. However, the limited precision of this modality was originally incompatible with such a demanding operation. As a result, CEA chose to optimize X and Y offsets in order to directly compensate the geometric error of the demonstrated location of the axis, as well as stiffness and oscillations magnitude as to improve the insertion robustness and efficiency.





As illustrated in the previous figure, a clear convergence towards a X-Y offset centroid was observed and the average success rate was significantly improved compared to trials using only the raw captured primitive. Even though the Lissajous oscillations mitigate the inaccuracy in a limited measure, they also introduce noises which slow the training by introducing false positives and negatives. Also, the overwhelming importance of the geometrical corrections meant that the other hyper parameters could

not be trained along, but rather in a following step instead. This illustrates the need for a specific "optimization sequence strategy".

1.4. Complementary workflows

1.4.1. Workflow for process engineers

1.4.1.1. In-depth scripting

As the rest of the framework, skills scripting relies on the Python programming language. This choice was motivated by the ease of use of this high-level language, and the fact that modern engineering courses often include Python classes, making it accessible to a majority of engineers.

Skill scripting consists in filling in headers (*Figure 26*) and callbacks (*Figure 27*) within a template. The most important callbacks include:

- An initialization method for synthetizing or acquiring high level information for the pre-condition, or perform initialization actions.
- For cyclic skills: a loop method for control.
- For composed skills: a sequence method for scripting the unfolding of the core task and the compositions.
- A checkout method, for performing final or validation actions, as well as synthetizing high-level data for the post-condition and formatting return data.
- A get_sensor_data method to enrich the exchange object with further information obtained via specific API calls. Note that this method is wrapped as to automatize the safe communication of this data.

In addition to callbacks, a skill integrates its own metadata, used for client discovery and exploitation:

- Specify parameters and return data, their types, and if needed their default value.
- Specify conditions.
- Various metadata

The set of available methods for primitives processing and skills composition, combined to the ease of use of Python and the existing skills library, aims at facilitating the development of new skills.

The current generic library contains :

- Basic movements (both in cartesian and joint spaces)
- Random trajectory replay with or without adaptation
- Transport skills
- Assembly skills like insertions
- Disassembly skills like extractions
- Buttons pushes
- Contact probing
- Compliant modes
- Hybrid force/position strategies
- Measurements
- Gripper controls
- Teleoperation skills

```python
class attractorControlRelative(baseAtom):
    #################
    ## Meta
    #################
    name = 'attractorControlRelative'
    robotRequirements = {}
    #################
    ## parameters
    #################
    execParamsNames = ['X','Stiffness']
    execParamsTypes = [[pose],['R+','R+','R+','R+','R+','R+']]

    internalParamsNames = ['timeLimit','moveUntilWithinEpsilon','maxInitialDistanceFromAttractor','ensureGraspWidth','damping']
    internalParamsTypes = [[pose],['R+','R+','R+','R+'],['R+','R+'],['R+','R+','R+','R+','R+','R+'],]

    hyperParamsNames = ['Xoffset']   #PUT EXACT KEYWORDS HERE
    hyperParamsTypes = [[frame]]

    #################
    ## returns
    #################

    returnNames = []
    returnTypes = []
```

Figure 26 - Extract of a skill header

```python
    def _sequence(self):
        feedback = self.sequence('getPoseCarte')

        self.initPose = self.extractCheckout(feedback)
        #self.subExecParams['releaseAttractor']={}
        self.sequence('releaseAttractor')

        if self.thresholdForce[0] is not None and self.attractorDisplacement[0]!=0 :
            stiffnessX = [self.forceRatio * self.thresholdForce[0]/self.attractorDisplacement[0],800,800,20,20,20]
            # attractorX = self.initPose
            attractorX = self.initPose.offset([self.attractorDisplacement[0],0,0,0,0,0], copy=True)
            self.subExecParams["setAttractorX"] = {'X':attractorX, 'Stiffness': stiffnessX}
            feedback = self.parallelize('setAttractorX', 'forceThresholdWatcher', interruptors=['forceThresholdWatcher'])
            result = self.extractCheckout(feedback['forceThresholdWatcher'])
            if result[0] == True:
                self.constrainedDoF[0] = True

        if self.thresholdForce[1] is not None and self.attractorDisplacement[1]!=0:
            stiffnessY = [800,self.forceRatio * self.thresholdForce[1]/self.attractorDisplacement[1],800,20,20,20]
            # attractorY = self.initPose
            attractorY = self.initPose.offset(0,[self.attractorDisplacement[1],0,0,0,0], copy=True)
            self.subExecParams["setAttractorY"] = {'X':attractorY, 'Stiffness': stiffnessY}
            feedback = self.parallelize('setAttractorY', 'forceThresholdWatcher', interruptors=['forceThresholdWatcher'])
            result = self.extractCheckout(feedback['forceThresholdWatcher'])
            if result[1] == True:
                self.constrainedDoF[1] = True

        if self.thresholdForce[2] is not None and self.attractorDisplacement[2]!=0:
            stiffnessZ = [800,800,self.forceRatio * self.thresholdForce[2]/self.attractorDisplacement[2],30,30,30]
            # attractorZ = self.initPose
            attractorZ = self.initPose.offset([0,0,self.attractorDisplacement[2],0,0,0], copy=True)
            self.subExecParams["setAttractorZ"] = {'X':attractorZ, 'Stiffness':stiffnessZ}
            feedback = self.parallelize('setAttractorZ', 'forceThresholdWatcher', interruptors=[1])
            result = self.extractCheckout(feedback['forceThresholdWatcher'])
            if result[2] == True:
                self.constrainedDoF[2] = True
```

Figure 27 - Extract of a sequence method

1.4.1.2. Composition by scripting

Composing new skills via scripting offers more freedom than codeless composition via the GUI. It mainly consists in completing the *sequence* callback. This solution requires a certain knowledge of the framework, but offers significantly more flexibility than compositions with the GUI. For instance, the programmer can take advantage of the range of operations implemented within the primitives classes. Moreover, one can write custom composition rules.

1.4.1.3. Composition with the GUI

As seen in section 1.2.2, a user can create new skills by sequencing them or adding error recovery behaviours. For a process engineer, another composition type can be relevant: parallelization, as pictured in next figure.

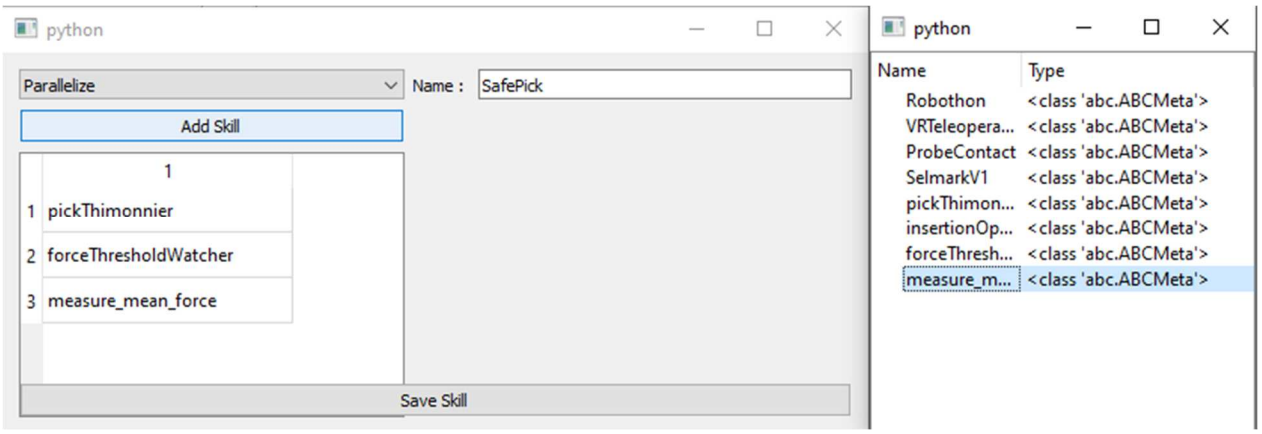


Figure 28 - Parallelization GUI

While the GUI is significantly easier to use, it is as for now to be limited to the highest levels, as the absence of code generation makes it impossible to script further compositions on top of it.

1.4.2. Workflow for integrators

1.4.2.1. Extending the support to new robots

SPIRE includes a base robotic API class, which is used as an abstraction layer to ensure robot interoperability. Since robot manufacturers offer various degrees of access to low level control, this API is split in different features, so that some robots can at least be partially supported. Moreover, the variety of end-effector leads to a similar split.

The first robot the receive an implementation of the robotic API was the FrankaEmika Panda. CEA used its in-house CORTEX modular control framework to implement basic control features as well as more advance force control modes.

The support was then extended to a UR10e robot. The lack of low level access prevented the use of CORTEX, thus the force control capabilities of the resulting implementation were limited to factory available features. Supported end-effectors include a custom gripper, a RobotIQ 2F-85 gripper, and a custom vacuum gripper.

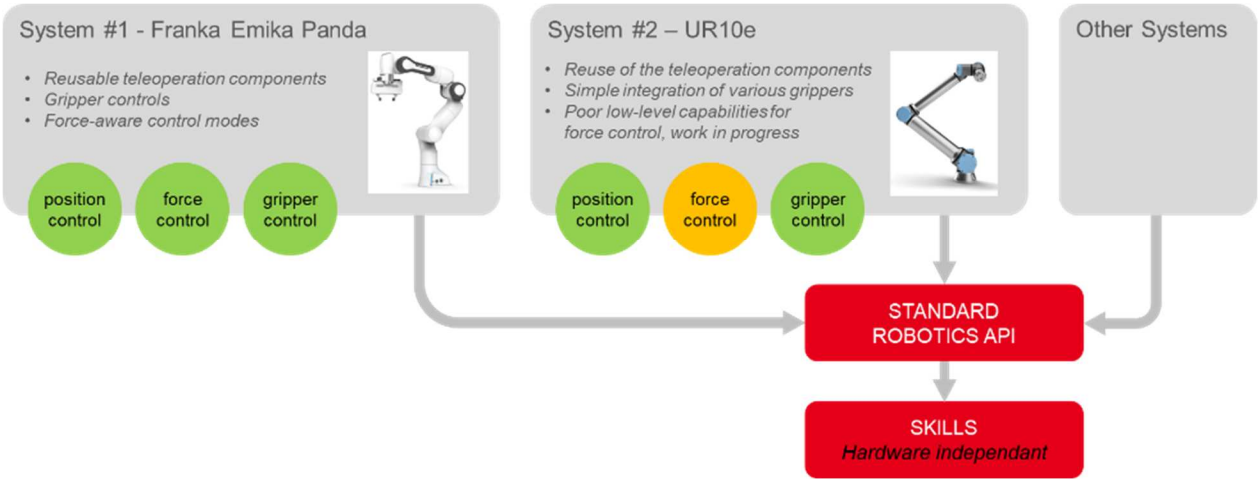


Figure 29 - Implementations of the robotic API

The main advantage of this abstraction layer is that once the implementation is completed, no further development is needed to transfer the skill library from a robot to another, thus significantly cutting the integration effort.

1.4.2.2. Extending the support to new devices/sensors

Current specifications are fairly weak, but programming a simple bridge that substitutes the communication and formats the data as a previously existing device/sensor makes it possible to integrate such device faster.

1.4.3. Workflow for researchers

1.4.3.1. Easing integration, development, and testing

The range of features included in SPIRE have the potential of easing the burden of setting up an entire demonstration to evaluate and illustrate research work. Firstly, the standard APIs ease the hardware integration by providing a library of existing implementations and providing examples for the support extension process. The hardware agnosticism means that the research work can be more easily transferred or replicated.

Moreover, the off-the-shelf skills library can provide a baseline for skills performance, versatility and convenience, and enable comparing strategies on concrete applications. Also, the library is a convenient resource to build more relevant demonstrations, that would present more realistic operations that a 'peg-in-hole' process. Building the whole infrastructure took a significant amount of time, therefore limiting the number of different approaches CEA could try for the merging use-cases. However, CEA intends to develop new skills as needed for the integration phases of Merging.

1.4.3.2. Designing low-level control algorithms

Advanced task-related skills such as fine manipulation or assembly tasks often require complex force/position control patterns, sometimes with context-specific variants. Moreover, some features, like reactive contact detection, are better included in the low-level layer. Therefore, the robotic API can be

extended incrementally to support these new features. Finally, improved versions of existing control patterns can be substituted conveniently.

As an example, the Thimonnier use-case lead CEA to implement an attractor based control method to handle interactions during the insertion stage. This pattern and all the related setting methods were interfaced within a new section of the robotic API. Later, another version of the prototype used high frequency Lissajous oscillations on top of this pattern. Such frequency could not be managed directly from the SPIRE framework, thus this feature was implemented in the low-level controller and interfaced in the robotic API.

1.4.3.3. Designing high-level algorithms

CEA implemented its motion capture teleoperation feature as a skill, as to take advantage of the hardware abstraction, both on the robot and on the tracker side. Moreover, this approach makes it easier to use skills as user assistances or custom controls, and build user-interactive scenarios alternating automated and manual phases. Another notable feature of the teleoperation skill is its specific interface with the dedicated GUI that reflects the state of the user assistances. High level algorithms can also be developed within the primitive classes. For instance, CEA initially implemented Kernelized Movement Primitives (KMP) as a way to synthetize a movement behaviour from multiple trajectories. Even though the development was discontinued for performance and reliability reasons, the range of primitives processing methods is bound to be extended over time with increasingly complex algorithms. Instead of KMP, analytical trajectory adaptation algorithms were implemented within the primitives classes, providing a robust way to create adaptive transport movements.

Autonomous parameters optimization is another example of a high-level feature that takes advantage of the SPIRE framework, notably the skills formalism, the conditions system and the error recovery capabilities. Similarly, this paves the way for future experimentations with online reinforcement learning.

1.5. Application on the Thimonnier use-case

The main goal of this demonstration was to validate the following features:

- Hardware interoperability.
- Offline Motion Capture.
- Skills reusability.
- Advanced scripted compositions.
- Contact and interaction management, force-based features.
- Error detection and recovery.
- Optimisation.
- Interface with external sensors.
- Complex parallelization.

1.5.1. Setup

1.5.1.1. Hardware setup

The experimental setup involves two robotic arms: an UR10e with a vacuum gripper, and a FrankaEmika Panda with custom gripper tips. We decided to go for a two-robot setup in order to facilitate pouch handling. Without the high dexterity provided by the Shadow hand, it is extremely hard to tackle both (1)

pouch grasping on a flat surface and (2) pouch insertion into a rail with a single end effector. The two-robot setup allows dissociating those two tasks, and using the most suited end-effector for each one.

The workcell also includes a Photoneo M 3D sensor for pouch localisation and an HTC Vive setup for demonstrations.

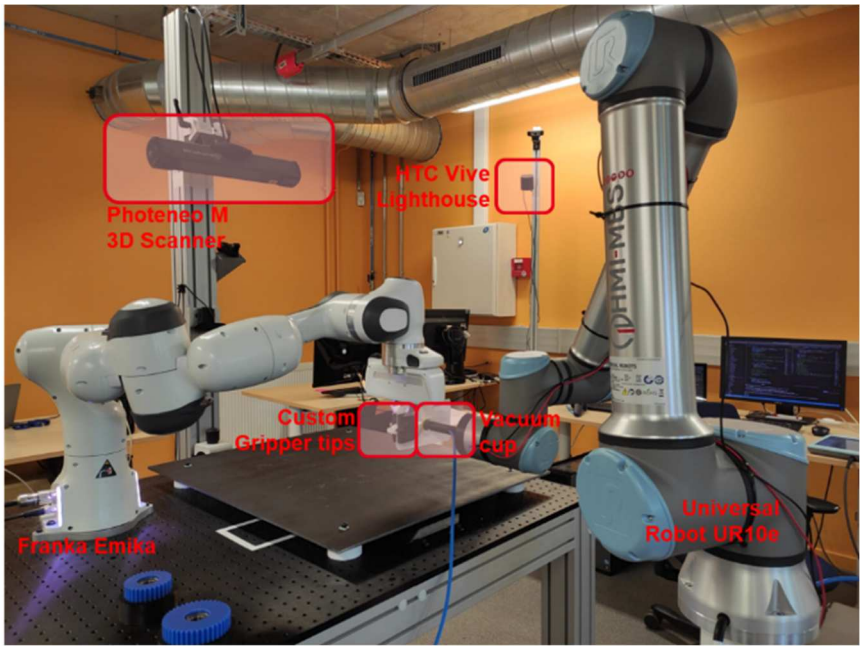


Figure 30 - CEA setup for the Thimonnier use-case

Motion capture demonstrations are used to define the rail position and orientation. They enable non-specialized operators to modify intuitively the target location of the rail, thus offering an increased flexibility in operations.

1.5.1.2. System precision

The main difficulty for robotic rail insertion is to get sufficient precision to align correctly the pouch tip with the rail. The precision of whole system is limited due to several factors (MoCap precision, Calibration errors, Robot precision), while the insertion clearance is submillimetric. To address this issue, strategies of precision retrieval had to be designed, in order to perform the final position adjustment during this critical step. It was done using force control, coupled with a dedicated design of gripper tips. Finally, an autonomous parameters optimisation was used to gain the final millimeters of precision, and ensure the reliability of the insertion.

1.5.1.3. Software value chain and resulting process

The following diagram details the computing hardware and software architecture used for the CEA demonstration.

Regarding overall complexity, it should be noted that no particular effort was made to reduce the number of computers. Vision and skills could probably be hosted on the same machine. VR is not mandatory for this application, since there is not much geometric data to teach.

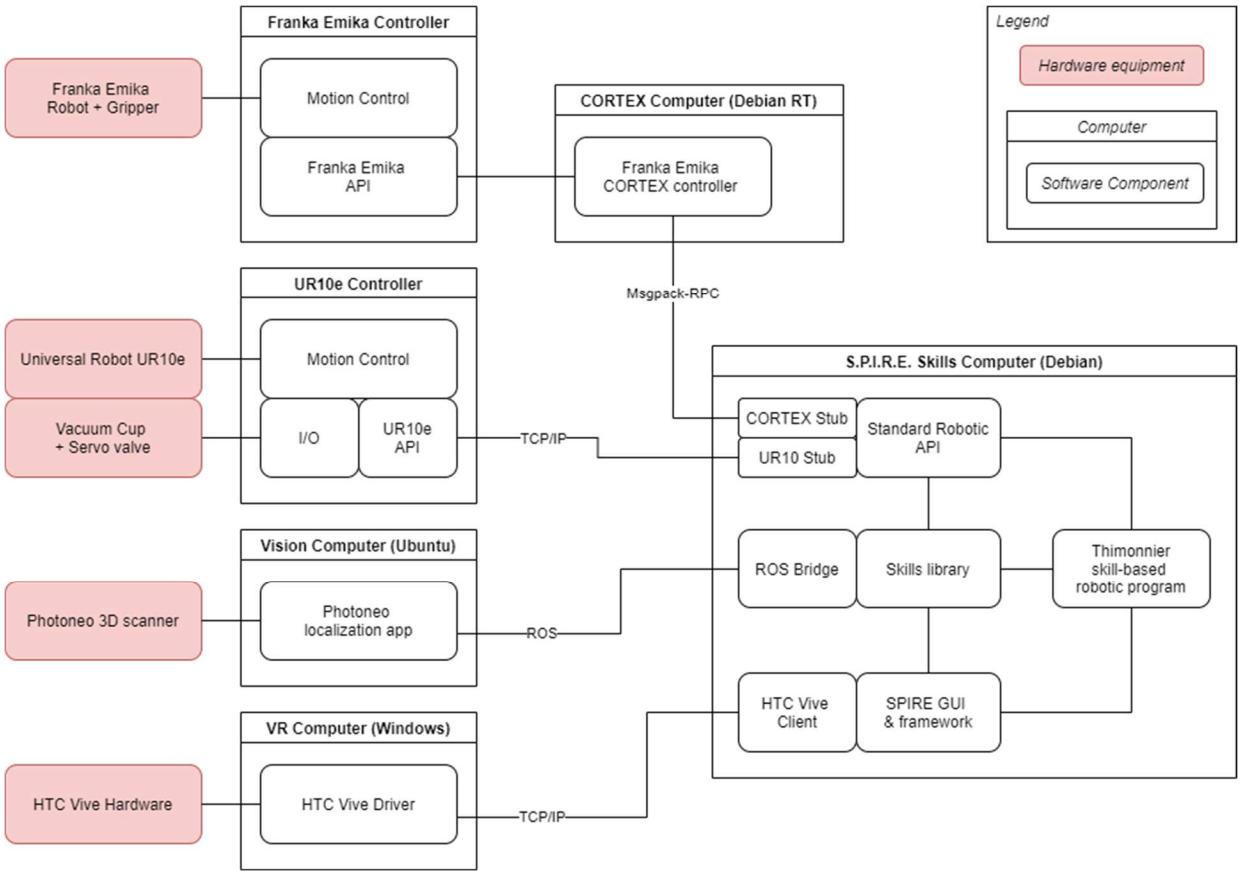


Figure 31 - CEA software architecture for the Thimonnier usecase

The vision is done using existing CEA LIST 3D localization software, that is expected to be replaced by a more robust localization module from AIMEN/WP5, using the same Photoneo hardware.

The CORTEX controller is a CEA LIST real-time robot controller that is used to encapsulate the Franka-Emika native controller, in order to provide the desired hybrid force-position control strategies and API. It allows us to force control the Franka Emika, with features such as force-controlled rail insertion, force threshold detection for skill success/failure detection. There is no CORTEX controller on top of the UR10e, because in this usecase, the UR10e is used for simple position control, and the native UR10e API is sufficient for this.

The following diagram illustrates the resulting process. Parallel execution of the three main elements of the setup allow for cycle time optimization. The highest-level composition was done with scripting

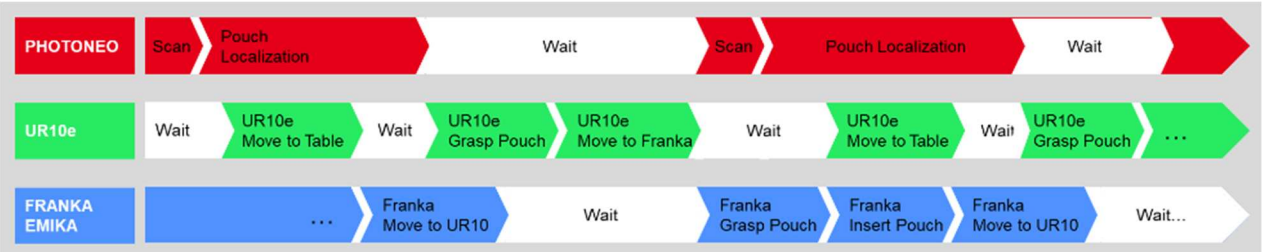


Figure 32 - CEA process timeline for the Thimonnier use-case

1.5.2. Strategy

In order to improve the cycle time, most of the steps are parallelized between both robots.

The insertion strategy uses two main components : custom gripper tips and force control. After the initial approach, the robot moves toward the rail while staying compliant in relevant directions. Once a certain force threshold is detected, the skills considers that contact with the rail has been established (it otherwise returns a failure signal after a timeout). A hybrid force-position translation along the rail follows, while still slightly pressing against the rail to ensure proper insertion. During this step, an excessive amount of force applied on the pouch indicates a jamming. The final check is performed by instructing the robot to apply force down the rail, and verifying that the rail does indeed apply a force on the end-effector in return. The robot then performs a set of operations in order to start over the cycle.

1.5.3. Optimisation

For this experiment, focus has been placed on the insertion of the pouch into the rail. While the task is critical, failures are easy to detect and correct.

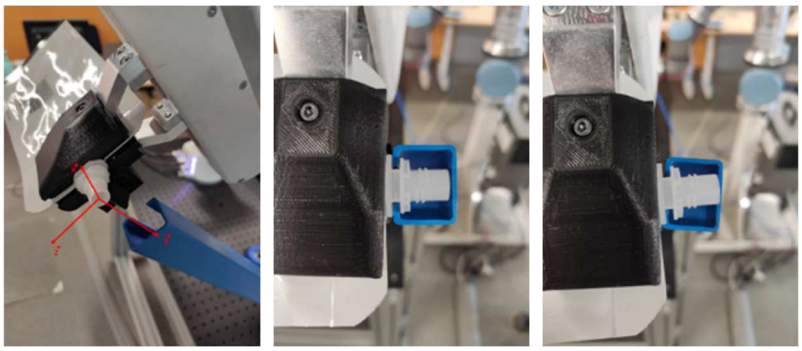


Figure 33 - a) Insertion setup, b) successful insertion, c) failed insertion due to the rail twisting

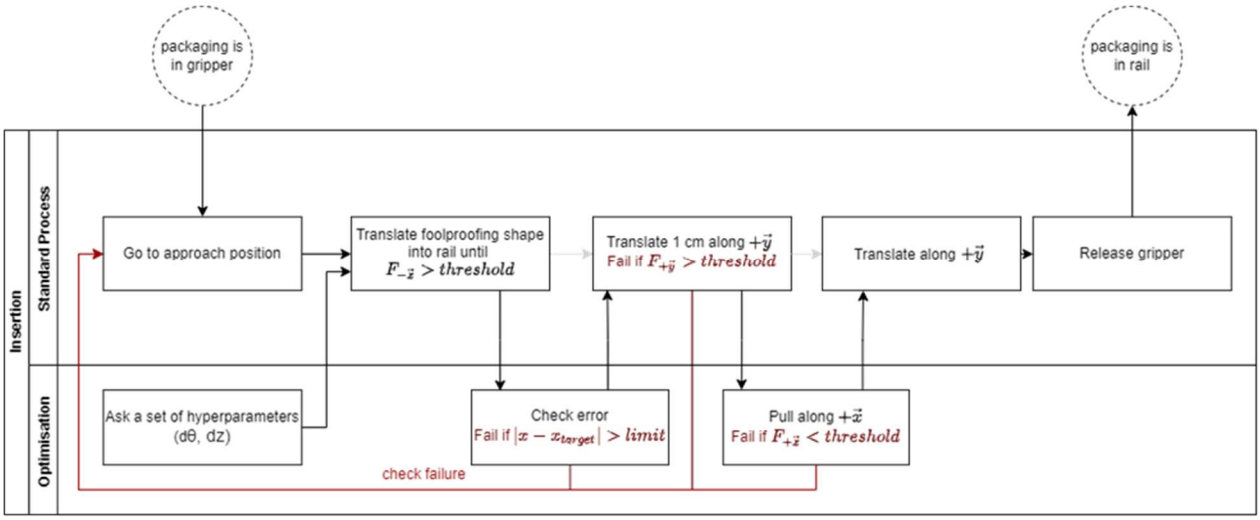
While the fool proofing shape on the side of the gripper's jaws help the centering, we define an angle dθ (around the z-axis) between the x-axis and the approach direction, as well as an offset dz on the z-axis, to ensure a smooth entrance. The rail is more flexible than the robot, and an off-center entrance may cause a misalignment of the groove and the rail despite a successful translation. The hypermeters to optimize are the approach direction **dθ**, which we will seek between 0˚ and 45˚ and the offset **dz** between -5mm and + 5mm.

The following figure presents a high-level overview of the insertion process and its optimization phase, including how failures are detected at each step. The set of hyperparameters for each trial is obtained from a CmaES sampler, and the most successful results are aggregated to obtain the final set.

We define three landmarks during insertion, defined by the checks in red in the graph above: contact with the rail, translation into rail, and successful insertion. The reward function for this task is a weighted sum for the criteria "number of landmark successfully completed" and "mean measured force during insertion", as we seek to minimize effort for a smoother entrance:

$$F(x) = 50 * n_{landmarks\_completed} - 1 * Force_{mean}$$

Our two hyperparameters dz and dθ are not independent and both serve the same purpose of optimizing success rate and minimize efforts. They have to be optimized together in a single sequence, which is possible since the search space dimension (two) stay low. Since no categorical hyperparameters are considered, CMAES is the more effective sampler.

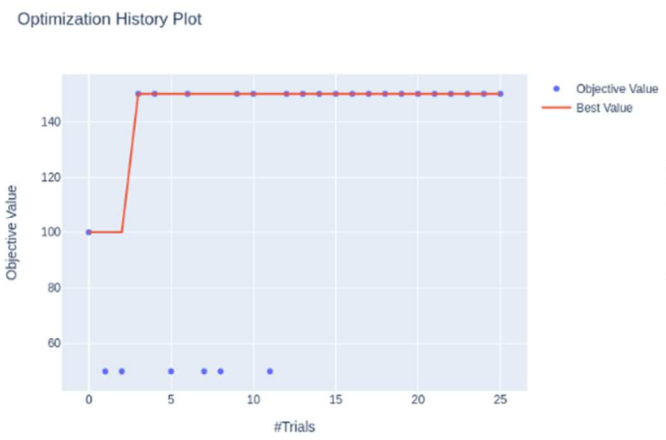


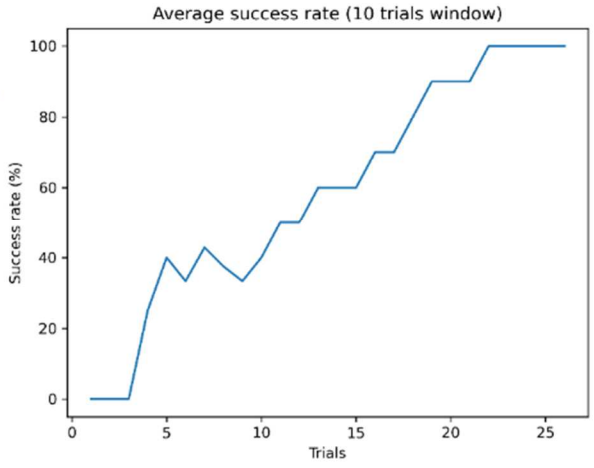
Optimisation stops after fifteen completely successful insertions. To avoid noise and false positives when choosing the optimal set of hyperparameters, we cluster the successful trials, choose the cluster with the highest mean reward, and select its centroid as our optimum.

The figure below shows the results of one experimental run of the optimisation sequence. In this particular situation, the very first trial is a partial success, only the first two landmark are successful: the system manage to make contact with the rail and translate along it, but the pouch remain outside the rail and is not successfully inserted. We then experience a mixture of complete failure (no contact) and complete success. After the twelfth trials, all attempts are complexly successful insertion.

Convergence is attained in less than a couple dozen trials. It is what was expected for system a of two hyperparameters and low precision requirements thanks to the proofing shape on the gripper's jaw, according to the simulation shown earlier.

The time gained is already distinctly noticeable against manual hyperparameters optimisation. While the system can complete twenty cycles, alone, in a few minutes, doing so manually, by changing the hyperparameters, checking success, measuring offsets iteratively can easily take an operator thirty minutes.





1.5.4. Experiment conclusions

This experiment demonstrated most of the overall workflow of the framework, as well as the convenience of MoCap demonstrations. All developed skills are now available off-the-shelf within SPIRE for quick reuse, and their code can be easily modified to fit more specific or demanding tasks.

1.6. Application on the Selmark use-case

1.6.1. Experimental setup

In this experiment, teleoperation (MoCap remote control) is used to capture a set of poses and trajectories for manipulating a sheet of fabric and laying it up, with two arms simultaneously. These primitives are then used as skills parameters. Without requiring manual editing of these demonstrations, the robotic cell is able to realize the task (grasping, transport, and lay up). A general speed parameter was introduced in order to reduce the cycle time (mostly by speeding up demonstrated trajectories).

Figure 34 - Selmark experimental setup, using bi-manual MoCap teleoperation for gesture teaching

This experiment demonstrated the potential of the abstracted robotic API layer, as skills were executed on different robots and grippers without requiring specific code. Moreover, it validates the adequacy of motion capture teleoperation for bi-manual manipulation. The user was able to teach an entire synchronized gesture. The teleoperation aspect enabled the operator to check the state of the ply, and keep both robots in configurations that wouldn't damage the object, as well as taking into account its deformable behaviour during the lay up stage. Finally, it demonstrated that SPIRE enables fast and intuitive programming of complex operations.

1.6.2. Experiment conclusions

The Selmark prototype validated the following features :

- Hardware interoperability, by using the same skills (teleoperation and motion) with two different robots and grippers.
- Teleoperation, in its bi-manual version.
- Primitives processing and management, for automatically smoothing captured trajectories, and adapting them between waypoints.
- Basic parallelization for synchronized motion.

1.7. Conclusion

During the Merging project, we implemented and tested a comprehensive software suite dedicated to easier programming by demonstration. This suite is build around three main capabilities:

- Skill-based intuitive programming
- Motion-capture teaching modalities (offline teaching and teleoperation teaching)
- Learning-based parameters optimization, for automatic skill configuration and fine-tuning.

A considerable effort has been undertaken to provide user ergonomy, and to ensure practical useability of the developed features. For this, we followed a workflow-centric design, and developed all necessary graphical user interfaces, along with various user assistance features to further facilitate teaching by demonstration.

The first experiments on the Thimonnier and Selmark usecase provided a very valuable and positive feedback on the effective value of the SPIRE software suite. The two motion-capture teaching modalities, with their associated user assistance features, provide a very smooth way to teach points, trajectories, and expert gestures. The skill paradigm, associated to the robotic API abstraction, provide a convenient and powerful programming experience.

Overall, a first qualitative analysis of the KPIs show a very positive potential for real-world applications, with a definite impact on teaching and programming time and facility. Quantitative analysis of KPIs will be done in WP8.

The main observed limitations are the lack of precision of the overall system (mainly due to the limited performance of both the HTC Vive MoCap system and the Franka Emika robot), the needed time for manual optimization of parameters, and the overall limited speed of execution, compared to manual task realization. The parameters optimization toolbox proved to be a very effective solution to the first two limitations, and can also help about cycle time optimization.

Future work will be focused on implementation refining – the current framework is still a prototype; WP6 Digital Twin coupling; and skills implementation, with the goal to build incrementally multiple trade-specific libraries of skills. It will start with the development of the required skills for the integration phase planned in WP7/WP8.

2. On-line robot control law for the control of robot arm under human-machine interaction

This part focuses on the co-manipulation of large objects, with relation to the VDL use case.

The final goal of the VDL use case is to define and reconfigure the lay-up process without programming and robotics expertise. In this regard, CEA LIST and LMS have worked on a co-manipulation system for the transportation and the precise positioning of large parts in a mould.

The interest is to develop a flexible tool that enables:

- to experiment a new process of fabrication of bus composite panel (Modify easily the process during the fabrication tests and verify all the steps with the operator)
- to handle a new model of panel without the intervention of robotics expert during the production
- a step towards automation

CEA distinguished two classes of large parts in the lay-up process due to their different characteristics that have impact on the co-manipulation task:

- Fabrics: the main characteristic is the deformation under the co-manipulation and the lack of force transfer ;
- Foam: the main characteristic is the fragility under the co-manipulation. A force can be transferred but has to be minimized to not broken the preform under constraints.

2.1. Large fabrics co-manipulation

This section is dedicated to the co-manipulation of large fabric parts, such as the various fabrics used in the VDL usecase.

2.1.1. Introduction

The goal of this sub-task is to allow a human operator to co-manipulate large deformable parts, like large pieces of fabrics, together with a robot. Most of the features for co-manipulating fragile parts are reusable for deformable parts. Nevertheless, the important deformation of these latest doesn't allow transmission of force informations between one extremity of the part to the other.

In this part, we developed two strategies of control:

- Data from a vision system and inertial measurement (acceleration and angular velocity) are fused to feed a position controller. We use operator's hand position, grasping forces and, if available the measure of the object deformation, and task context, to control the robot.
- Data from only inertial measurement (acceleration and angular velocity) feeds a velocity controller

In order to generate these data, we designed an instrumented glove that captures hands' movement, fingers' position and force feedback on touch.

2.1.2. Glove/Computer Vision fusion controller

2.1.2.1. Controller design principle

For this approach, we need the absolute position of the operator with relation to the robot base frame. These informations are vision-based and will come from the WP5 outputs, but in this demonstration, a Kinect Azure V2 has been used. Absolute position informations and IMU informations are then merged within a Kalman filter in order to obtain an input for the co-manipulation algorithm.

The fusion algorithm is a modified Kalman filter, based on the algorithm in [Alatise,Hancke2017]. The idea is to mainly use the vision-based position and correct with the IMU if needed i.e when occlusions occur.

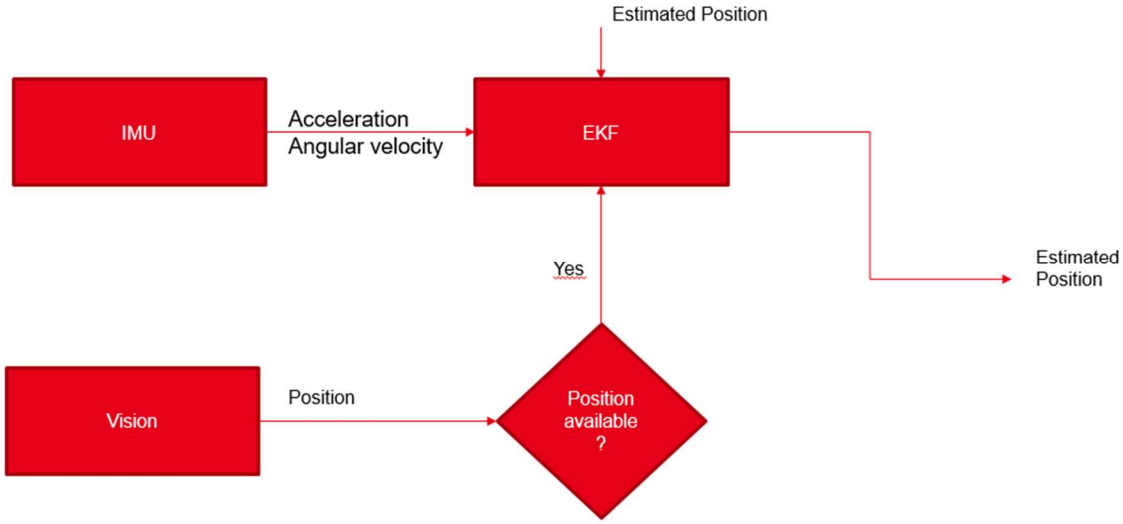


Figure 35 Fusion algorithm

When occlusion occurs (in the frame), the algorithm computes the position using only the IMU data. It switches to a fused position when the vision data is back.

The force and flex data are used to control the gripper and detect operator's intention.

The control point is expressed in the cartesian space. It means that it can be easily associated to one manipulator or to several synchronised manipulators with a common servoing on that Cartesian control point [1].

2.1.2.2. Experimental demonstration

a. Setup

Glove

As no force transmission is possible between extremities of the deformable part, we needed a co-manipulation strategy that is not based on force control. Our approach is to measure information of the operator's hand behavior by using a tactile glove and then process the robot's action. This tactile glove is equipped with an IMU in order to measure the orientation of the operator's hand and to help pros the position. Nevertheless, due to the large drift on IMU position estimation, it is necessary to use also an absolute position estimation for recalibration. For that purpose, several solutions could be used, for example depth cameras, stereo cameras, or other trackers.

For the Merging project, a simple tactile glove was designed and used. This glove is equipped with five tactile/force sensors, five flex sensors, three IMU (one on the hand, one on the wrist, one on the tip of the thumb), an MPU with integrated Bluetooth communication. One of the tactile sensors is used to detect user's intention to couple/decouple the robot.

For real-time control, the data throughput of IMUs and sensors data is at 100 Hz. The glove is able to stay on for more than 6 hours in normal mode with a small LiPo (Lithium polymer) battery. Manipulating while wearing this glove is relatively the same as wearing normal protection gloves.

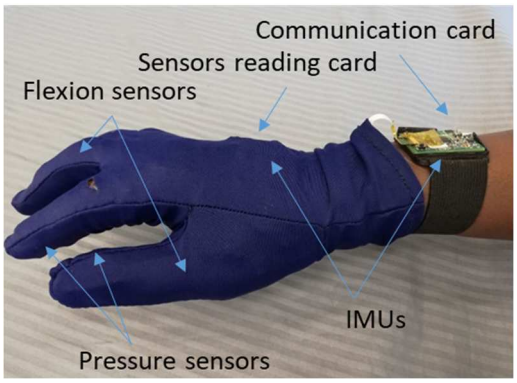
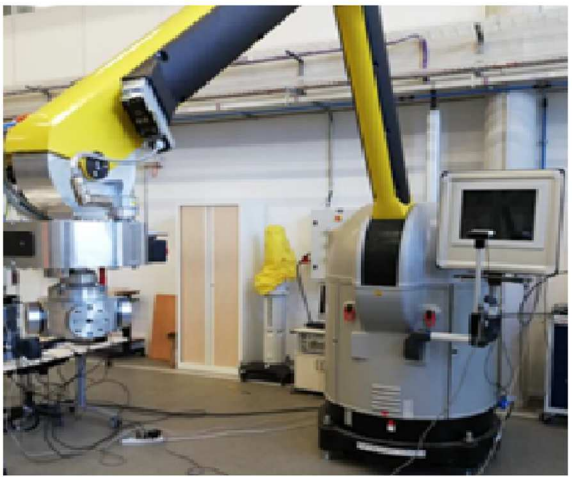


Figure 36 - Glove prototype

Figure 37 - Industrial version of the glove

Cobomanip robot



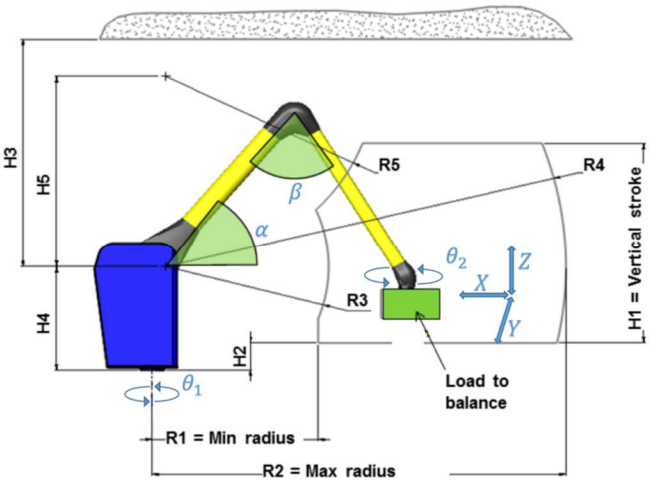


Figure 38 Cobomanip robot

H1	H2	H3	H4	H5
2100 mm	283 mm	2390 mm	1085 mm	2000 mm
R1	R2	R3	R4	R5
1580 mm	3810 mm	1550 mm	3680 mm	2000 mm

Table 1 Workspace data of the Cobomanip

The Cobomanip is a collaborative robot designed to have very low friction and high back-drivability. Using this manipulator will reduce the force on the fabrics. The kinematic architecture of the Cobomanip is described in *Figure 38 Cobomanip* . In its standard version it consists in a cranelike structure with 3 or 4 degrees of freedom (dof) depending on the initial configuration. The first body rotates around a vertical axis represented by θ1. The 2nd and 3rd bodies rotate around two horizontal axes and, represented by α and β angles. Upon request, thanks to parallelogram structure maintaining the end effector to a fixed orientation, a 4th vertical axis located at the end effector may be added (rotation represented by θ2).

The cobomanip has a wide workspace array (almost 4meters max radius, over 2 meters radius range) that is best suited for the VDL case when manipulating large pieces (larger than 1000x1000 mm).

b. Results

In this demonstration, we validated the position control of the robot. The robot will always try to keep the fabrics straight meaning the fourth axis (rotation θ2) keeps facing the operator.

Figure 39 Testing the position control loop

This test has allowed us to validate that the robot could follow the position of the operator's hand. In nominal mode (Vision position fused with IMU data), we reach a positioning accuracy which is in the centimetre order of magnitude. However, when occlusions occur, the accuracy falls to 10 cm.

We did a second test in a configuration that close to the VDL use-case. The operator standing on a point A lifts the fabrics and moves to a point B, 2.5 meters away from point A. Then, the operator places the fabric on the mould.

When handling the fabric, the lower part of the operator's body is hidden. This does not count as an occlusion because it is still possible to retrieve the body pose from the vision system.

Figure 40 Co-manipulation of the fabric standing on point A

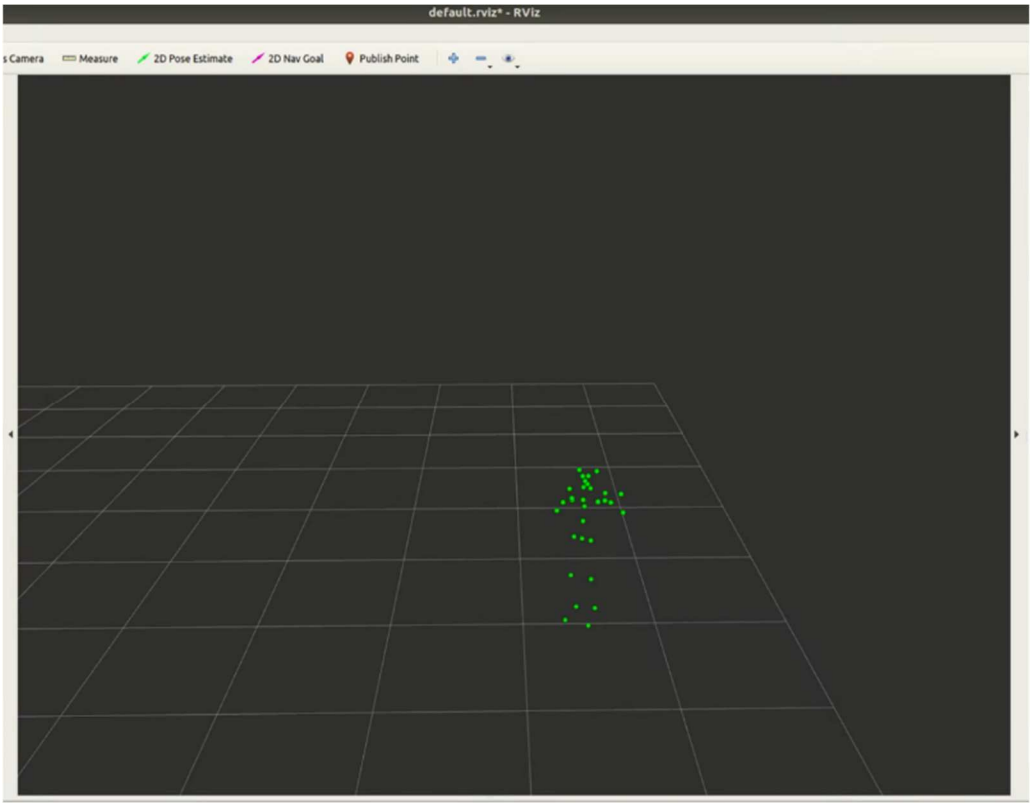


Figure 41 Vision data display (point A)

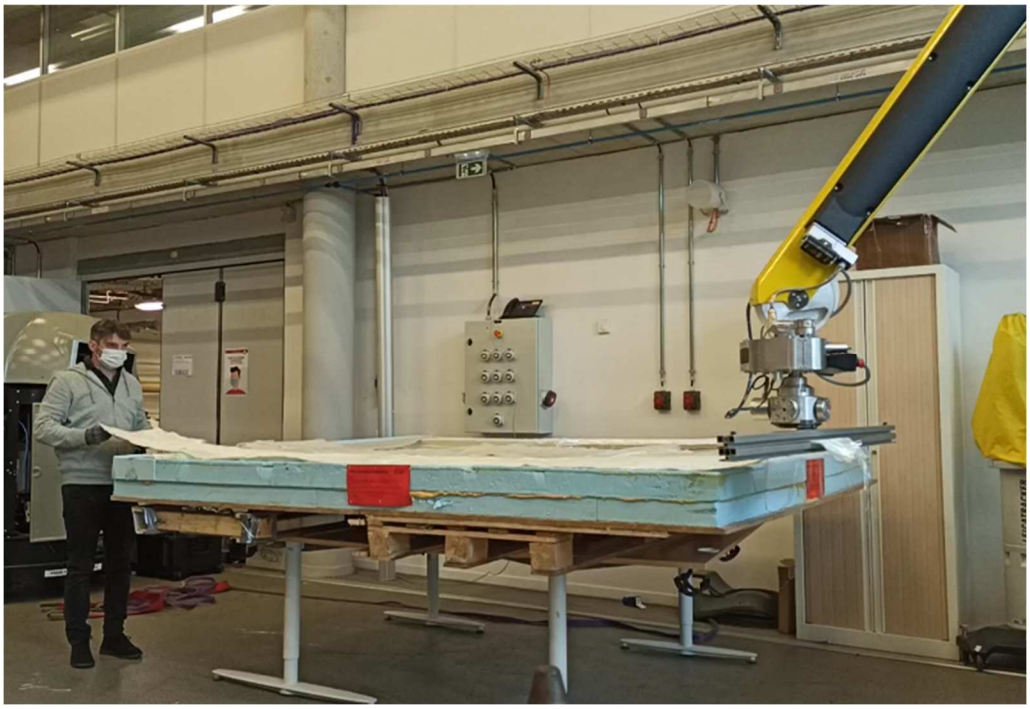


Figure 42 Co-manipulation of the fabric standing on point B

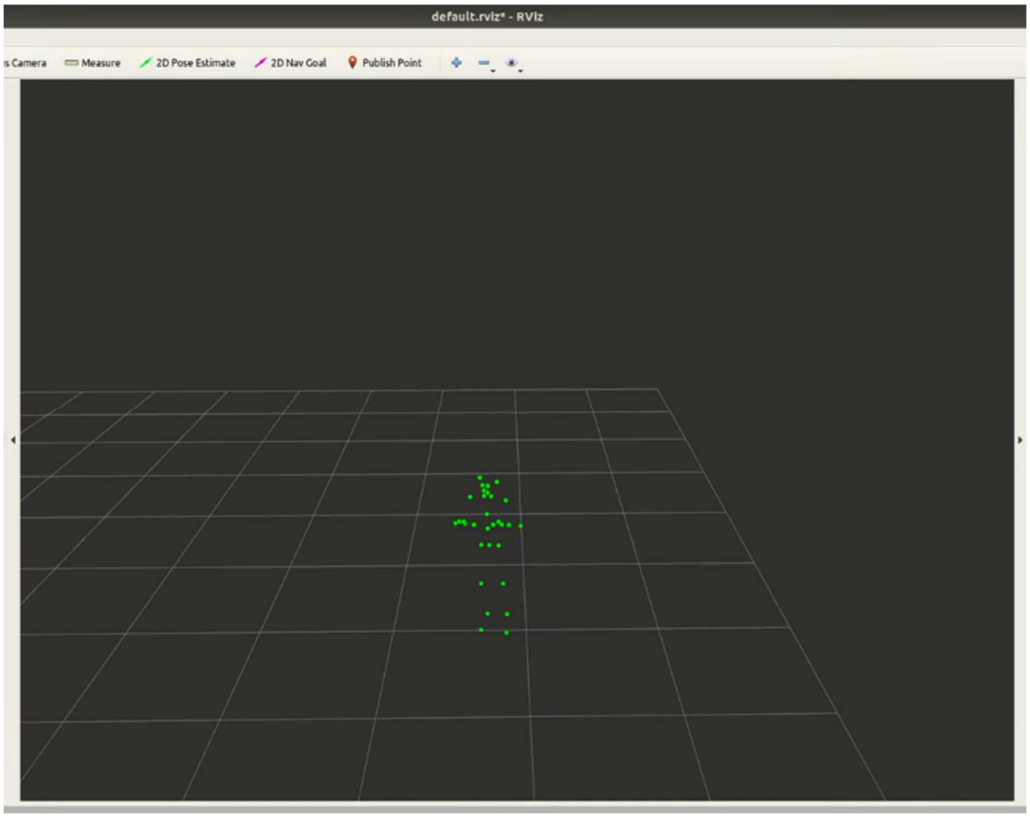


Figure 43 Vision data display (point B)

The robot followed the movement of the operator and they could place the fabric on the mould. We concluded that the positioning accuracy is sufficient. The movement is smooth but a bit slow (maximum 10 cm/s) because the Cobomanip has low velocity limits.

The limitations of this controller is that, we cannot easily keep the fabric spread. As the robot is mirroring the operators' behaviour when coupled, the distance between the operator and the robot is constant (relative to the accuracy). The operator has to decouple the robot and spread the fabrics by stepping back. Doing this is not always convenient, so during the integration phase, we will use the information from the perception system in WP5 to adjust automatically the position of the robot.

When a fused position is not available, the controller computes a position from the IMU only. This state cannot be kept for more than 5 seconds, with a notable loss of precision. This motivated us to find a more robust strategy, based on the Glove stand-alone library.

2.1.3. Glove stand-alone library

2.1.3.1. Controller design principle

In the previous section, a teleoperation mode in position using computer vision has been presented. In this previous mode, one needs to have a quite robust estimation of the operator's hands position. If we want to be resilient, we should handle situations when, due to the observation distance and occlusion occurrences, ambient cameras (see WP5) hand localization outputs are not available for a relatively long period.

In order to prevent this eventuality, in this section, an alternative control mode will be presented using only the data glove (or an equivalent sensor).

The main logic is to avoid the need of the precise estimation of the operator's hand by using a velocity control mode (instead of the position in the previous control mode). Again, in order to improve the robustness, only the rotation measures will be used in this control law. The advantage of rotation measures is that the algorithm that computes them recalibrates with earth's gravity vector to remove the drift.

As it is a co-manipulation system, the control loop is closed by the operator, seeing and adjusting the displacement of the robot. In this case, the operator pilots the robot as if he/she is using joysticks. The fine positioning is realised by the operator him/herself. If a pre-recording of the trajectory or a CAD model is available, we could implement a virtual guide that helps the operator fine position the fabric (see 2).

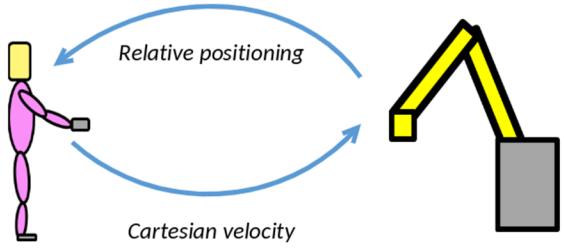


Figure 44 : Co-manipulation principle with glove standalone library

Then, the glove stand-alone library has been built, as intuitive as possible, in order to send Cartesian velocities using only the glove orientation measures (pressure measure will be used in the automaton, but also during the integration, for other types of controls i.e. modifying a virtual constraint). Another feature is that the operator can remotely control the one or more robots without having any piece of fabric in hand.

A description of the Glove Stand-alone Library is given in the following table. It is not an exhaustive description of this library as other control modes can be added, and will be added, if needed, during the integration phase. For example, the rotations around robot X and Y (cf. frame description on *Figure 45*) are locked for the moment, but can be added if needed.

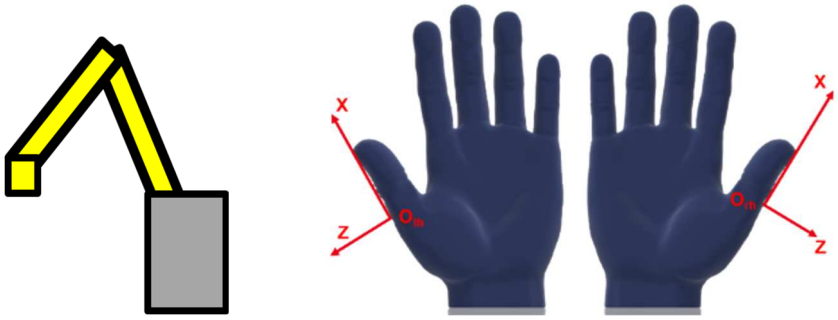
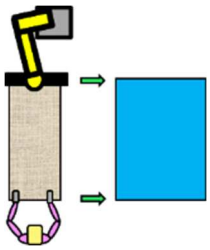
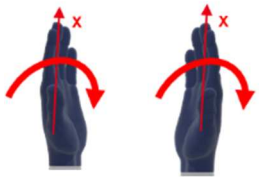
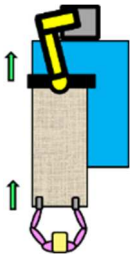
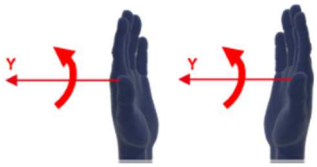
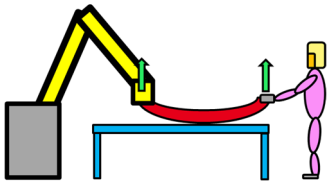
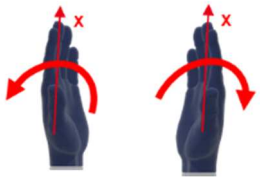
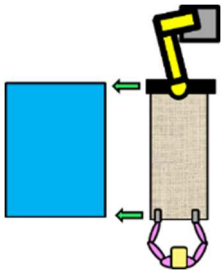
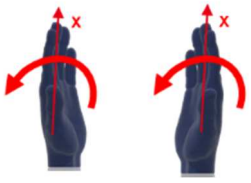
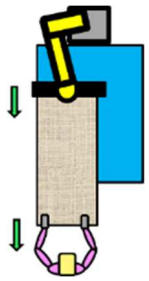
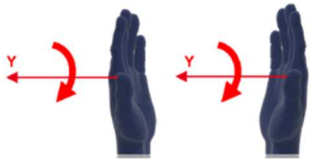
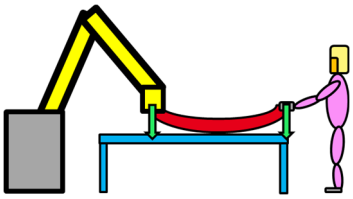
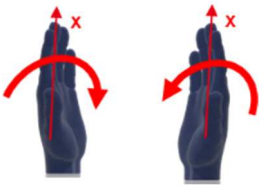
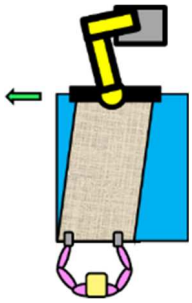
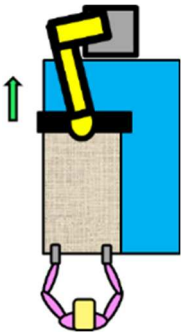
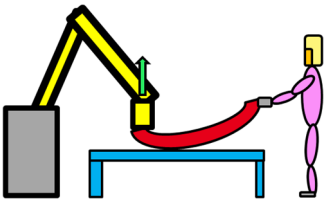


Figure 45: Robot and hands framed description

Displacement representation	Hand langage	Hand language schematic
	**ST1 right:** **Rot(Olh, X)>0 and Rot(Orh, X)>0** ST1 left: Rot(Olh, X)<0 and Rot(Orh, X)<0	
	**ST2 forward:** **Rot(Olh, Y)>0 and Rot(Orh, Y)>0** ST2 back: Rot(Olh, Y)<0 and Rot(Orh, Y)<0	
	**ST3 up:** **Rot(Olh, X)<0 and Rot(Orh, X)>0** ST3 down: Rot(Olh, X)>0 and Rot(Orh, X)<0	
	ST1 right: Rot(Olh, X)>0 and Rot(Orh, X)>0 **ST1 left:** **Rot(Olh, X)<0 and Rot(Orh, X)<0**	

Displacement representation	Hand langage	Hand language schematic
	ST2 forward: Rot(Olh, Y)>0 and Rot(Orh, Y)>0 **ST2 back:** **Rot(Olh, Y)<0 and Rot(Orh, Y)<0**	
	ST3 up: Rot(Olh, X)<0 and Rot(Orh, X)>0 **ST3 down:** **Rot(Olh, X)>0 and Rot(Orh, X)<0**	
	UT1 right: Rot(Olh, X)>0 and Rot(Orh, X)>0 **UT1 left:** **Rot(Olh, X)<0 and Rot(Orh, X)<0**	**Cf. ST1 left without movement of the operator** **Same thing for** UT1 right and ST1 right
	**UT2 forward:** **Rot(Olh, Y)>0 and Rot(Orh, Y)>0** UT2 back: Rot(Olh, Y)<0 and Rot(Orh, Y)<0	**Cf. ST2 forward without movement of the operator** **Same thing for** UT2 back and ST2 back
	**UT3 up:** **Rot(Olh, X)<0 and Rot(Orh, X)>0** UT3 down: Rot(Olh, X)>0 and Rot(Orh, X)<0	**Cf. ST3 up without movement of the operator** **Same thing for** UT3 down and ST3 down

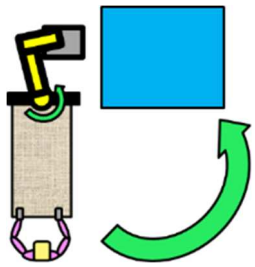
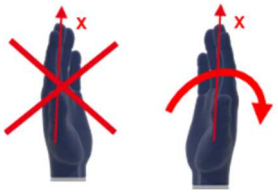
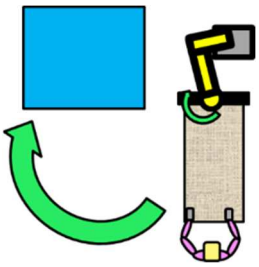
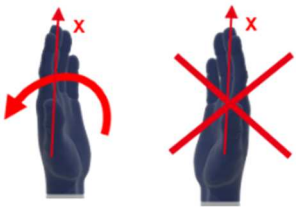
Displacement representation	Hand langage	Hand language schematic
	**SR1 right:** **Rot(Olh, X)=0 and Rot(Orh, X)>0** SR1 left: Rot(Olh, X)<0 and Rot(Orh, X)=0	
	SR1 right: Rot(Olh, X)=0 and Rot(Orh, X)>0 **SR1 left:** **Rot(Olh, X)<0 and Rot(Orh, X)=0**	

Table 2 Glove Stand-alone Library description: Synchronized Translations are noted STi, and Unsynchronized Translations UTi.

2.1.3.2. Experimental demonstration

a. Setup

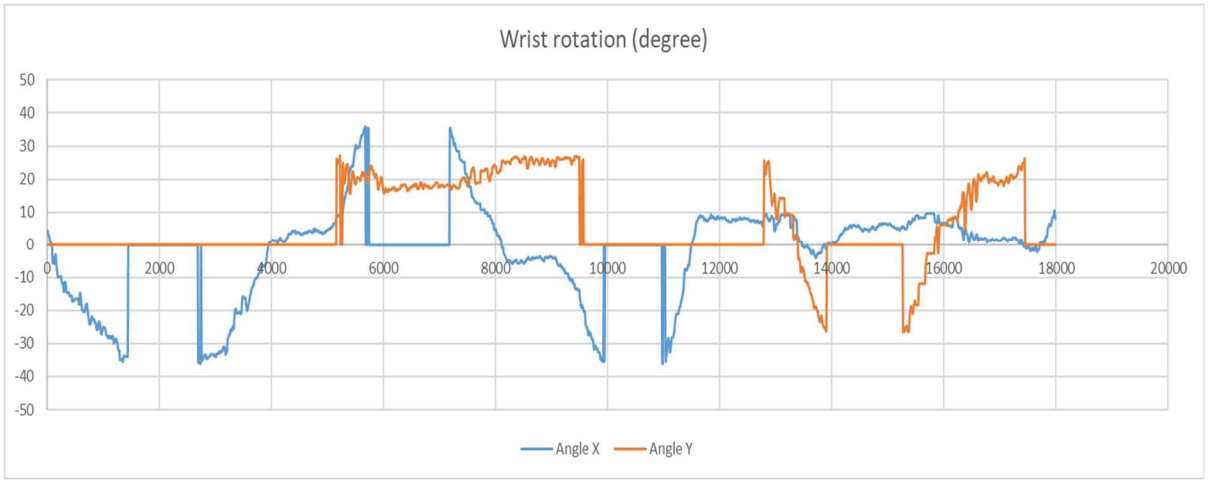
The setup is the same as the previous setup but the robot runs with a velocity controller and there is not any camera.

b. Results

We tested Right-Left, Forward-Back, Up-Down translations and Right-Left rotation around the robot's fourth axis. The velocity amplitude follows a linear progression as the operator increases his hands' rotation : $Velocity = Gain * Angle$. Angle is normalized between -0.5 and 0.5 radian (-45 and 45 degree) corresponding to human wrist-arm's degree of freedom angular range. Here, we set the gain at 10 which means that the velocity amplitude is 10 cm/s.

This test shows the robots response to the operator's gesture. Every case in the library is independent but it is possible to combine the movement. Here, the accuracy depends on the operator's skill and the robot's dynamic capabilities.

This controller allows the user to control with and without the fabric in hand. With this feature, we easily solved the issue with spreading the fabric. The operator can move the effector while standing still.



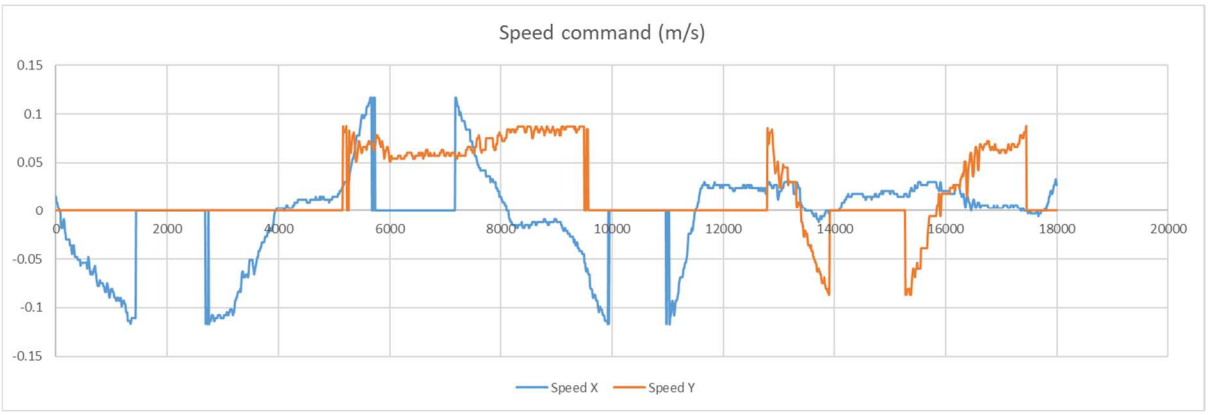
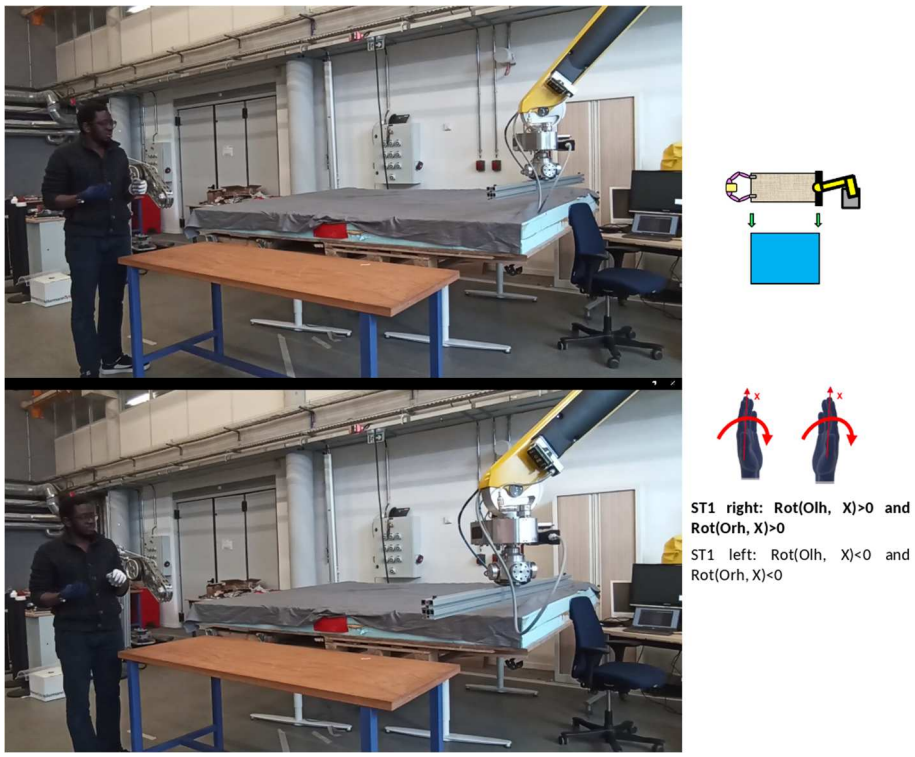
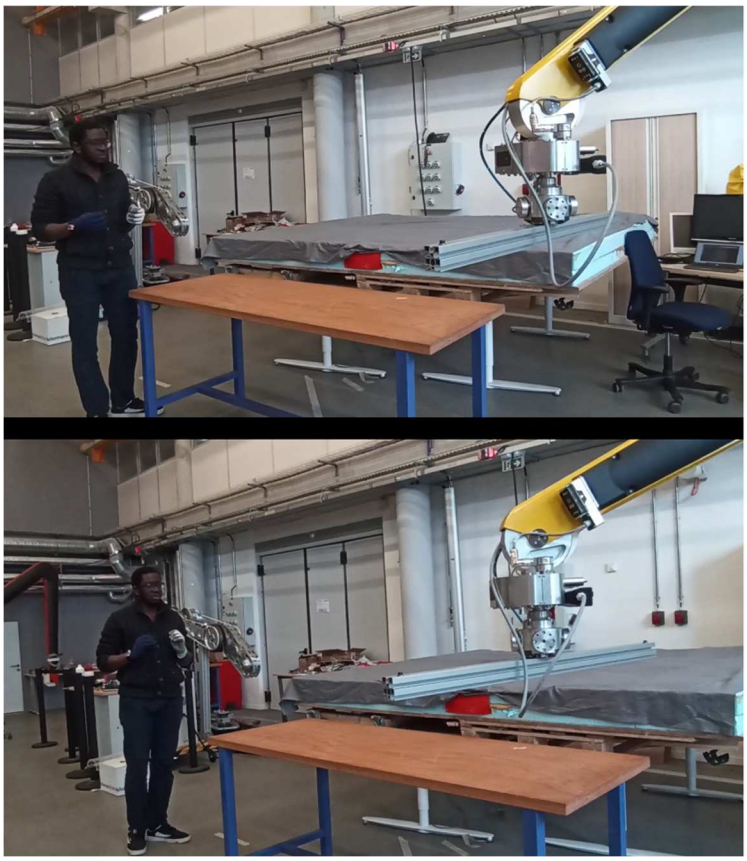
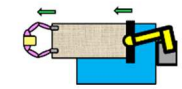
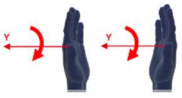


Figure 46 Wrist rotation and speed command



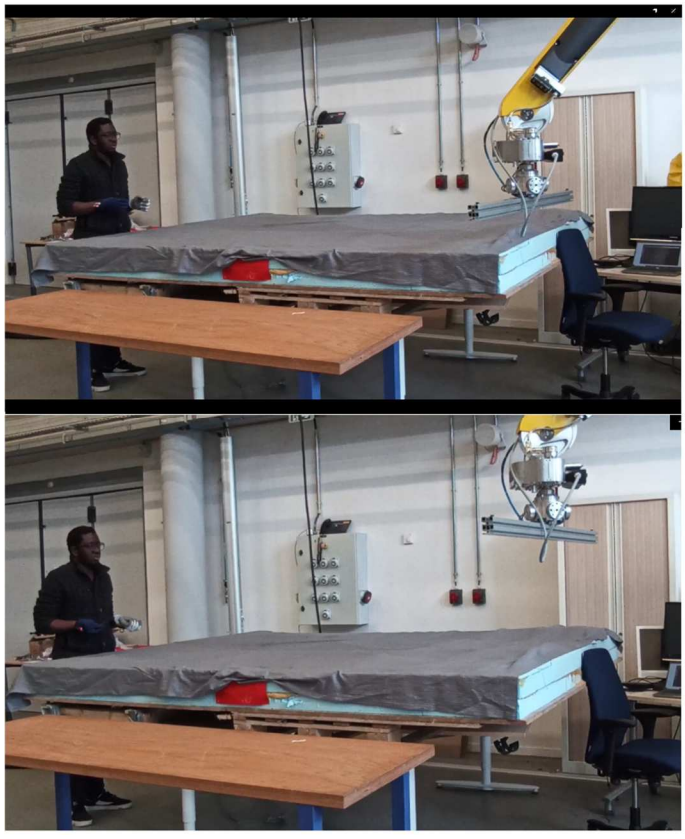


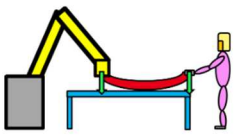


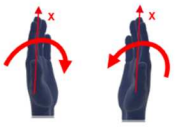


ST2 forward: Rot(Olh, Y)>0 and Rot(Orh, Y)>0

ST2 back: Rot(Olh, Y)<0 and Rot(Orh, Y)<0

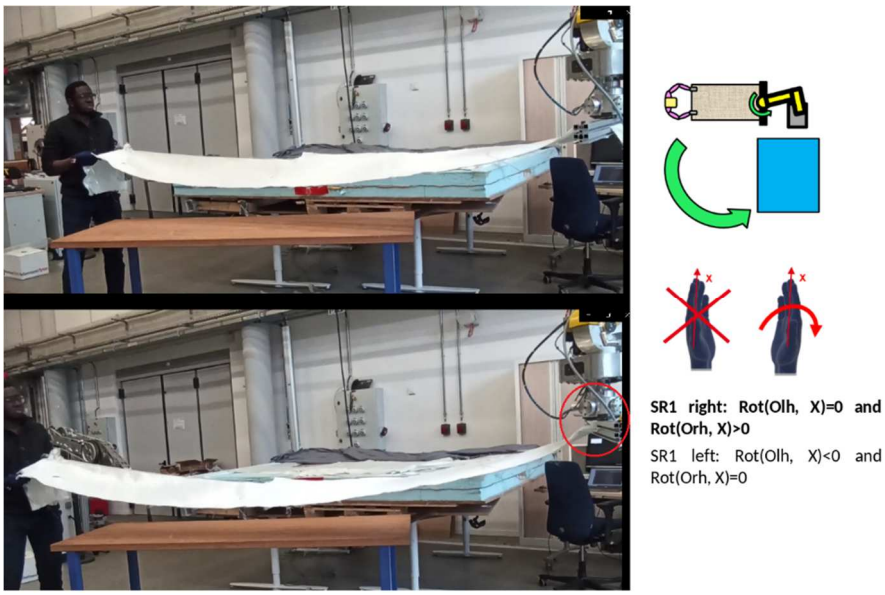






ST3 up: Rot(Olh, X)<0 and Rot(Orh, X)>0

ST3 down: Rot(Olh, X)>0 and Rot(Orh, X)<0



2.1.4. Conclusion

We designed two complimentary controllers' strategies: Glove/Computer Vision fusion and Glove stand-alone library.

They offer many advantages while positioning large and deformable fabrics:

- Intuitive and robust control
- Cheap and light hardware
- Full robot's trajectory setting

Those experiments being simplified versions of the VDL use-case requirements, improvements will be implemented as required during the integration phase.

The system Glove – Controller is reusable in others co-manipulation tasks, and also teleoperation tasks.

2.2. Foam blocks co-manipulation

This section is dedicated to the co-manipulation of large fragile parts, such as the foam blocks used in the VDL use case.

Foam co-manipulation is split into two steps:

1. Transportation step, for quick displacement and rough positioning of the part
2. Precise positioning step

2.2.1. Introduction

Large and fragile object co-manipulation tasks are extensively performed in industrial contexts, for instance in aerospace industry as well as in construction field. Our work has been driven by VDL use-case of the project: the manufacture of composite parts for the electrical bus production. The manufacturing process of composite parts requires the transport of large and fragile parts like foam blocks into a mould where they are then precisely positioned.

The main features - large and fragile - of the manipulated object lead to involve several operators for executing the task and often cause ergonomics issues. Operators sometimes use industrial assisting systems but these systems suffer from a lack of flexibility.

In this regard, large and fragile objects co-manipulation task has resulted in much research and development effort in the robotics field, and in particular in the human-robot joint collaboration domain.

As discussed in D4.1, the main challenge is to enable simultaneous motions of the operator and the robot gripping points without damaging the large and fragile object and by preserving operator from applying high efforts.

In D4.1, we described two new robotic controllers that have been designed during MERGING project:

- a controller of a robotic assistant for the co-transport of large and fragile objects ;
- a controller of a robotic assistant for the precise positioning of large and fragile objects.

with the aim to make the co-manipulation tasks of large and fragile objects more reliable by:

- having both partners that apply only forces (no torques) as a human-human dyad usually does;
- allowing any kind of motions;
- ensuring an easy predictability of the robot behaviour for the human partner.

In the following sections, we will present the functionalities of these two controllers through experimental demonstration of elementary use from operator's side.

2.2.2. Functionalities for the co-transport of large and fragile objects

We demonstrate the functionalities of our new controller of a robotic assistant to realize the transportation of large and fragile objects (for instance into the mould for VDL use-case) jointly with an operator.

2.2.2.1. Controller design principle

The system consists in designing an active follower robotic partner. The human operator is the leader of the dyad in order to handle the displacement of the object along the transport trajectory. The robot is the follower in order to avoid heavy stress on the fragile part due to a plan disagreement but it actively performs its gripping point motions in order to prevent operator from applying high torques.

To do this, we propose to:

- Allow rotations at the robot gripping point with a zero-force mode that prevent from robot torques application.
- Correct the object orientation, which corresponds to an angle α at the robot gripping point, by a displacement of the robot gripping point along the transport trajectory (*Figure 47*).

Our approach requires a partial knowledge of the task plan. At least the targeted point has to be known.

This condition can easily be fulfilled in an industrial context, for instance with a vision module or with programming-by-demonstration. In the VDL use-case application, an approximate position of the mould obtained from vision component of WP5 is enough.

A definition of the trajectory by the digital twin, which is developed in WP6 for instance, is also a possible input for our controller.

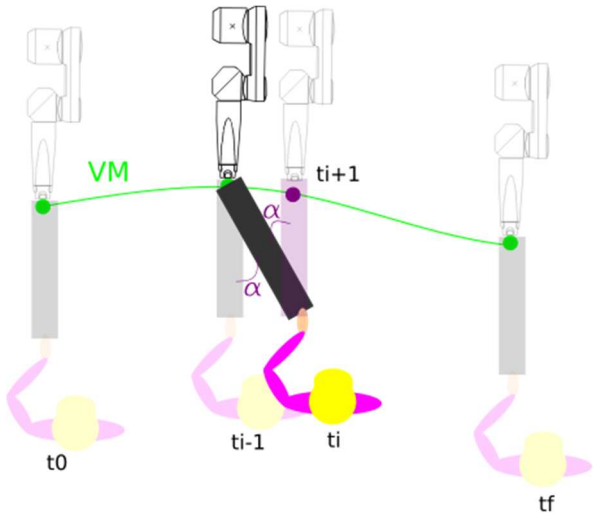


Figure 47 - Principle of the co-manipulation strategy

The block diagram of the controller is reminded in next figure.

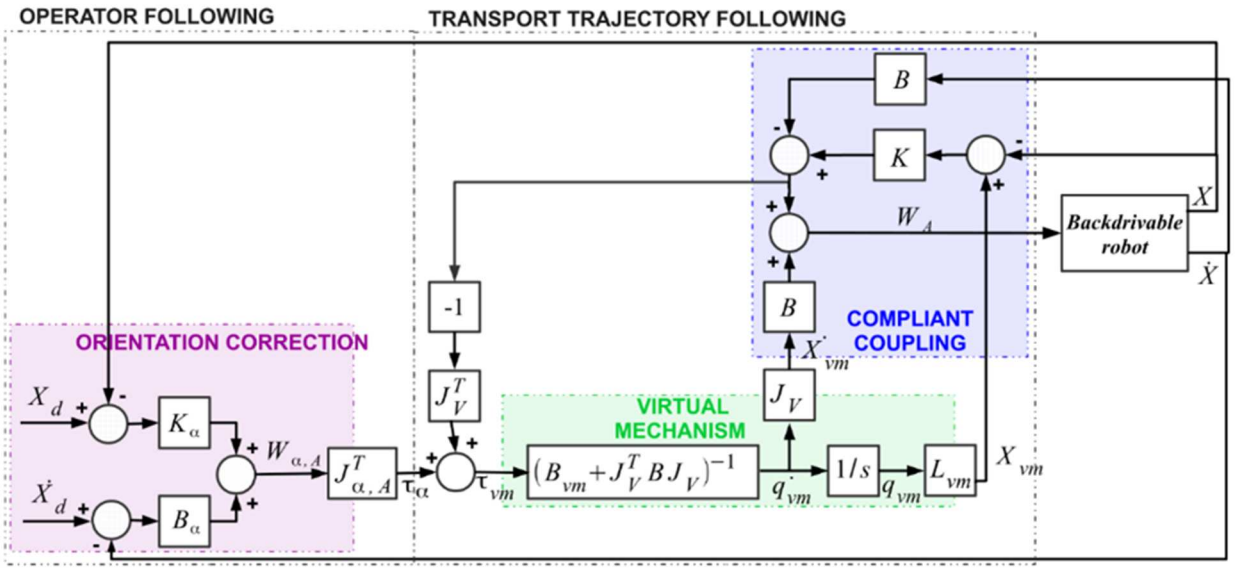


Figure 48 - Block diagram of the control system

2.2.2.2. Experimental demonstration

a. Objective

We carried out a pilot experiment in order to evaluate the efficiency of our approach for transporting large and fragile objects.

Moreover, in order to highlight the benefits of our extension of [Sanchez2020] to co-transport large and fragile objects, we compare the two methods on a joint collaborative transport task.
Both of these methods allow assisting the path following with a given orientation along the path.
The work in [Sanchez2020] proposes a passive assistant corresponding to a backdrivable robot coupled with a virtual mechanism described with the XSpline framework.

In the following section, our approach is called “active assistant” and the method proposed in [Sanchez2020] is called “passive assistant”.

b. Setup

We lead the experiment on a 4-DOF articulated arm manipulator (next figure).

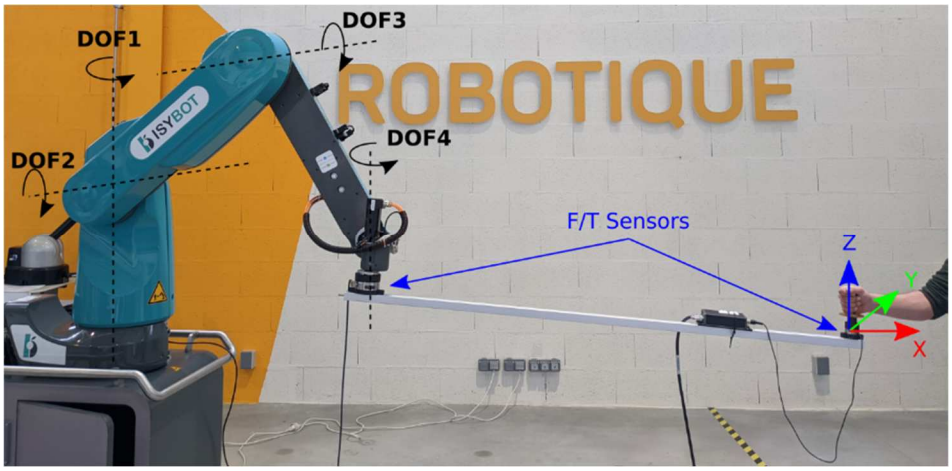


Figure 49 - Experimental setup

However, it is important to notice that the controller presented in D4.1 is working for co-transport tasks in 6-DOF. The choice of the robot has been made only according to the availability of the arm manipulator in the lab.

This manipulator is mechanically backdrivable. Otherwise, it is necessary to implement an impedance or an admittance control at the lower level.

The transported object is a bar of 1.8 meters long. The end-effector of the robot is rigidly attached to one end of the bar while an operator’s handle is fixed on the other end.

In order to observe wrenches applied on each gripping point, one 6-axis Force/Torque sensor has been mounted between the wrist of the robot and the bar and another 6-axis Force/Torque sensor has been mounted between the bar and the operator’s handle.

Wrenches were recorded at 1kHz.

Knowing that the stiffness and damping gains ($\boldsymbol{K}$, $\boldsymbol{B}$) of the robot/VM coupling are independent of the defined VM, we use the same gains for both methods. They have been set empirically in order to feel a sufficient attractive force that enables to stay on the guide.

We also empirically tune the stiffness and damping gains ($\boldsymbol{K}_{\alpha}$, $\boldsymbol{B}_{\alpha}$) applied for the wrist’s angle correction in order to obtain a comfortable following speed of the operator by the robot.

Although several users have tested our approach during and after its development, only one subject participates to the experimental confrontation reported in this section. This participant does not contribute to the development of either of the two methods.

The participant was trained to perform different transport tasks with both methods before recording the data.

The participant taught four waypoints by demonstration (Figure 50 - The four waypoints taught by demonstration with the computed orientation along the path) in order to define the transport task.

The desired orientation was computed in order to keep the object orientation orthogonal to the constraining path (*Figure 50*).

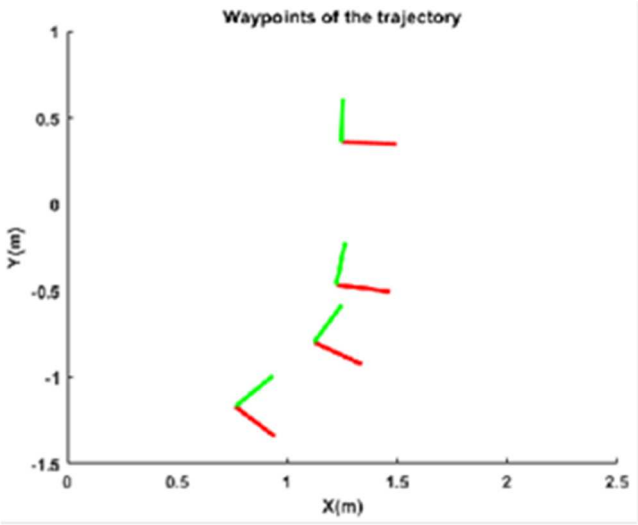


Figure 50 - The four waypoints taught by demonstration with the computed orientation along the path

Between points, we used a spherical cubic interpolation of quaternions (SQUADs) [Sanchez2020].

The same trajectory was used for the object transport with both methods.

The operator executed the defined transport task for the first time with the active assistance, then with the passive assistance.

c. Results

With a 4-DOF robotics arm, the controller is active only on given directions. For this reason, we present only the components of wrench that are relevant.

The wrench applied at the operator gripping point during the transport task with both methods is presented *Figure 51*. The wrench applied at the robot gripping point during the transport task with both methods is presented *Figure 52*. Both of these methods allow to prevent the operator from applying high torques as observed on *Figure 51*.

As expected, we observe a significant torque (with a peak of 12.99N.m) at the robot gripping point with the passive assistance while a significantly weaker torque (with a peak of 3.26N.m namely almost 4 times weaker) is observed with the active assistance (Figure 52). The torque observed with the active assistance is mainly due to non-compensated friction and inertia of the fourth robot joint.

Moreover, we observe a significant force on X-direction at both gripping points (with a peak of 69.64N) with the passive assistance while weaker force (with a peak of 11.54N namely almost 7 times weaker) is observed with the active assistance (Figure 51). An explanation coming from discussions with the participant is the intuitiveness of the active assistance compared to the passive assistance. For the active assistance, the operator gives only the intention of displacement and the robot actively follows the path by itself. With the passive assistance, the operator has to apply forces in order to move the robot on the trajectory. However, the transport trajectory is not visible. Even if the operator has a rough idea of the trajectory, it is too difficult to apply precise direction of forces along the trajectory. Thus, the operator relies on the constraints to guide the movement along the trajectory by applying a force in the normal

direction of the path (X-direction of the F/T sensor). That confirms that this well-known virtual guide feature [Boy2003] is not compatible with a fragile object co-manipulation because it leads to apply significant stress on the object.

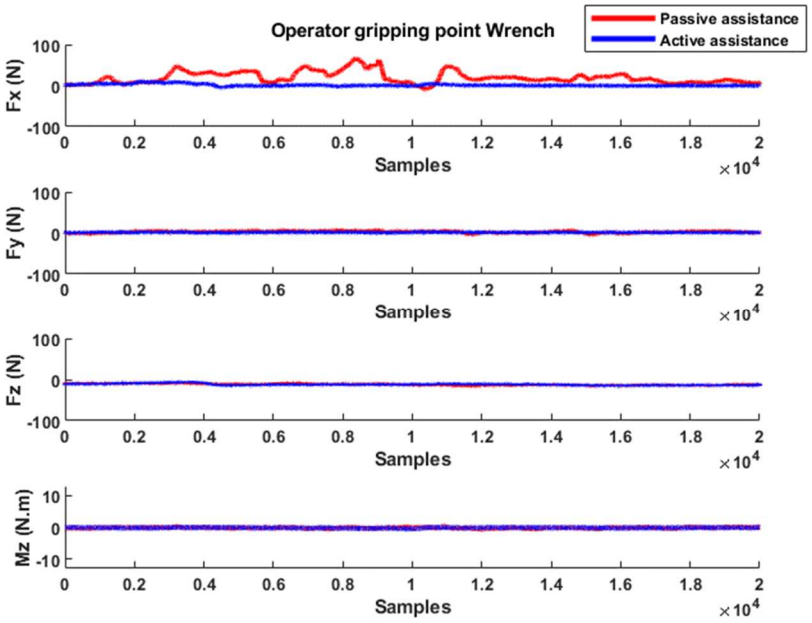


Figure 51 - Wrench at operator gripping point

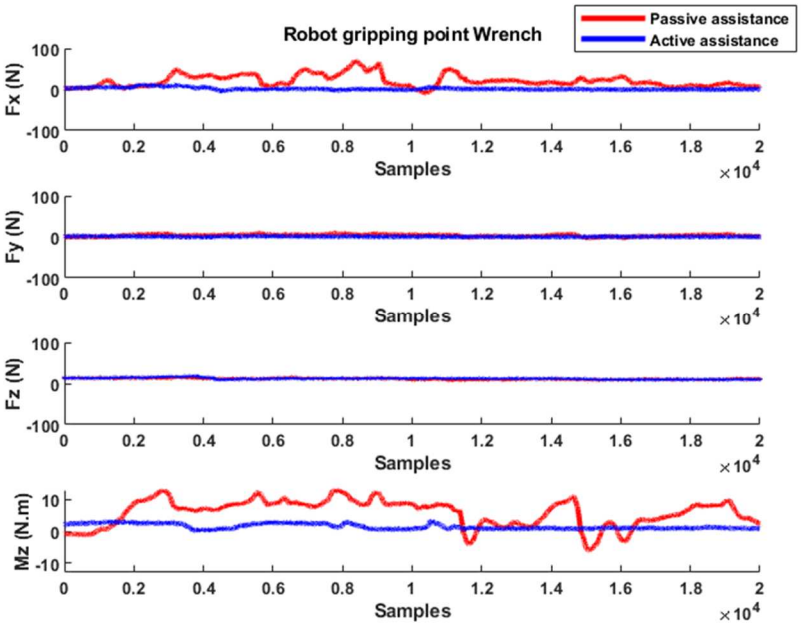


Figure 52 - Wrench at robot gripping point

2.2.2.3. Conclusion

We presented in this Deliverable the application of the proposed control law for a transport task of large and fragile objects. Moreover, we demonstrate, through a pilot experiment, the effectiveness of this

approach for transporting a large and fragile object by preventing torques application at robot and operator gripping points and more generally, by minimizing stress on the object that is a necessary condition for large and fragile objects transportation. The experimental confrontation also pointed out the intuitiveness of our active assistance compared to the passive assistance proposed in [Sanchez2020].

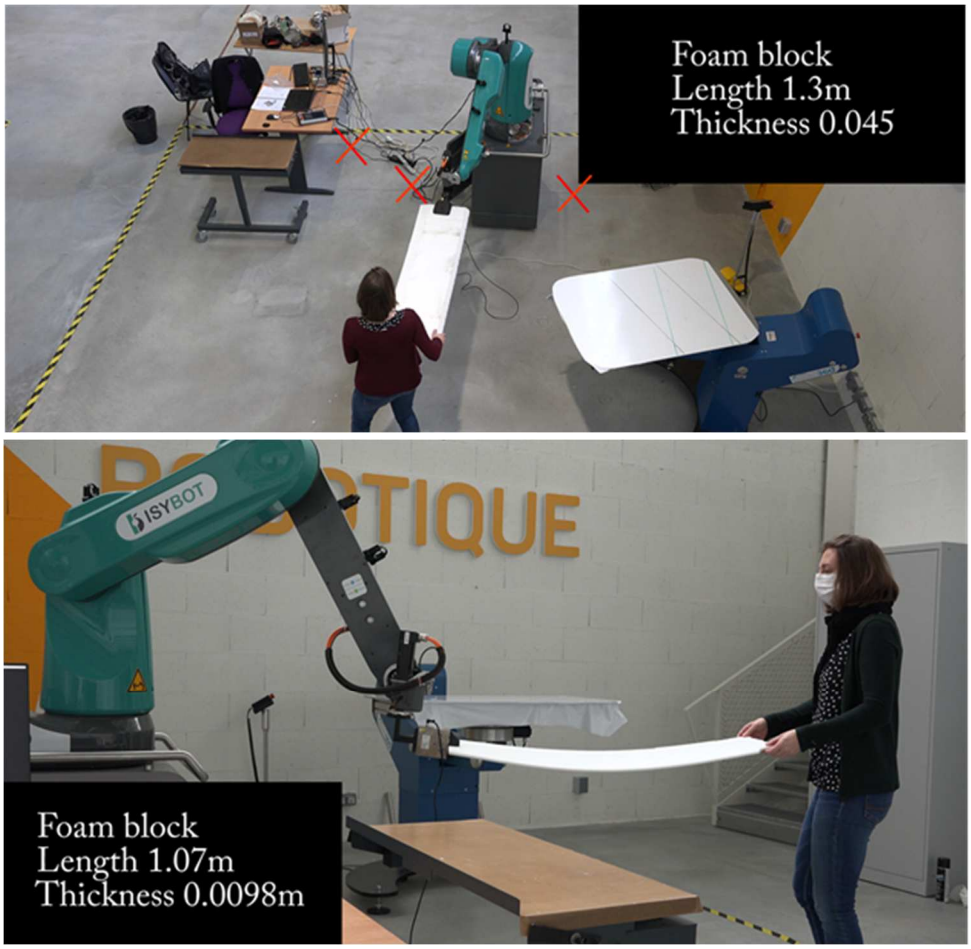
This work led to:

- a submission of an international conference paper: « Controller design of robotic assistant for the transport of large fragile part » IROS 2022 - accepted
- a patent pending : « Procédé de co-manipulation d'une pièce par un opérateur aidé par un partenaire robotique »

We qualitatively evaluate the application of this controller in order to cotransport objects with different thickness and length (*Figure 53*).

We qualitatively check the limitations of the controller application by trying to cotransport flexible objects as fibre textiles (*Figure 54*). In this case, we succeed the transport task only at low velocity. However, it would be interesting to evaluate this controller for textile co-manipulation with the final gripper of the MERGING project because textiles slip in our gripper.

To conclude, the proposed controller has been designed and presented to efficiently co-transport large and fragile objects. However, it appears to be also very interesting for co-manipulating large objects with high inertial robots.



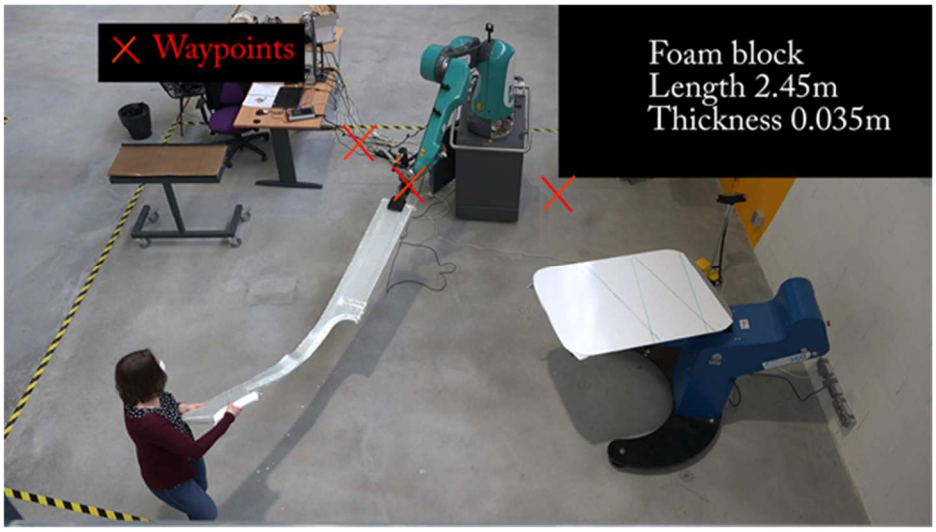


Figure 53. Co-manipulation of objects with different thickness and length

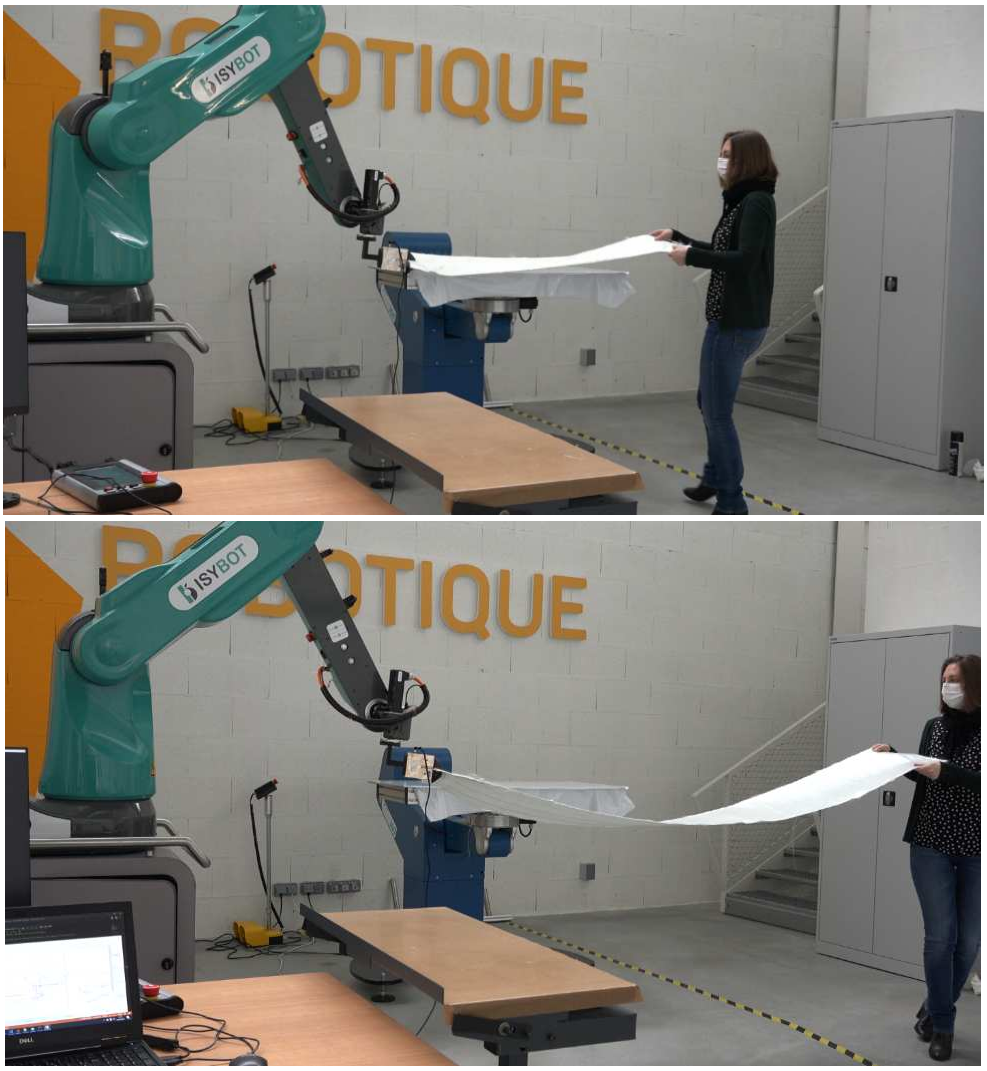


Figure 54. Co-manipulation of flexible objects with different length

2.2.3. Modification of the constraints

In order to deal with a wrong definition of the transport trajectory or with unexpected events during the transport, we developed an additional important feature presented in D4.1: to allow the operator to modify the transport trajectory with different modalities of modification:

- off-line modification of the trajectory that enables to deal, for instance, with a wrong definition of the transport trajectory;
- on-line modification of the trajectory that enables to face unexpected events that might occur during the transport.

In a complementary way, we implemented a module to visualize the virtual guide during the co-manipulation in augmented reality. We use a HTC Vive with a Zed-M camera. The robotic controller communicates the parametrization of the guide and the position of the robot to a physical engine developed by CEA [Weistroffer2022], which relies on Unity3D for the graphical display.

The off-line modification can be performed with any device that provide a position input, for instance, with a controller of HTC Vive, as well as with the digital twin information or by demonstration with the robot.

We present in this section the application of the online modification, which requires additional steps compared with offline modification (see D4.1).

We demonstrate the efficiency of avoiding an obstacle present on the guide during the co-manipulation.

STEP 0:

The initial virtual guide is displayed in green among the real scene (*Figure 55*).

Then, an obstacle is placed on the initial virtual guide (*Figure 56*).

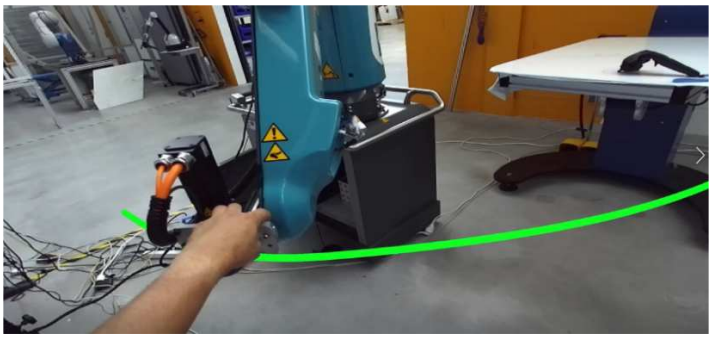


Figure 55. Display of the view of the operator. In green the virtual guide in the real scene

Figure 56. Obstacle on the initial guide

STEP 1:

The operator is able to escape the guide and to define a new portion of the trajectory to avoid the obstacle (Figure 57).

The new waypoints are recorded (*Figure 58*) and selected according to the method presented in D4.1.

Figure 57. The operator escapes the guide in order to avoid the obstacle

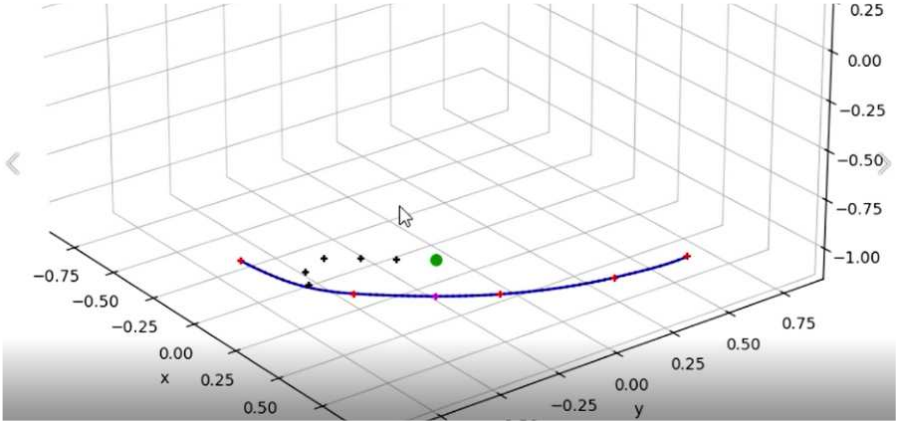


Figure 58. Plot of the initial guide and the new waypoints. In blue, the initial guide; in red, the initial waypoints; in black the new waypoints currently being recorded and in green the position of the robot.

STEP 2:

The new guide, combination of the initial guide and the new portion, is computed according to the method presented in D4.1. It enables the operator to avoid the obstacle (Figure 59).

Figure 59. Modified guide that enables to avoid the obstacle

This demonstration shows the efficiency of modifying online the virtual guide in order to face to unexpected events that might occur during the co-manipulation. This highlights the flexibility and intuitiveness of the method that does not require any programming and robotics expertise in order to define a specific trajectory.

These modules of guide modification and augmented reality display can be used in a largest context than the transport task.

2.2.4. Functionalities for the precise positioning of large and fragile objects

2.2.4.1. Controller design principle

We designed a new controller of a robotic assistant to realize the precise positioning of large objects (for instance in the mould for VDL use-case) jointly with an operator.

The collaborative mode for the precise positioning task is designed with a hybrid force/velocity controller, leading to a mix of teleoperation and cobotics modes.

We shared the control of the degrees of freedom into two modes in order to avoid the torques application on the fragile part and in order to ease the manipulation of a large object:

- A force mode allows the operator gripping point motions: the rotations of the robot's wrist are free in order to avoid any torque application at the robot's gripping point and in order to enable the operator to position easily his gripping point (*Figure 60*, in green). The sharing direction, which is the direction through both gripping points, can also be free because the force applied at the operator gripping point in this direction does not generate torque (*Figure 60*, in dark green).
- A velocity/position mode handles the robot gripping point motions: The two other directions of translation (*Figure 60*, in magenta). It avoids the torques application at the operator gripping point.

The operator uses an external device to generate the velocity inputs while the force inputs are directly the forces applied by the operator on the object.

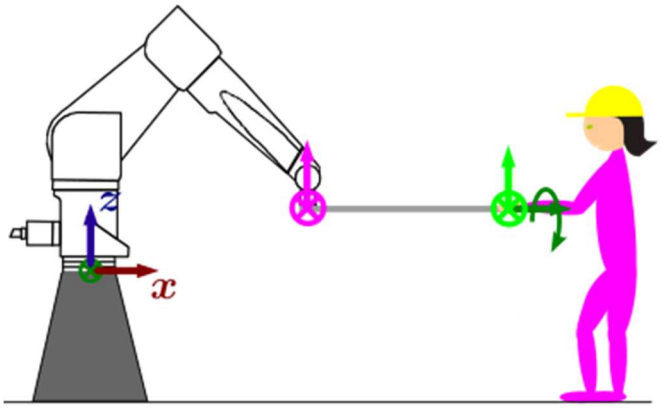


Figure 60. Shared control of the degrees of freedom between a force mode (cobotics mode) and a velocity mode (teleoperation mode). In green, the degrees of freedom that are piloted in a force mode. In magenta, the degrees of freedom that are piloted in a velocity mode with an external device. In dark green, the degrees of freedom that we can choose to control either in a force mode or in a velocity mode

The block diagram of the controller is reminded in *Figure 61*.

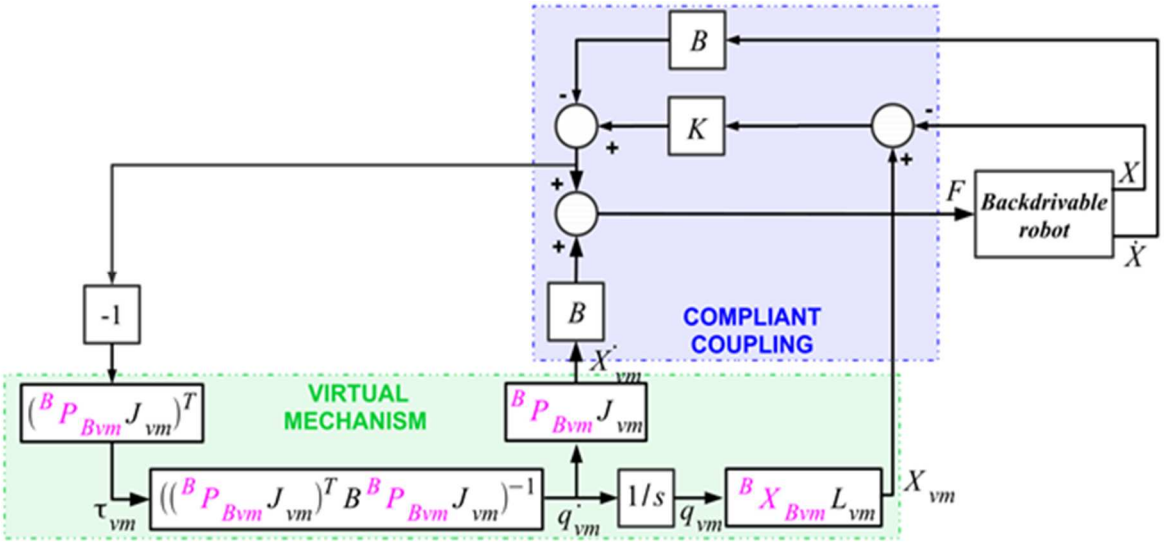


Figure 61. Block diagram of the control system.

2.2.4.2. Experimental demonstration

We lead the experiment on a 4-DOF articulated arm manipulator (*Figure 62*).

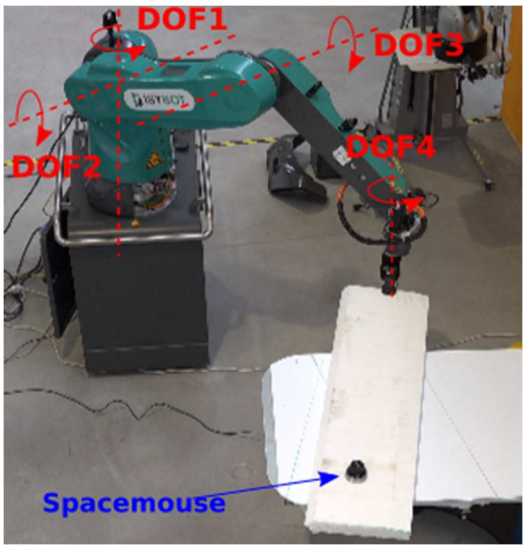


Figure 62. Experimental setup

However, it is important to notice that the controller presented in D4.1 is working for co-manipulation tasks in 6-DOF. The choice of the robot has been made only according to the availability of the arm manipulator in the lab.

This manipulator is mechanically backdrivable. Otherwise, it is necessary to implement an impedance or an admittance control at the lower level. A two parallel fingers gripper is attached at the wrist of the robot in order to grasp the objects.

We use a 3D-spacemouse (*Figure 62*) as the external device that enables to control the two translations of the robot gripping point (see previous section).

As the stiffness and damping gains ($\boldsymbol{K}$, $\boldsymbol{B}$) of the robot/VM coupling are independent of the defined VM, we use the same gains than the controller presented in Section 2.1.2.

We perform different demonstrations of precise positioning with different targeted orientations in order to show the capacity of the system to easily turn around any point of the object. Two sequences with opposite targeted orientations are presented in *Figure 63*. Targeted positions are represented by straight lines of different colors (black or green).

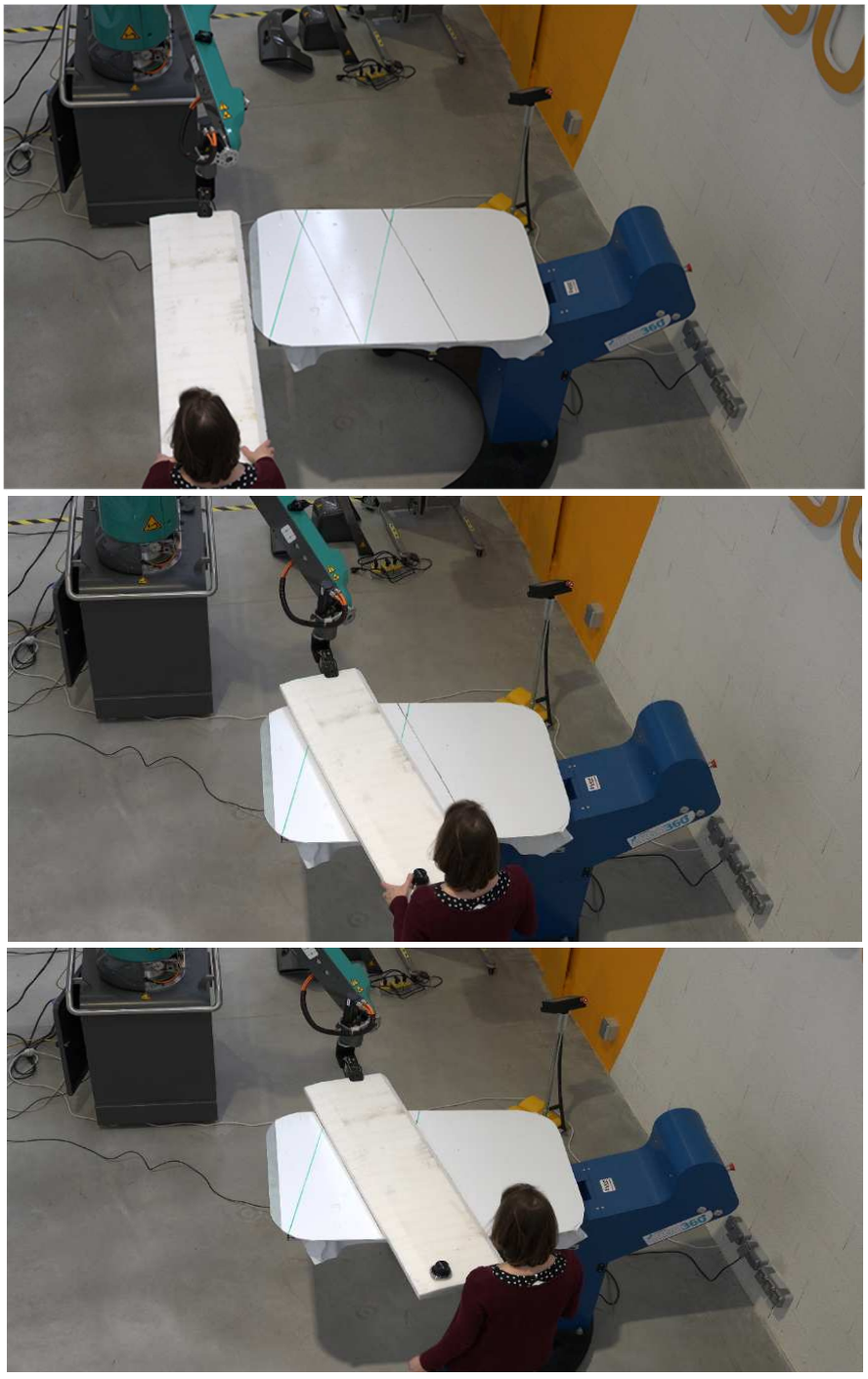


Figure 63. Sequence of the positioning of the large object in the black target

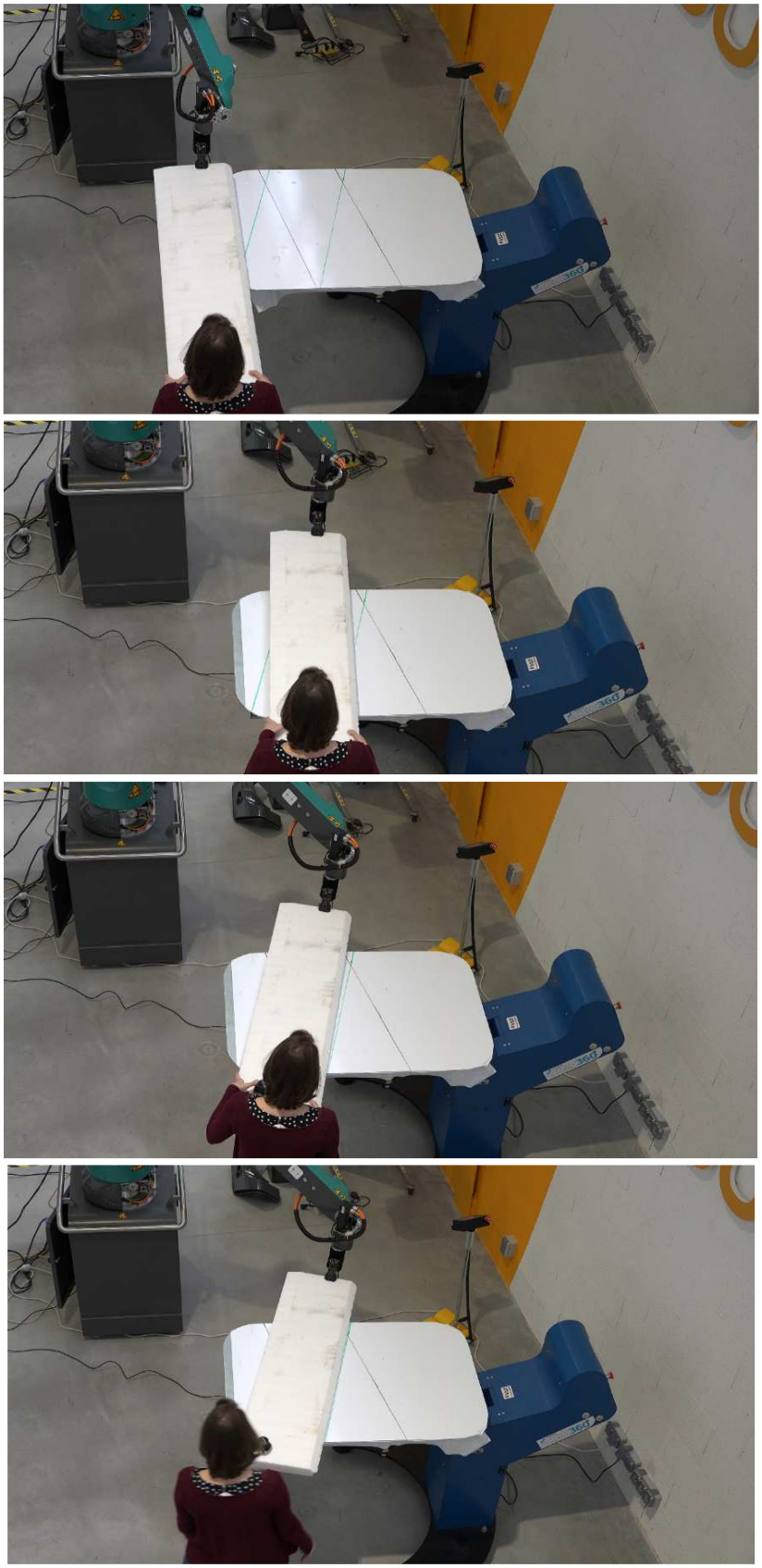


Figure 64. Sequence of the positioning of the large object in the green target

2.2.4.3. Conclusion

During the MERGING project, we designed a new controller for efficiently positioning a large and fragile object that offers many advantages in comparison with the state-of-the-art approaches:

- No pre-programming is required, including the final targeted position;
- Enable simultaneous and independent motions of the operator and the robot gripping points ;
- Enable holonomic motions (possibility to vary the rotation point of the object);
- Ensure an easy predictability of the robot behaviour for the human partner;
- Enable both partners to apply only forces (no torques) as human-human dyad usually does.

This system enables to precisely position large and fragile objects without damaging it and by preserving operator from applying high efforts. The quantitative evaluation of these performances with KPI will be presented in WP8 delivrable.

A patent pending process has been initiated on this work.

The same controller with a different parametrization of the virtual mechanism can also be used, for instance, to sweep a surface with a given force while controlling the trajectory in the plane of the surface.

Appendix A - Bibliography

[Boy2003] E. Boy, E. Burdet, C. Teo, and J. Colgate, “Motion guidance experiments with scooter cobot,” in 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2003, pp. 63 – 69.

[Sanchez2020] S. Sanchez Restrepo, G. Raiola, J. Guerry, E. D’Elia, X. Lamy, and D. Sidobre, “Toward an intuitive and iterative 6d virtual guide programming framework for assisted human–robot co-manipulation,” Robotica, vol. 38, no. 10, pp. 1778-1806, 2020.

[Weistroffer2022] V. Weistroffer, F. Keith, A. Bisiaux, C. Andriot, and A. Lasnier, “Using Physics-Based Digital Twins and Extended Reality for the Safety and Ergonomics Evaluation of Cobotic Workstations”, Frontiers in Virtual Reality, vol. 3, 2022.