# In Pursuit of Safety: An Open-Source Library for Physical Human-Robot Interaction

Benjamin Navarro, Aïcha Fonte, Philippe Fraisse, Gérard Poisson, Andrea Cherubini

# In Pursuit of Safety

## *An Open-Source Library for Physical Human–Robot Interaction*



By Benjamin Navarro, Aïcha Fonte, Philippe Fraisse, Gérard Poisson, and Andrea Cherubini

OpenPHRI is a C++/Python general-purpose software scheme with several built-in safety measures designed to ease robot programming for physical human–robot interaction (pHRI) and collaboration. Aside from providing common functionalities, the library can be easily customized and enhanced thanks to the project's open-source nature. The OpenPHRI framework consists of a two-layer damping controller, depicted in Figure 1. This allows the user to provide compliance and other safety features at both the joint and task levels, depending on the application.

### pHRI

A situation in which direct contact occurs between a person and a robot is referred to as *pHRI*. From the human perspective, such interplay can be either intentional or undesired. Undesired contact may occur if a person enters the robot workspace without activating a presence-detection system (e.g., a light barrier, floor mat, or laser scanner). Such contact may, of course, lead to severe injuries. Voluntary physical interactions, by contrast, are needed whenever a person connects with a robot to stop, guide, or teach it a behavior.

This type of interaction is needed in factories so that robots and workers may operate closely or jointly. Other scenarios include physiotherapy health-care centers and domestic assistance cases for elderly or disabled people. In all these situations, measures must be taken to ensure the safety of the people in a robot's vicinity.

Such measures can be implemented at the hardware level, using passively compliant actuators, or at the control/software level. While mechanically compliant devices allow fast impact force absorption, they are available on only a restricted set of robots and add a nonnegligible cost to the platform. However, control-level solutions can be applied to virtually any robot. Moreover, they can provide preventive actions (e.g., collision avoidance and deceleration) that reduce the risk of undesired

*Digital Object Identifier 10.1109/MRA.2018.2810098*
*Date of publication: 17 May 2018*

impact with the operator. Ideally, both solutions should be combined to provide the highest level of safety.

Although pHRI has been extensively investigated by the research community, to the best of our knowledge, no general open-source software solution exists to date. Thus, each research team or industrial organization is forced to develop its own applications, limiting the adoption, benchmarking, and growth of pHRI in the community. So the main motivation behind OpenPHRI is to provide a full-featured, open-source software library that can also be easily extended to develop pHRI applications.

## Overview of the Library

The controller, constraints, and inputs described in this article are all available in the OpenPHRI software library, distributed online [14] free of charge under the GNU Lesser General Public License version 3 (LGPLv3) [15]. This license allows integration with open- or closed-source software as long as any modifications made to the library are shared with the community.

OpenPHRI is written in C++ to maximize efficiency in terms of computation and memory footprint and to easily embed it in existing projects. Python bindings are also provided, because this language is largely used in the robotics community and because it allows quick prototyping, as most computations are performed in machine language to keep computational times small. An interface with the robotics simulator Virtual Robot Experimentation Platform (V-REP) [16] is also furnished. The V-REP remote Application Programming Interface (API) library embedded in OpenPHRI has no particular license, so it does not violate the LGPLv3. A wrapper for the Robot Operating System framework will be released in the near future. Users can easily integrate other simulators, frameworks, and robots at will. The detailed hierarchy of the project is given in Table 1.

### *Example*

Listing 1 presents a short but meaningful example of OpenPHRI usage. In fewer than 35 lines of code (comments excluded), one can set up a V-REP scenario in which a serial manipulator robot Kuka LWR4+ is moved with an external force while at the same time limiting its velocity, reading sensory input, and sending joint commands to the simulator. It can be seen (at lines 10 and 18) that smart pointers (shared pointers from the standard C++ library) are used instead of raw pointers to pass data through the library. This has the advantage of automatically releasing the associated memory when it is no longer referenced in the program, avoiding memory leaks. Also, using pointers instead of values allows the user to change some parameters (e.g., maximum velocity) online very easily.

Because the example is self-explanatory thanks to the comments, we highlight only a few key elements of the library. First, we require a robot object. This is a data structure containing all of the information regarding its current state (e.g., joint positions, external force, and kinematics)

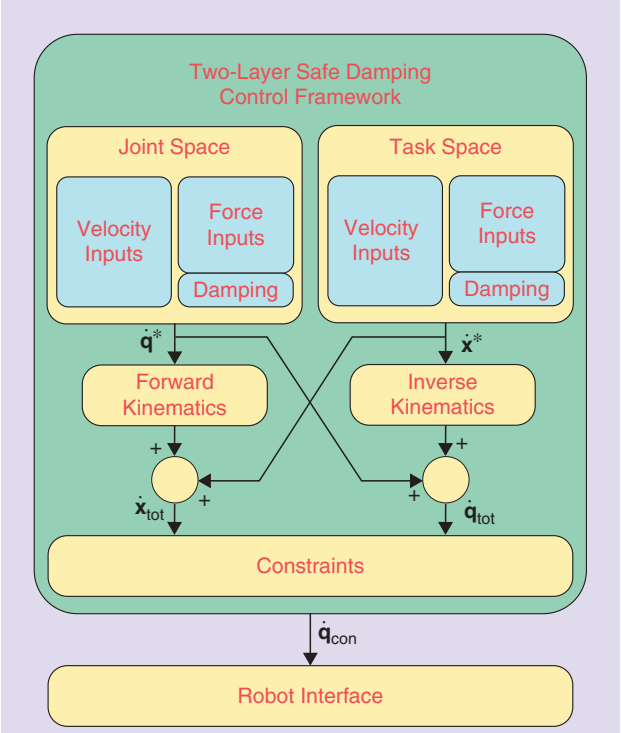


**Figure 1.** An overview of the OpenPHRI framework.

and control parameters (e.g., velocity bounds and damping factor). Next, we create the controller itself, called *SafetyController*, and we pair it to the robot to be controlled. A generic add method can be used to add constraints, velocity, and force inputs to the controller. The name given as the first parameter to the add method can be used to retrieve or remove the associated constraint or input from the controller. Then, to run the controller, we use the call operator (line 36).

## The OpenPHRI Framework

The OpenPHRI control framework (outlined in Figure 1), is based on damping control, a particular case of impedance control [1], which makes the robot act as a mass-spring-damper system:

$$\mathbf{f}^* = \mathbf{K}_t \Delta\mathbf{x} + \mathbf{B}_t \Delta\dot{\mathbf{x}} + \mathbf{M}_t \Delta\ddot{\mathbf{x}}, \tag{1}$$

with $\mathbf{K}_t$, $\mathbf{B}_t$, and $\mathbf{M}_t$ the stiffness, damping, and mass matrix, respectively, and $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_r$, $\mathbf{x}$ and $\mathbf{x}_r$ being the current and reference positions, respectively. $\mathbf{f}^*$ is the force to be realized by the robot. Here, we consider only damping control, under its admittance form, since it allows the mixing of velocities and forces together (the two types of input considered in this work). Then, we extend the paradigm to both task and joint spaces using

$$\dot{\mathbf{x}}^* = \mathbf{B}_t^{-1}\, \mathbf{f}_{ext} + \dot{\mathbf{x}}_r, \tag{2}$$

$$\dot{\mathbf{q}}^* = \mathbf{B}_j^{-1}\, \boldsymbol{\tau}_{ext} + \dot{\mathbf{q}}_r, \tag{3}$$

with $\dot{\mathbf{x}}^*$ and $\dot{\mathbf{q}}^*$ being the output velocity at each level, $\mathbf{B}_t$ and

**Table 1. The detailed project hierarchy.**

| Hierarchy | Content |
|---|---|
| src | Source files for the libraries |
| • OpenPHRI | C++ implementation of the controller and of the robot data structure |
| ◦ constraints | Constraints implementation |
| ◦ force_generators | Task space force inputs |
| ◦ velocity_generators | Task space velocity inputs |
| ◦ torque_generators | Joint space force inputs |
| ◦ joint_velocity_generators | Joint space velocity inputs |
| ◦ utilities | Various utilities, such as clock, data logger, integrator/derivator |
| • pyOpenPHRI | Python bindings for OpenPHRI, developed with Boost.Python* |
| • vrep_remote_api | API** for external V-REP control |
| • vrep_driver | OpenPHRI to V-REP interface |
| include | Header files for the libraries, which follow the same structure as src |
| tests | Unit tests for various parts of the OpenPHRI library |
| apps | Examples and demonstrations to help get started with OpenPHRI |
| share | Robot models and scenes for V-REP |
| build | Build directory |

*http://www.boost.org/doc/libs/1_64_0/libs/python/doc/html/index.html
**Courtesy of Coppelia Robotics.

$\mathbf{B}_j$ being the diagonal positive matrices of the damping parameters, and $\mathbf{f}_{ext}$ and $\boldsymbol{\tau}_{ext}$, respectively, being the forces at the control point (joints). For simplicity, we will use the general term *force* when dealing with both forces and torques at either the control point or joints. The control point is usually the end effector or the tip of the attached tool. Also, throughout this article, the subscripts *t* and *j* indicate variables related to task and joint space, respectively.

Although (2) and (3) have proven useful for complying with interaction forces while following a predefined trajectory, they can be extended to fit many more scenarios. We designed a more generic controller that includes sets of force inputs $\mathcal{F}$ and $\Gamma$ and of velocity inputs $\mathcal{V}$ and $\Omega$:

$$\dot{\mathbf{x}}^* = \mathbf{B}_t^{-1} \sum_{\mathbf{f}_i \in \mathcal{F}} \mathbf{f}_i + \sum_{\dot{\mathbf{x}}_i \in \mathcal{V}} \dot{\mathbf{x}}_i, \tag{4}$$

$$\dot{\mathbf{q}}^* = \mathbf{B}_j^{-1} \sum_{\boldsymbol{\tau}_i \in \Gamma} \boldsymbol{\tau}_i + \sum_{\dot{\mathbf{q}}_i \in \Omega} \dot{\mathbf{q}}_i. \tag{5}$$

Typically, real forces can be combined with virtual ones, and it is possible to add virtual velocity sources in $\mathcal{V}$ and $\Omega$ to the reference (real) joint or task space velocities. In this work, the focus has been on

- real interaction forces exchanged with the human or the environment
- virtual mass and stiffness forces (generating inertial and elastic effects)
- virtual forces repelling obstacles
- velocities generated by a predesigned trajectory generator
- velocities output by a force controller.

This is not restrictive, and other inputs can be considered to fit more scenarios.

When considering the safety of human–robot interaction, most solutions can be expressed in some form of velocity reduction. This includes stopping the robot upon contact, reducing its velocity when operators are approaching, and imposing constraints on velocity, kinetic energy, or transferred power. To assess the danger, we must monitor the velocities in both the task and joint space, because either one can lead to undesired behaviors. The total task and joint space velocities can be derived from (4) and (5) using forward and inverse kinematics:

$$\dot{\mathbf{x}}_{tot} = \dot{\mathbf{x}}^* + \mathbf{J}\dot{\mathbf{q}}^*, \tag{6}$$

$$\dot{\mathbf{q}}_{tot} = \mathbf{J}^{\dagger}\dot{\mathbf{x}}^* + \dot{\mathbf{q}}^*, \tag{7}$$

with $\mathbf{J}$ the task Jacobian. Note that (6) and (7) are related by $\dot{\mathbf{x}}_{tot} = \mathbf{J}\dot{\mathbf{q}}_{tot}$ and, as such, represent the same motion expressed in two different spaces. Vectors $\dot{\mathbf{x}}^*$ and $\dot{\mathbf{q}}^*$ are needed in both equations so that, for example, one can design a joint trajectory in $\Omega$ and add some compliance to the control point by including the external force in $\mathcal{F}$.

Both (6) and (7) must be solved to derive the set of constraints $\mathcal{C}$ that slow down the robot motion if needed (see Figure 1). Constraints can be expressed at both the joint and task levels, depending on the safety requirements or on the available sensor inputs (e.g., collision

**Listing 1. An example of a short OpenPHRI application.**

```
#include <OpenPHRI/OpenPHRI.h>
#include <vrep_driver/vrep_driver.h>

//Use namespaces to shorten the types
using namespace phri;
using namespace std;

int main(int argc, char* argv[]) {
  //Create a robot with a name (used by the V-REP driver) and a joint count
  auto robot = make_shared<Robot>("LBR4p", 7);
  //Set task space damping values to 100
  *robot->controlPointDampingMatrix() *= 100.;

  //Create a controller for the robot
  auto safety_controller = SafetyController(robot);

  //Create a pointer to store the maximum velocity, here 0.1 m/s
  auto max_vel = make_shared<double>(0.1);
  //Add this to the controller
  safety_controller.add("velocity constraint", VelocityConstraint(max_vel));

  //Feed the external force to the controller
  safety_controller.add("external force",ExternalForce(robot));

  //Create a V-REP driver for sending joint positions with 5-ms sample time
  vrep::VREPDriver driver(robot, ControlLevel::Joint, 0.005);
  //Use V-REP synchronous mode.
  driver.enableSynchronous(true);
  //Start the simulation
  driver.startSimulation();

  while(1) {
    //Update the robot with the current simulation data
    driver.getSimulationData();
    //Run the controller
    safety_controller();
    //Send the control output
    driver.sendSimulationData();
    //Trigger a simulation step
    driver.nextStep();
  }
}
```

detection based on joint torque sensors or the force/torque sensor at the end effector). The constraints are scalar values $C_i \in \mathbb{R}_{\geq 0}$ that become active when they are smaller than one. They determine the value of the velocity scaling factor $\alpha \in [0, 1]$:

$$\alpha = \min(1, \min(C)). \tag{8}$$

This is finally used to reduce (if needed) the joint velocity that is sent to the robot actuators:

$$\dot{\mathbf{q}}_{\text{con}} = \alpha\, \dot{\mathbf{q}}_{\text{tot}}. \tag{9}$$

If multiple constraints are active at the same time, the most restrictive one, i.e., the one leading to the lowest velocity, will be chosen. This ensures that all of the constraints are satisfied at any given time. Equations (4)–(9) make up the OpenPHRI framework.

Because multiple inputs can be enabled at the same time, they may not be all realizable. This can occur, for instance, if the robot must deviate from its task (or joint) space trajectory by the presence of an obstacle. In this case, the control velocity vector will be composed of both inputs. In general, conflicting inputs can arise whenever the controller has been misconfigured. Nevertheless, the OpenPHRI framework is designed to guarantee the safety constraints $\mathcal{C}$ at its lower layer. Hence, these will always be satisfied, rendering the robot motion safe. In the next sections, we detail the various force and velocity control inputs as well as the constraints that have been considered in this work.

## Force Inputs

This section presents joint or task space force inputs that, when included in sets $\mathcal{F}$ in (4) and $\Gamma$ in (5), respectively, make the robot comply with real-world forces or react to virtual ones. An illustrative example is given in Figure 2.

### *Interaction Forces*

In many cases, it is necessary to adapt the robot motion in the presence of external forces, e.g., for kinesthetic guidance (teaching by demonstration). In such scenarios, the external force $\mathbf{f}_{\text{ext}}$ can be included in $\mathcal{F}$. If this is the only force input in $\mathcal{F}$, the controller is a classic damping controller. The same can be accomplished in the joint space by including $\boldsymbol{\tau}_{\text{ext}}$ in $\Gamma$.

### *Virtual Mass and Stiffness*

Using a full admittance model, including stiffness and mass effects in (2) or (3), has been intensively investigated in the literature and proven useful in many cases [2]–[4]. This can be easily done in our framework by adding a virtual spring and/or a virtual mass that generates forces along any motion direction. In the task space, for instance, these virtual forces will be $\mathbf{f}_{\text{t,stiff}} = -\mathbf{K}_{\text{t}}(\mathbf{x} - \mathbf{x}_{\text{r}})$ and $\mathbf{f}_{\text{t,mass}} = -\mathbf{M}_{\text{t}}(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_{\text{r}})$, with $\mathbf{K}_{\text{t}}$ and $\mathbf{M}_{\text{t}}$ the diagonal positive semidefinite matrices of stiffness and mass parameters, respectively.

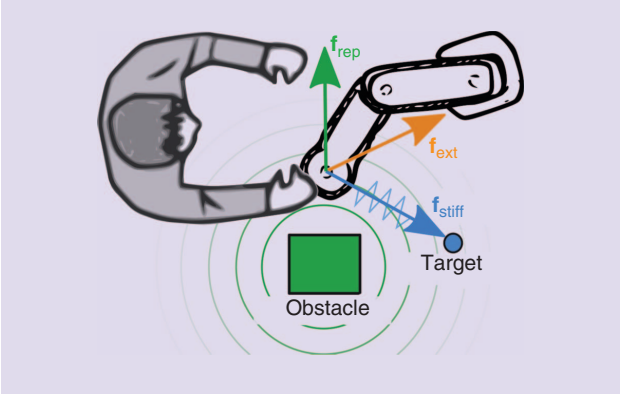


**Figure 2.** Some examples of interaction, stiffness, and repulsive forces.

### *Virtual Repulsive Forces*

To prevent the robot from hitting operators or to control its motion in a cluttered environment, a collision avoidance algorithm should be used. Despite providing local solutions, the potential fields approach [5] is well adapted to dynamic scenarios where a complete knowledge of the environment is unavailable because of moving and unpredictable obstacles and a limited field of view of the sensors. The potential fields approach consists of modeling obstacles (targets) as sources of repulsive or attractive forces. Summing up these forces results in a motion in the most promising direction. Hence, potential fields can be trivially integrated within our framework by adding the required virtual forces (e.g., repulsive forces $\mathbf{f}_{\text{rep}}$) to sets $\mathcal{F}$ or $\Gamma$.

## Velocity Inputs

In this section, we describe possible joint and task space velocity inputs to be included in $\Omega$ and $\mathcal{V}$, respectively. These velocities can be the result of a trajectory generator or of a force-control law.

### *Velocity Reference Trajectory*

Because trajectory generation and tracking are present in most robotics applications, it is crucial to have it in OpenPHRI. To this end, we developed within the library a trajectory generator based on fifth-order polynomials. This outputs smooth velocity trajectories [$\dot{\mathbf{q}}_{\text{r}}(t)$ or $\dot{\mathbf{x}}_{\text{r}}(t)$] with the following features:

- Each trajectory can have an arbitrary initial and final value as well as a first and second derivative, for a total of six degrees of freedom (6 DoF).
- Each trajectory can be composed of multiple segments (fifth-order polynomials), joined by intermediate waypoints. The trajectory is $C^3$.

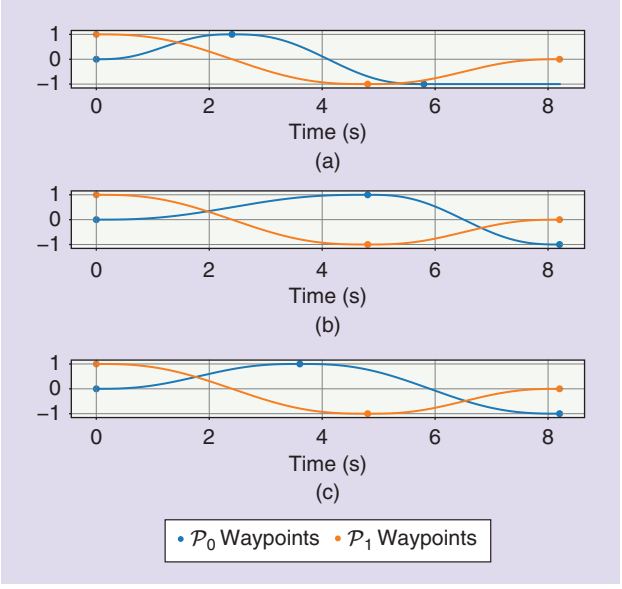


**Figure 3.** An illustration of trajectory synchronization. Given two trajectories, each composed of two segments (one intermediate waypoint), we can apply (a) no synchronization, (b) waypoint synchronization, and (c) whole trajectory synchronization.

- Multiple trajectories, each composed of several segments, can be synchronized (see Figure 3).
- Arbitrary first- and second-derivative constraints can be applied to each segment, and the trajectory duration is determined accordingly.
- Instead of first- and second-derivative limits, a minimum duration can be specified (e.g., it can be increased for synchronization purposes).

Similar existing solutions, such as the Reflexxes Motion Library [6], do not provide bounded continuous accelerations or waypoints for complex trajectory design.

### Velocities for Force Control

Force control is required for various applications, such as grinding, polishing, assembling, echographic monitoring, needle insertion, and minimally invasive surgery. To include force control in our framework, we map it to a velocity command. Let us first define the target force $\mathbf{f}_r$ and its associated error vector:

$$\Delta\mathbf{f} = \mathbf{S}(\mathbf{f}_r - \mathbf{f}_{ext}). \tag{10}$$

Here, $\mathbf{S}$ is a diagonal binary selection matrix, with elements $\mathbf{S}(i,i) = \{0,1\}$ used to set the task space components to be driven by the force controller. Then, proportion and derivative control can be applied to compute the control point velocity $\dot{\mathbf{x}}_{fc}$ that regulates $\Delta\mathbf{f}$ to 0. A similar technique can be applied in the joint space.

## Constraints

In this section, we qualitatively explain the design of each constraint $C_i$ that will reduce the robot velocity via (8) and (9). The equations are omitted for simplicity. For the implementation of these constraints, see the following classes: `EmergencyStopConstraint`, `VelocityConstraint`, `JointVelocityConstraint`, `AccelerationConstraint`, `PowerConstraint`, `ForceConstraint`, `KineticEnergyConstraint`, and `SeparationDistanceConstraint`.

### Emergency Stop

A simple way to provide some level of safety, as demonstrated in [7] and [8] and imposed by the ISO/TS 15066 safety standard [9], is to stop the robot motion when strong contact with a nearby operator occurs. We assume that the robot relies only on proprioception (external force measurement) and that physical contact with humans or with the environment should have limited magnitude. Then, to stop the robot, we set constraint $C_{stop}$ to zero as soon as $|\mathbf{f}_{ext}|$ (or $|\boldsymbol{\tau}_{ext}|$) passes some pretuned threshold. Deactivation thresholds are also needed to specify when to increase $C_{stop}$ to one, using hysteresis. Figure 4 gives the evolution of $C_{stop}$, with activation and deactivation thresholds of 5 N and 1 N, respectively.

### Velocity Limitation

Another very common safety criterion is velocity limitation. This is often present in robotics safety standards, such as ISO 10218:2011 [10] and ISO/TS 15066. Note that, even when the trajectory has been preplanned to fulfill the velocity bounds, other control inputs (e.g., kinesthetic forces applied by the operator, force control, or repulsive force) can lead to accelerations that break the constraint. To respect the limitation at all times, we design constraint $C_{vel}$ to be inversely proportional to the norm of the total velocity (either $\dot{\mathbf{x}}_{tot}$ or $\dot{\mathbf{q}}_{tot}$) and unitary when this is greater than the limit $V_{max}$. An illustration of this velocity limitation is given in Figure 5, where the norm of the total velocity $\dot{\mathbf{x}}_{tot}$ and the output velocity $\dot{\mathbf{x}}$ as well as the value of $C_{vel}$ are displayed.

### Acceleration Limitation

To avoid abrupt robot motions, the acceleration can also be limited to $A_{max}$. Because (8) offers velocity reduction only, we use $A_{max} > 0$. To limit the acceleration, we express $C_{acc}$ as $C_{vel}$, by replacing the current velocity with the predicted one in the formulation, if the acceleration was at its maximum allowed value $A_{max}$ during the next time step. The acceleration limitation mechanism is depicted in Figure 6. We can see that the velocity increases linearly during the first 3 s because of the acceleration limit.

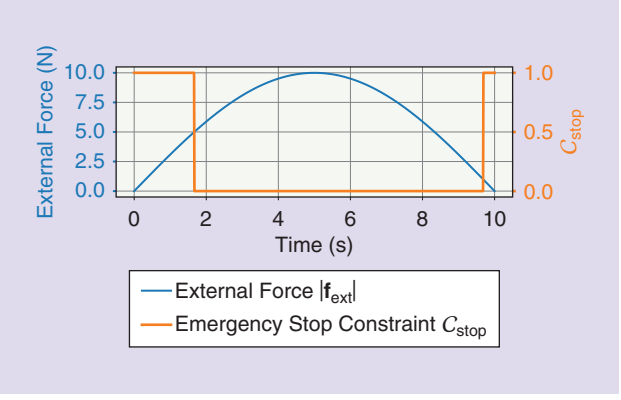


**Figure 4.** The characteristics of the emergency stop constraint. Activation threshold = 5 N; deactivation threshold = 1 N.

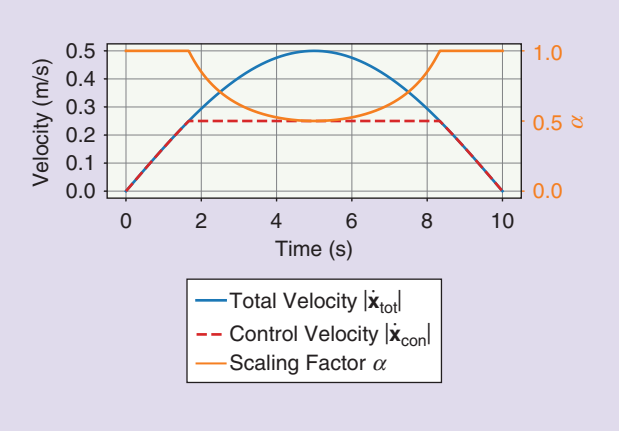


**Figure 5.** The evolution of the velocity, with $V_{max} = 0.25$ m/s.

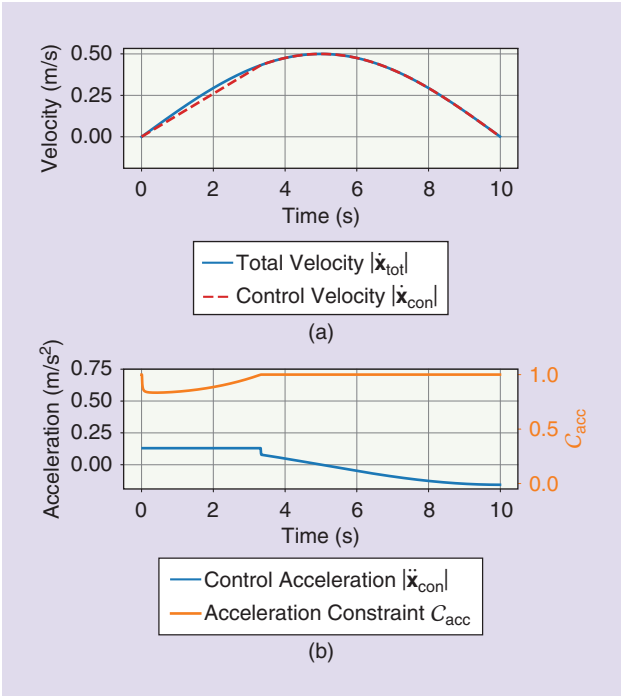


**Figure 6.** The evolution of (a) the velocity and (b) acceleration, with $A_{max} = 0.13\,m/s^2$.

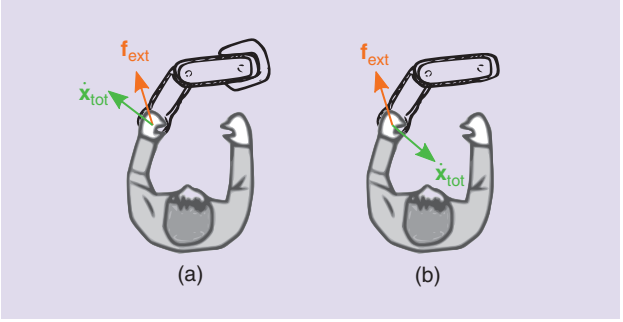


**Figure 7.** (a) Safe and (b) potentially unsafe situations depend on the sign of the transferred power.

## Power Limitation

The ISO 10218:2011 and ISO/TS 15066 standards also impose a limitation on the power exchanged between human and robot. Power can be limited at the hardware level, e.g., the electric power, as with the Kuka LWR4+, or at the control level, as we do here. The advantage of operating at the control level is that the limitation can be tuned online (e.g., deactivated to allow high dynamic motions when no operator is present). To define constraint $C_{pow}$, we consider the mechanical power, i.e., the scalar product between force and velocity. The limitation is effective only when this is negative, i.e., when energy is absorbed by the human and the robot represents a potential threat to the person (see Figure 7). In this case, $C_{pow}$ is inversely proportional to the power, and it is unitary if the absolute value of the power is greater than the allowed limit $P_{max} > 0$. An example of force limitation is depicted in Figure 8. We can see that the transmitted power is effectively limited only when it is negative and passes the limit.

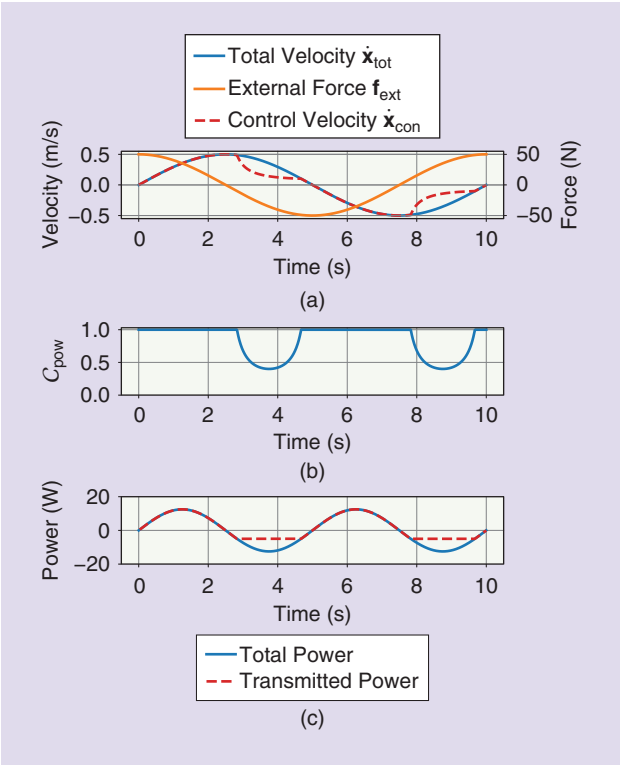


**Figure 8.** The evolution of the (a) velocity, (b) external force, and (c) transmitted power, with $P_{max} = 5$ W.

## Force Limitation

Force limitation is the third and final constraint imposed by ISO 10218:2011 and ISO/TS 15066. Actual force limitation is a very challenging problem, because it requires complete knowledge of the environment, including the humans present. While a complete map of the environment (position, materials, and so forth) can be obtained, it is nearly impossible to obtain the same knowledge regarding humans, as this would require estimating their motion and body impedance parameters, which change over time (e.g., fatigue or muscular cocontraction may stiffen a joint) and from one person to another. Hence, we decided to adopt a reactive approach that does not rely on an environmental model. By doing so, if the external force instantaneously passed a limit $F_{max} > 0$, the robot would react to quickly move away from the impact and reach a safe state.

Our approach has two steps. The first consists of generating a velocity in the direction opposite to the external force, to move away from the collision. This velocity, noted as $\dot{\mathbf{x}}_{F_{lim}}$, is added to set $\mathcal{V}$. The second step consists of including one or more constraints in $C$ to slow down the robot, according to (8), and guarantee that it behaves safely while executing $\dot{\mathbf{x}}_{F_{lim}}$. For example, with respect to ISO 10218:2011, both velocity and power limitations must be applied:

$$C_{force} = \min(C_{vel}, C_{pow}). \tag{11}$$

### Kinetic Energy Limitation

When robot and human collide, the level of injury endured by the latter can be related to the robot's kinetic energy [11], [12]. Hence, kinetic energy is a major concern when it comes to safety, and, as such, it should be limited. For a rigid body of mass $m$, the kinetic energy is defined as $E_k = m\|\mathbf{v}\|^2/2$. For rigid manipulators, an equivalent mass (perceived at any collision point on the robot structure) can be derived using the joint's dynamic model [11], [13]. Therefore, in both cases, limiting the kinetic energy can be seen as a form of velocity limitation, with the mass (real or equivalent) acting as a scale factor. As such, constraint $\mathcal{C}_{\text{kin}}$ is equivalent to $\mathcal{C}_{\text{vel}}$, with $V_{\max} = \sqrt{2E_{k\max}/m}$.

### Separation Distance

When the separation distance between the robot and nearby operators is monitored, it can be used to adapt the aforementioned limits (e.g., on velocity and power). It is also required to fulfill the Speed and Separation Monitor mode of ISO/TS 15066. For instance, a low level of security may be required if no one is present in the surroundings, whereas very strict limitations may be imposed when a robot is working near or in collaboration with humans. A simple example is depicted in Figure 9. To comply with this, we use fifth-order polynomials, implemented in OpenPHRI (OpenPHRI/utilities/fifth_order_polynomials.{h,cpp}), to allow a smooth adaptation of the limits, depending on the distance to the closest operator or to any other object to be avoided. Then, any limit (e.g., $V_{\max}$) will vary from a low value at a fixed minimal distance to a large value at a higher distance and may vary smoothly in between.

## Benchmark Tests

In pHRI, for the robot to react quickly in the case of an impact or to be as transparent as possible when physically collaborating with a human, its control loop should run at a minimum of 1 kHz. It is, therefore, crucial that the implementation of our controller in OpenPHRI be fast enough to comply with this timing constraint. To assess the performance of our library, we ran some benchmark tests on a computer equipped with an Intel i7-6700HQ at 2.6 GHz, operating Linux 4.11. Here, we refer to a benchmark in the computing (not robotics) sense—i.e., the act of running a computer program or a set of programs to assess their performance.

In Figure 10, we present the results of the benchmark tests for the controller associated with different constraints and force and velocity inputs, running on a 7-DoF manipulator. At each iteration, the controller is run 10,000 times to get meaningful results, and the average computation time is logged. In Figure 10(a)–(e), we give the average computation time $\bar{t}$ and the standard deviation $\sigma$ over 1,000 iterations. The computation of the forward and inverse kinematics is not included in these results so that we may focus on the control computation time overhead. Also, the current controller implementation is single-threaded, but, given the very low computation time [$\bar{t} < 4\ \mu s$ in the most complex scenario presented in Figure 10(e)], a multithreaded version does not seem necessary. Figure 10(f) shows that the memory usage (measured using the Massif tool from the Valgrind software) stays very low, with a peak at 186 KiB. (Here, the abscissa indicates snapshots taken regularly during execution.)

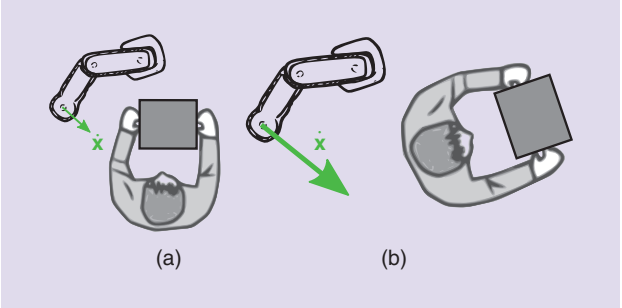


**Figure 9.** The velocity limitation varies smoothly as a function of the separation distance: (a) limited velocity; (b) full velocity.

## Experiments

In this section, we present the results of a full-featured experiment using the framework described in this chapter. The experiment is split into two phases: 1) a teaching-by-demonstration phase and 2) a replay phase, in which the robot operates autonomously in the presence of an obstacle and near the human operator. Figure 11 shows the setup, consisting of a Kuka LWR4+ arm with external force $\mathbf{f}_{\text{ext}}$, estimated through the Fast Research Interface (FRI) [17]. All the code was written in C++ using the OpenPHRI library and, to interface with the hardware, integrated inside the Knowbotics framework, currently under development at Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (a public release is expected once the software becomes mature enough). The FRI library was used to communicate with the Kuka arm. The controller sample time was $T = 1$ ms. To manage the robot behavior, we used OpenPHRI to design the finite state machine (FSM) shown in Figure 12.

It is important to note that our framework is used continuously throughout both the teaching and replay phases. An equivalent application using the V-REP simulator is available in the OpenPHRI repository under apps/demo [18]. The whole application has fewer than 600 lines of code: 125 for the main file and 440 for the FSM (header plus source). For the FSM, most of the code just adds or removes inputs and constraints to fit with Figure 12, so one can expect more or less code to write depending on the FSM complexity.

The teaching phase consists in manually guiding the robot [Figure 13(b) and (c)] by applying $\mathbf{f}_{\text{ext}} \in \mathcal{F}$, to teach it the waypoints where it should later realize a force-control task (applying $f_r = 30\ N$ for 2 s perpendicularly to the end effector). The number of waypoints is not known a priori. A waypoint is recorded when no motion is detected for 3 s, and the teaching phase ends if the robot remains still for 3 s more.

Once the operator has specified all the desired points, the

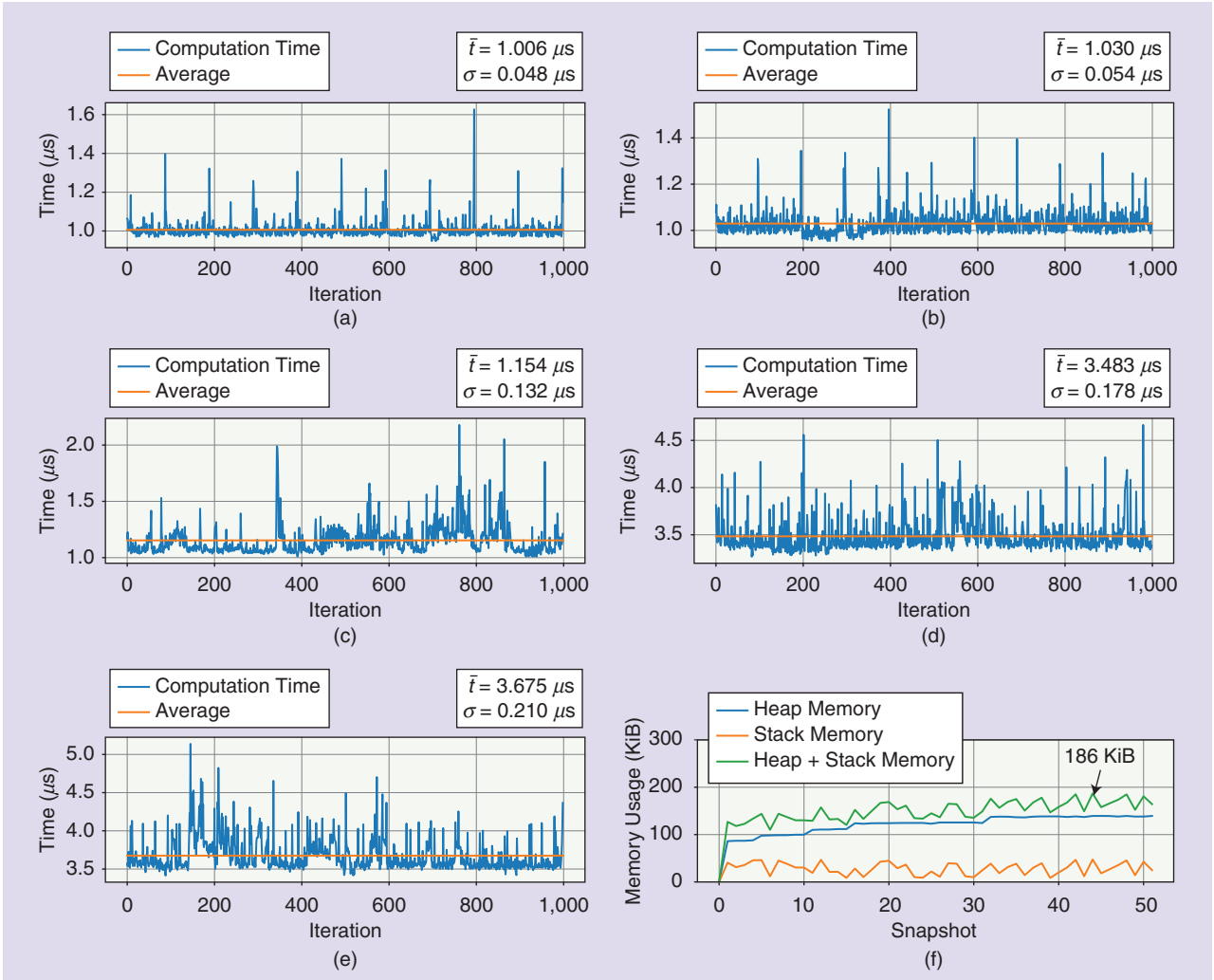


**Figure 10.** The benchmarks of the controller running on a 7-DoF manipulator. (a) The controller with no constraints and no inputs. (b) The controller with a velocity constraint ($C = \{C_{\text{vel}}\}$). (c) The controller with velocity and power constraints ($C = \{C_{\text{vel}}, C_{\text{pow}}\}$). (d) The controller with velocity, power, and kinetic energy constraints ($C = \{C_{\text{vel}}, C_{\text{pow}}, C_{\text{kin}}\}$). (e) The controller with velocity, power, and kinetic energy constraints and with a potential field, a virtual stiffness, and a force controller in the task space ($C = \{C_{\text{vel}}, C_{\text{pow}}, C_{\text{kin}}\}$, $\mathcal{F} = \{\mathbf{f}_{\text{t,stiff}}, \mathbf{f}_{\text{rep}}\}$, $\mathcal{V} = \{\dot{\mathbf{x}}_{\text{fc}}\}$. (f) The memory usage while sequentially executing the controller benchmarks (a)–(e).

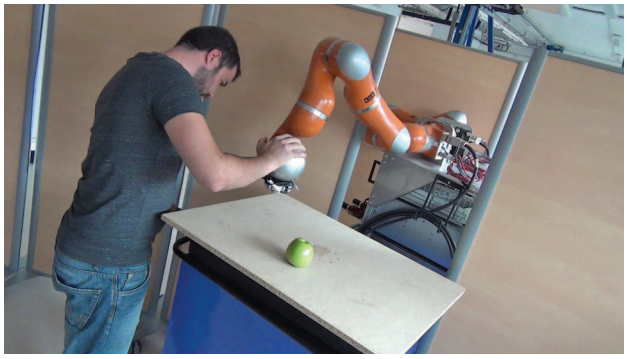

**Figure 11.** The setup for the experiment.

replay phase is triggered. The trajectory generator is used to output the control point (end effector) reference velocity ($\dot{\mathbf{x}}_{\text{r}} \in \mathcal{V}$) to reach each waypoint [Figure 13(d)]. When a waypoint is reached, the task space force controller is activated [$\dot{\mathbf{x}}_{\text{fc}} \in \mathcal{V}$, Figure 13(e)]. After the force has been correctly applied, the robot moves to the next waypoint. Once all of the force-control tasks have been performed, the robot returns to its original position, using the trajectory generator [Figure 13(h)]. During the replay phase, while moving between waypoints, the external force at the control point is monitored to trigger an emergency stop if its norm exceeds 10 N [Figure 13(g)], as explained in the "Emergency Stop" section. Motion is resumed only when the external force is lower than 1 N. Additionally, potential fields ($\mathbf{f}_{\text{rep}} \in \mathcal{F}$) are used to avoid a known object (here, an apple) in the environment [Figure 13(f)].

Throughout the experiment, the joint velocities sent to the robot are output by (9), with the scaling factor $\alpha$ computed with the constraints in (8). The task space damping matrix is set to $\mathbf{B}_t = \text{diag}(250, \ldots, 250)$, while joint space damping $\mathbf{B}_j$ is not used. During force-control tasks execution, an acceleration limit ($C_{\text{acc}} \in C$) of $A_{\max} = 0.5\ \text{m/s}^2$ is applied to avoid

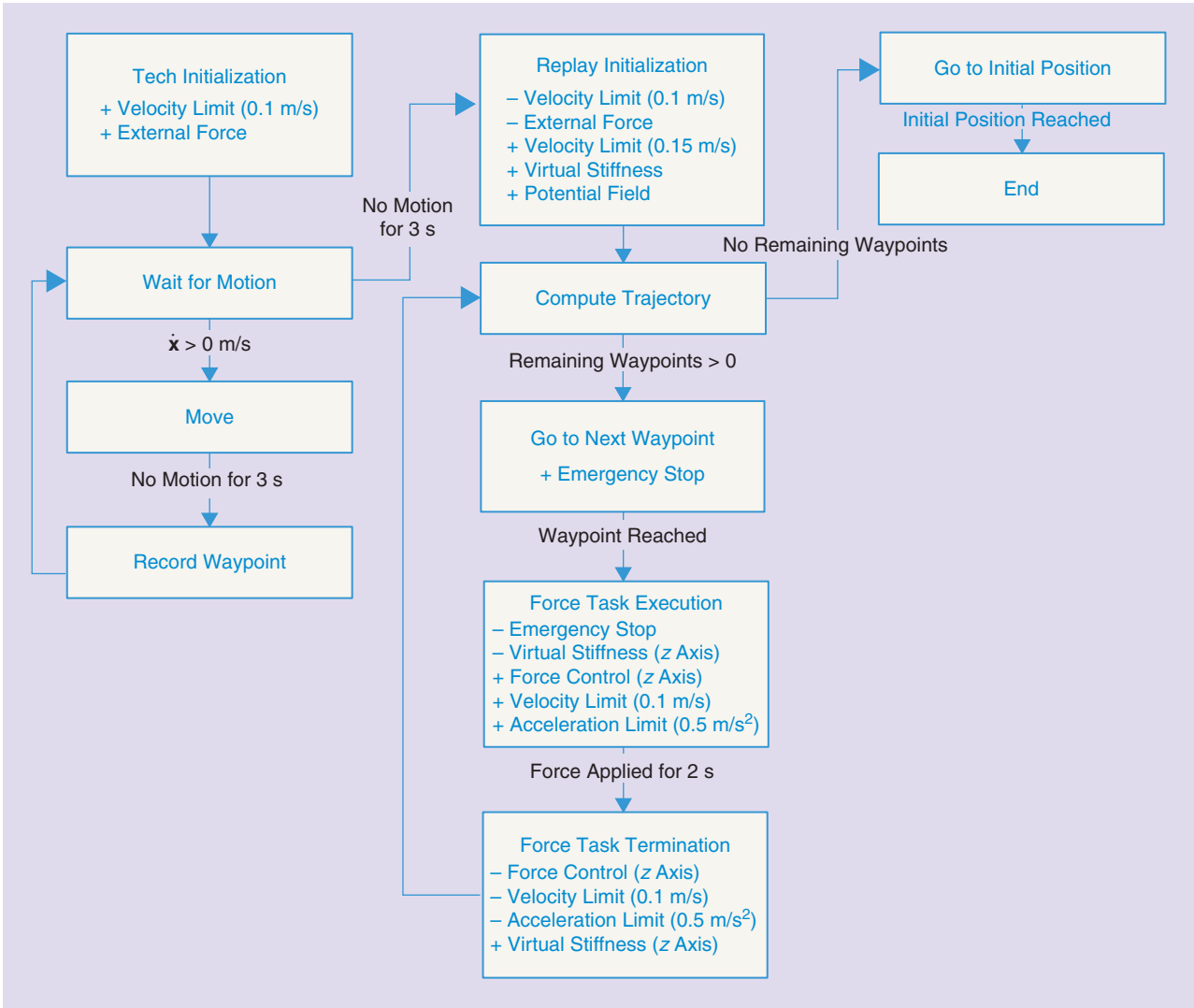


**Figure 12.** The FSM used for the experiment. A plus sign indicates an addition to the controller (a new constraint or new input) while a minus indicates a removal.

abrupt motions. During the replay phase, a virtual stiffness $\mathbf{K}_t = \text{diag}(1000, \ldots, 1000)$, described in the "Virtual Mass and Stiffness" section, is added to compensate for deviations from the trajectory. The potential fields for obstacle avoidance are activated when the distance from the obstacle is below 0.2 m. Throughout the experiment, the velocity is limited to $V_{\max} = 0.1$ m/s during teaching and to $V_{\max} = 0.15$ m/s during replaying.

Snapshots of the experiment are displayed in Figure 13, while the results are shown in Figure 14. A video of the experiment is available [19]. The teaching phase takes place during the first 36 s. Then the replay phase starts. Figure 14(c) shows that the scaling factor is decreased multiple times to comply with the constraints. For example, during manual guidance, the applied external forces, visible in Figure 14(a), would have led to velocities above the limit if $C_{\text{vel}}$ was not present. The same occurs when the obstacle is being avoided between 73 and 74 s. As Figures 14(a) and 13(d) show, at 70 s, an unexpected external force is applied by the operator, leading to a complete stop of the robot (normal operation is resumed at $t = 72$ s). Control point velocities before and after scaling are presented respectively in Figure 14(b) and (d). Finally, Figure 14(e) shows how scaling from the total ($\mathbf{v}_{\text{tot}}$) to the applied ($\mathbf{v}_{\text{con}}$) translational velocity norms complies with the imposed limit $V_{\max}$.

## Conclusions

This article introduces OpenPHRI, a new software library intended for pHRI and collaboration. We present its structure, including its core and components (force inputs, velocity inputs, and constraints), as well as a short but meaningful example. It should be noted that, in some cases, tasks (i.e., inputs) can conflict and hence not be fully realized. This behavior can be sometimes desirable (e.g., collision avoidance while following a trajectory) or unwanted (caused by controller misconfiguration). Nevertheless, the constraints applied at the lower layer of OpenPHRI, guarantee safe robot behavior at all times.

Aside from safety, a real-life experiment also demonstrates

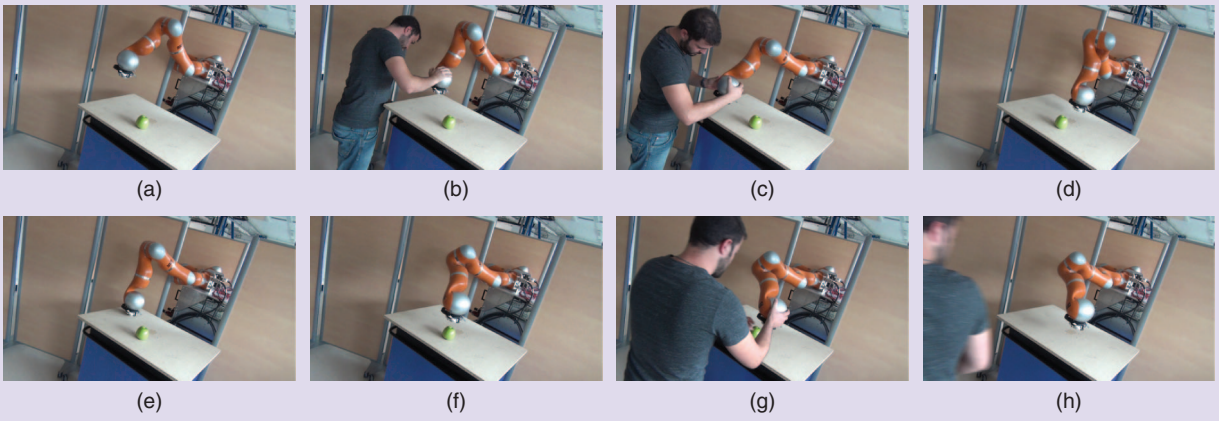


**Figure 13.** Some snapshots of the experiment: (a)–(c) teaching phases and (d)–(h) replay phases. (a) The robot is waiting in its initial position. (b) The operator teaches the first waypoint. (c) The operator teaches the second waypoint. (d) The robot goes to the first waypoint. (e) Force control is performed at the first waypoint. (f) The robot avoids the obstacle by using repulsive potential fields. (g) The operator stops the robot to access the workspace. (h) The robot returns to the initial pose.

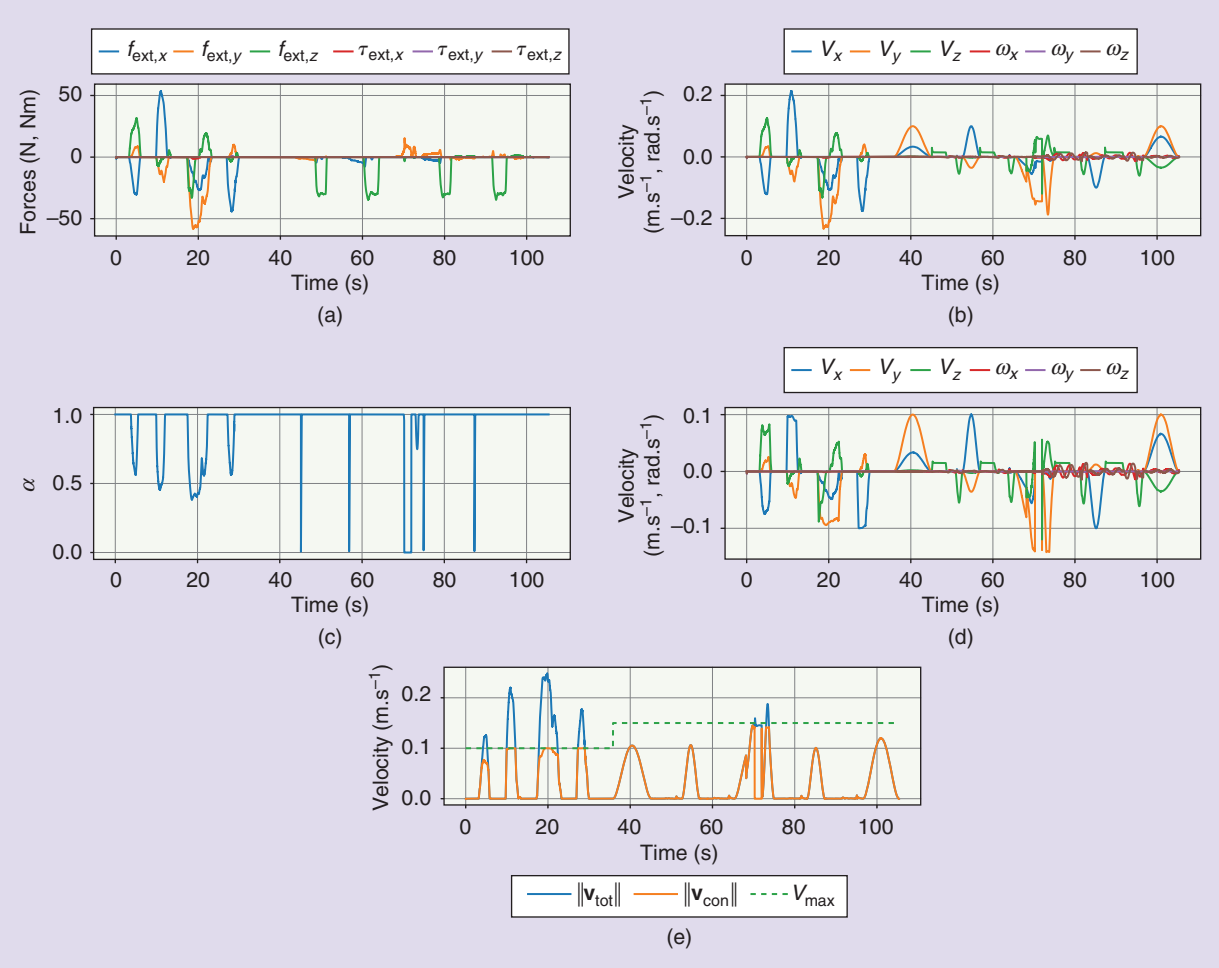


**Figure 14.** The relevant variables during the experiment. (a) The components of the external force $\mathbf{f}_{ext}$, applied by the human for teaching or upon collision (at t = 70 s), then by the robot during the four force-control tasks. (b) The components of the control point total velocity $\dot{\mathbf{x}}_{tot}$. (c) The scaling factor $\alpha$, diminishing whenever the constraints are active. (d) The components of the control point velocity applied after velocity reduction $\dot{\mathbf{x}}_{con} = \mathbf{J}\dot{\mathbf{q}}_{con}$. (e) A comparison between the current velocity limit $V_{max}$ and the total and applied translational velocity norms ($\|\mathbf{v}_{tot}\|$ and $\|\mathbf{v}_{con}\|$).

the advantages of OpenPHRI in terms of ease of use, both when the human is active (teaching by demonstration) and passive (the smooth generated trajectories are intuitive and predictable). In addition, OpenPHRI controllers can be executed at very high rates (>1 kHz) or on low-end machines, while still achieving excellent performance. Finally, OpenPHRI programs can be written in a very concise way, while retaining high readability. The open-source nature of the project allows its users to add new features and share them with the community, keeping the project up-to-date.

## Acknowledgments

This work was supported by the French National Research Agency SISCob project ANR-14-CE27-0016 and received funding from the European Union's Horizon 2020 research and innovation program under grant 731330.

## References

[1] N. Hogan, "Impedance control: An approach to manipulation. Part II: Implementation," *J. Dynamic Syst., Measurement, Control*, vol. 107, no. 1, pp. 8–16, 1985.

[2] S. Jung, T. C. Hsia, and R. G. Bonitz, "Force tracking impedance control of robot manipulators under unknown environment," *IEEE Trans. Control Syst. Technol.*, vol. 12, no. 3, pp. 474–483, May 2004.

[3] F. Almeida, A. Lopes, and P. Abreu, "Force-impedance control: A new control strategy of robotic manipulators," in *Recent Advances in Mechatronics*. Singapore: Springer-Verlag, 1999, pp. 126–137.

[4] H. Sadeghian, L. Villani, M. Keshmiri, and B. Siciliano, "Experimental study on task space control during physical human robot interaction," in *Proc. 2nd RSI/ISM Int. Conf. Robotics and Mechatronics (ICRoM)*, Oct. 2014, pp. 125–130.

[5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robotics Res.*, vol. 5, no. 1, pp. 90–98, Mar. 1986.

[6] Torsten Kroger, "Opening the door to new sensor-based robot applications: The Reflexxes Motion Libraries," in *Proc. 2011 IEEE Int. Conf. Robotics and Automation*, May 2011. doi: 10.1109/ICRA.2011.5980578.

[7] Y. Yamada, Y. Hirasawa, S. Huang, Y. Umetani, and K. Suita, "Human-robot contact in the safeguarding space," *IEEE/ASME Trans. Mechatronics*, vol. 2, no. 4, pp. 230–236, 1997.

[8] A. De Luca and F. Flacco, "Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration," in *Proc. 2012 4th IEEE RAS and EMBS Int. Conf. Biomedical Robotics and Biomechatronics (BioRob)*, June 2012. doi: 10.1109/BioRob.2012.6290917.

[9] *Robots and Robotic Devices—Collaborative Robots*, International Organization for Standardization Standard ISO/TS 15066:2016, 2016.

[10] *Robot for Industrial Environments—Safety requirements—Part 1: Robot*, International Organization for Standardization Standard ISO 10218-1:2011, 2006.

[11] S. Haddadin, S. Haddadin, A. Khoury, T. Rokahr, S. Parusel, R. Burgkart, A. Bicchi, and A. Albu-Schaffer, "A truly safely moving robot has to know what injury it may cause," in *Proc. 2012 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Oct. 2012. doi: 10.1109/IROS.2012.6386163.

[12] A. Meguenani, V. Padois, and P. Bidaud, "Control of robots sharing their workspace with humans: An energetic approach to safety," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, Sept. 2015, pp. 4678–4684.

[13] O. Khatib, "Inertial properties in robotic manipulation: An object-level framework," *Int. J. Robotics Res.*, vol. 14, no. 1, pp. 19–36, Feb. 1995.

[14] B. Navarro. (2017, Sept. 25). OpenPHRI, a complete and generic solution for safe physical human-robot interactions. [Online]. Available: https://github.com/BenjaminNavarro/OpenPHRI

[15] Free Software Foundation. (2016, Nov. 18). GNU lesser general public license. [Online]. Available: https://www.gnu.org/licenses/lgpl-3.0.en.html

[16] Coppelia Robotics. V-REP: Virtual Robot Experimentation Platform. [Online]. Available: http://www.coppeliarobotics.com

[17] Fast Research Interface Library: Manual and documentation. [Online]. Available: http://cs.stanford.edu/people/tkr/fri/html/

[18] B. Navarro. (2017, Sept. 6). Open-phri. [Online]. Available: https://github.com/BenjaminNavarro/OpenPHRI/tree/master/apps/demo

[19] B. Navarro. (2017, Nov. 2). OpenPHRI framework demonstration. [Online]. Available: http://bit.do/openphrivideo

***Benjamin Navarro,*** Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, Centre National de la Recherche Scientifique-Université de Montpellier, France. PRISME Laboratory, University of Orléans, France. E-mail: navarro@lirmm.fr.

***Aïcha Fonte,*** Laboratoire Pluridisciplinaire de Recherche, Ingénierie des Systèmes, Mécanique, Énergétique, University of Orléans, France. E-mail: aicha.fonte@univ-orleans.fr.

***Philippe Fraisse,*** Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, Centre National de la Recherche Scientifique-Université de Montpellier, France. E-mail: fraisse@lirmm.fr.

***Gérard Poisson,*** Laboratoire Pluridisciplinaire de Recherche, Ingénierie des Systèmes, Mécanique, Énergétique, University of Orléans, France. E-mail: gerard.poisson@univ-orleans.fr.

***Andrea Cherubini,*** Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, Centre National de la Recherche Scientifique-Université de Montpellier, France. E-mail: cherubini@lirmm.fr.