OpenPHRI: an open-source library for safe physical human-robot interaction

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

OpenPHRI: an open-source library for safe physical human-robot interaction

Benjamin Navarro[1,2], Aïcha Fonte[2], Philippe Fraisse [1], Gérard Poisson[2], and Andrea Cherubini[1]

[1] LIRMM CNRS-UM, 161 Rue Ada, 34090 Montpellier, France. lastname@lirmm.fr
[2] PRISME Laboratory, University of Orléans, 12, Rue de Blois, 45067 Orleans, France.
firstnames.lastname@univ-orleans.fr

August 1, 2017

OpenPHRI is a C++/Python general purpose software with several built-in safety measures, designed to ease robot programming for physical human-robot interaction (pHRI) and collaboration (pHRC). 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 in 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.

1 Physical Human-Robot Interaction

Physical human-robot interaction refers to situations where a direct contact occurs between the two agents. From the human perspective, such contact can be either intentional or undesired. Undesired contacts may happen if the human enters the robot workspace and no presence detection system (e.g. light barrier, floor mat, laser scanner) is active. These contacts may of course lead to severe injuries. Voluntary physical interactions, on the contrary, are needed whenever the person makes contact with the robot to stop it, guide it, or teach it a behavior.

This type of interaction is more and more demanded in factories, to make robots and workers operate closely or jointly. Other scenarios include health care centers for physiotherapy and domestic assistance of elderly or disabled people. In all these situations, measures must be taken to ensure the safety of persons present in the 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 only available on a restricted set of robots, and add a non negligible cost to the platform. On the contrary, control level solutions can be applied to virtually any robot. Moreover, they can provide preventive actions (e.g. collision avoidance, deceleration) that reduce the risk of undesired impact with the operator. Ideally, both solutions should be combined to provide the best 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 is forced to develop its own software, limiting the adoption, benchmarking and growth of pHRI in the community.

Table 1: 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, intergrator/derivator
• pyOpenPHRI	Python bindings for OpenPHRI, developed with Boost.Python[a]
• vrep_remote_api	API[b] for external V-REP control
• vrep_driver	OpenPHRI to V-REP interface
include	Header files for the libraries, that follow the same structure as src
tests	Unit tests for various parts of the OpenPHRI library
apps	Example and demonstrations, to help getting started with OpenPHRI
share	Robot models and scenes for V-REP
build	Build directory

[a] `http://www.boost.org/doc/libs/1_64_0/libs/python/doc/html/index.html`
[b] Courtesy of Coppelia robotics.

This is the main motivation behind OpenPHRI: to provide a full-featured open source software library, that can also be easily extended, to develop pHRI applications.

2 Overview of the library

The controller, constraints and inputs described in this paper are all available in the OpenPHRI software library, distributed online[1] free of charge under the GNU LGPL license[2]. The library 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, since this language is largely used in the robotics community and since it allows quick prototyping, while keeping low computational times, as most computations are performed in machine language. An interface with the robotics simulator V-REP[3] is provided and interfaces to other simulators and robots can be easily added. The detailed hierarchy of the project is given in Table 1.

2.1 Example

Listing 1 presents a short but meaningful example of OpenPHRI usage. In less than 35 lines of code (comments excluded) one can set up a V-REP scenario where a serial manipulator robot Kuka LWR4+ is moved with an external force applied, while 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[4] are used instead of raw pointers to pass data through the library. This has the advantage of releasing automatically 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

[1] `https://github.com/BenjaminNavarro/OpenPHRI`
[2] `https://www.gnu.org/licenses/lgpl-3.0.en.html`
[3] `http://www.coppeliarobotics.com`
[4] Shared pointers from the standard C++ library.

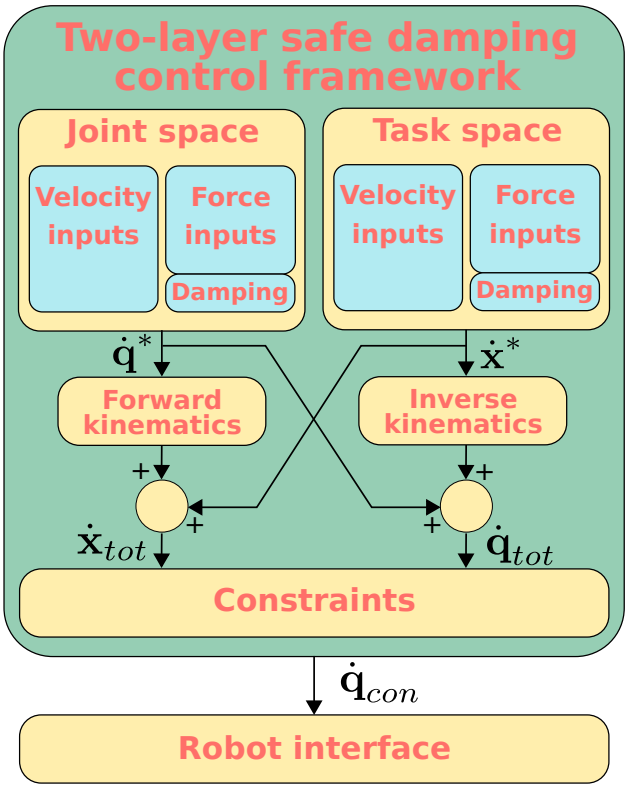


Figure 1: Overview of the OpenPHRI framework.

parameters (e.g. maximum velocity) online very easily.

Since the example is self-explanatory thanks to the comments, we only highlight few key elements of the library. First, a *Robot* object is required. This is a data structure containing all the information regarding its current state (e.g., joint positions, external force, kinematics) and control parameters (e.g., velocity bounds, damping factor). Next, the controller itself, called *SafetyController*, is created and paired to the robot to control. A generic *add* method can be used to add constraints, velocity and force inputs to the controller. The name given as 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, the call operator (line 36) is used.

3 The OpenPHRI framework

The OpenPHRI control framework (outlined in Fig. 1), is based on damping control, a particular case of impedance control [1]. Impedance control relies on a mass-spring-damper system, that relates the external forces $\mathbf{f}_{ext}$ applied on the robot control point to its displacement from the reference pose $\mathbf{x}_r$. Here, we consider only damping, and extend the paradigm to both task and joint spaces, using:

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

$$\dot{\mathbf{q}}^* = \mathbf{B}_j^{-1}\ \boldsymbol{\tau}_{ext} + \dot{\mathbf{q}}_r, \tag{2}$$

with $\mathbf{B}_{t,j}$ diagonal positive matrices of damping parameters, and $\mathbf{f}_{ext}$ (respectively, $\boldsymbol{\tau}_{ext}$) forces at the control point (joints). For simplicity, the general term force will be used when dealing with both forces and torques at either the control point or joints. Also, throughout this paper, subscripts t and j indicate variables related respectively to task and joint space.

Although (1) and (2) were proven useful for complying with interaction forces while following a predefined trajectory, they can be extended to fit many more scenarios. We design a more generic controller that includes sets of *force inputs* $\mathcal{F}$ and Γ and of *velocity inputs* $\mathcal{V}$ and Ω:

$$\dot{\mathbf{x}}^* = \mathbf{B}_t^{-1} \sum_{\mathbf{f}_i \in \mathcal{F}} \mathbf{f}_i + \sum_{\dot{\mathbf{x}}_i \in \mathcal{V}} \dot{\mathbf{x}}_i \tag{3}$$

$$\dot{\mathbf{q}}^* = \mathbf{B}_j^{-1} \sum_{\boldsymbol{\tau}_i \in \Gamma} \boldsymbol{\tau}_i + \sum_{\dot{\mathbf{q}}_i \in \Omega} \dot{\mathbf{q}}_i. \tag{4}$$

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

```cpp
#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 0.1m/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 5ms sample time
  vrep::VREPDriver driver(robot, ControlLevel::Joint, 0.005);
  // Use V-REP synchronous mode.
  driver.enableSynchronous(true);
  // Start the simualtion
  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();
  }
}
```

Listing 1: Example of a short OpenPHRI application.

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

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

When considering 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 nearby operators are approaching, and imposing constraints on velocity, kinetic energy or transferred power. To assess the danger, we must monitor the velocities in both task and joint space, since either one can lead to undesired behaviors. The total task and joint space velocities can be derived from (3) and (4):

$$\dot{\mathbf{x}}_{tot} = \dot{\mathbf{x}}^* + \mathbf{J}\dot{\mathbf{q}}^* \tag{5}$$

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

with $\mathbf{J}$ the task Jacobian. Note that (5) and (6) 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 trajectory at the joint level in Ω and add some compliance to the control point by including the external force in $\mathcal{F}$.

Equations (5) and (6) must be both solved, to derive the set of *constraints* $\mathcal{C}$ that slow down the robot motion if needed (see Fig. 1). The constraints are scalar values $\mathcal{C}_i \in \mathbb{R}_{\geq 0}$ that become active when smaller than 1. In fact, they determine the value of *velocity scaling factor* $\alpha \in [0, 1]$:

$$\alpha = \min(1, \min(\mathcal{C})). \tag{7}$$

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

$$\dot{\mathbf{q}}_{con} = \alpha\ \dot{\mathbf{q}}_{tot}. \tag{8}$$

Equations (3) - (8) make up the OpenPHRI framework.

Note that, since multiple inputs can be enabled at the same time, they may not be all realizable. This can occur, for instance, if the robot is deviated 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.

4 Force inputs

This section presents joint or task space force inputs that, when included respectively in sets $\mathcal{F}$ in (3) and Γ in (4), make the robot comply with real world forces or react to virtual ones. An illustrative example is given in Fig. 2.

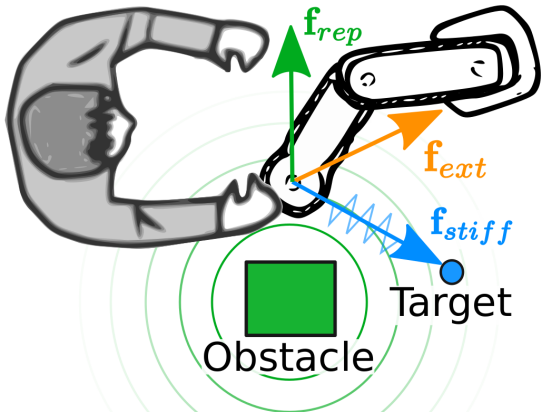


Figure 2: Examples of interaction, stiffness and repulsive forces.

4.1 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 $\mathbf{f}_{ext}$ 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 done in the joint space by including $\boldsymbol{\tau}_{ext}$ in Γ.

4.2 Virtual mass and stiffness

Using a full admittance model, including stiffness and mass effects in (1) or (2) has been intensively investigated in the literature and has been 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 generate forces along any motion direction. In the task space, for instance, these virtual forces will be respectively $\mathbf{f}_{t,stiff} = -\mathbf{K}_t\left(\mathbf{x} - \mathbf{x}_r\right)$ and $\mathbf{f}_{t,mass} = -\mathbf{M}_t\left(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_r\right)$, with $\mathbf{K}_t$ and $\mathbf{M}_t$ diagonal positive semi-definite matrices of stiffness and mass parameters, respectively.

4.3 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 of limited field of view sensors. The potential fields approach consists in modeling obstacles (respectively, targets) as sources of repulsive (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 $\mathbf{f}_{rep}$) to sets $\mathcal{F}$ or Γ.

5 Velocity inputs

In this section, we describe possible joint and task space velocity inputs, to be included respectively in

Ω and $\mathcal{V}$. These velocities can be the result of a trajectory generator (Sect. 5.1) or of a force control law (Sect. 5.2).

5.1 Velocity reference trajectory

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

- Each trajectory can have arbitrary initial and final: value, first, and second derivative (for a total of 6 degrees of freedom).
- Each trajectory can be composed of multiple segments (fifth-order polynomials) joined by intermediate waypoints, and that the waypoint the trajectory is C^3.
- Multiple trajectories, each composed of several segments, can be synchronized (see Fig. 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., increased for synchronization purposes).

Similar existing solutions, such as the Reflexxes Motion Library [6], do not provide bounded continuous accelerations, nor waypoints for complex trajectory design.

5.2 Velocities for force control

Force control is required for various applications, such as grinding, polishing, assembling, echographic monitoring, needle insertion or 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:

$$\Delta\mathbf{f} = \mathbf{S}(\mathbf{f}_r - \mathbf{f}_{ext}). \tag{9}$$

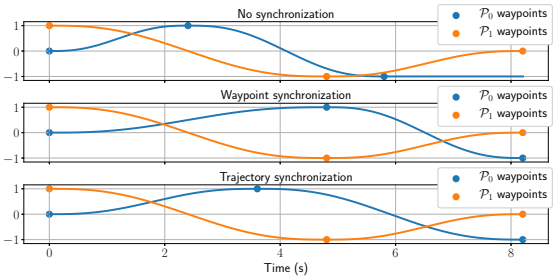


Figure 3: Trajectory synchronization. Given two trajectories, each composed of two segments (one intermediate waypoint), one can apply no synchronisation (top), waypoint synchronization (middle) and whole trajectory synchronization (bottom).

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, PD 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.

6 Constraints

In this section, we detail the design of the various constraints that reduce the robot velocity, through (7).

6.1 Emergency stop

A simple way to provide some level of safety, as demonstrated in [7, 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 $\mathcal{C}_{stop}$ to 0 as soon as $|\mathbf{f}_{ext}|$ (or $|\boldsymbol{\tau}_{ext}|$) passes some pre-tuned thresholds. Deactivation thresholds are also needed to specify when to increase $\mathcal{C}_{stop}$ to 1, using hysteresis. Fig. 4 gives the evolution of $\mathcal{C}_{stop}$, with activation and deactivation thresholds of 5 N and 1 N respectively.

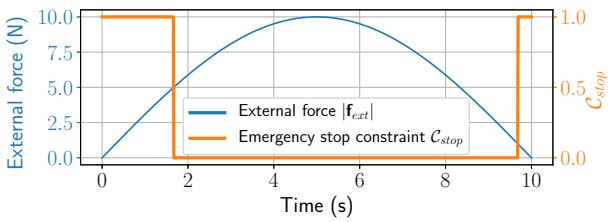


Figure 4: Characteristic of the emergency stop constraint. Activation threshold = 5 N, deactivation threshold = 1N.

6.2 Velocity limitation

Another very common safety criterion is velocity limitation. This is often present in robotics safety standards, as ISO 10218:2011 [10] and ISO/TS 15066. Note that even when the trajectory has been pre-planned to fulfill the velocity bounds, other control inputs (e.g., kinesthetic forces applied by the operator, force control, repulsive force) can lead to accelerations that break the constraint. To respect the limitation at all times, we design constraint $\mathcal{C}_{vel}$ 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 on this velocity limitation is given in Fig. 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 $\mathcal{C}_{vel}$ are displayed.

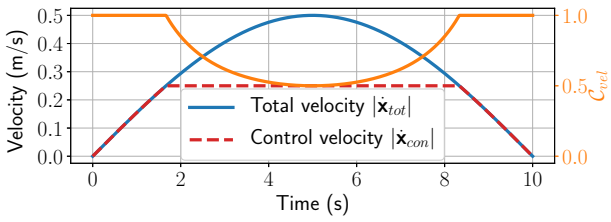


Figure 5: Evolution of the velocity with $V_{max} = 0.25$ m/s.

6.3 Acceleration limitation

To avoid abrupt robot motions, the acceleration can also be limited to A_{max}. Since (7) offers velocity reduction only, we use $A_{max} > 0$. To limit the acceleration, we express $\mathcal{C}_{acc}$ as $\mathcal{C}_{vel}$, by replacing, in the formulation, the current velocity with the predicted one, if the acceleration was at its maximum allowed value A_{max} during the next time step. The acceleration limitation mechanism is depicted in Fig. 6. It can be seen that the velocity increases linearly during the first 3s, due to the acceleration limit.

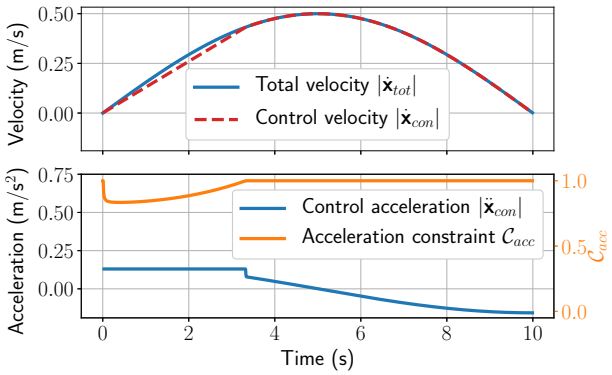


Figure 6: Evolution of the velocity and acceleration with A_{max} =0.13 m/s^2.

6.4 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 hardware level, e.g. electric power as with the Kuka LWR4+, or at 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 $\mathcal{C}_{pow}$, we consider 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 for him/her (see Fig. 7). In this case, $\mathcal{C}_{pow}$ is in-

versely proportional to the power, and 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 Fig. 8. It can be seen that the transmitted power is effectively limited only when it is negative and passes the limit.

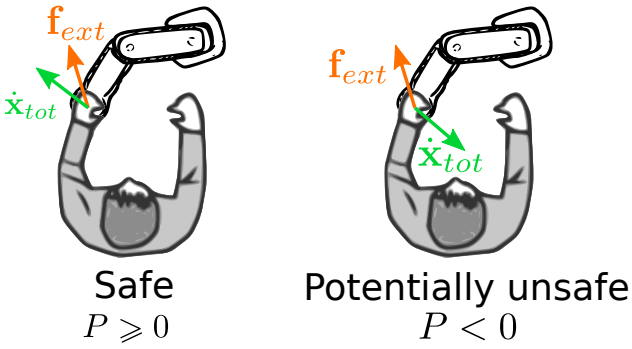


Figure 7: Safe and unsafe situations depend on the sign of the transferred power.

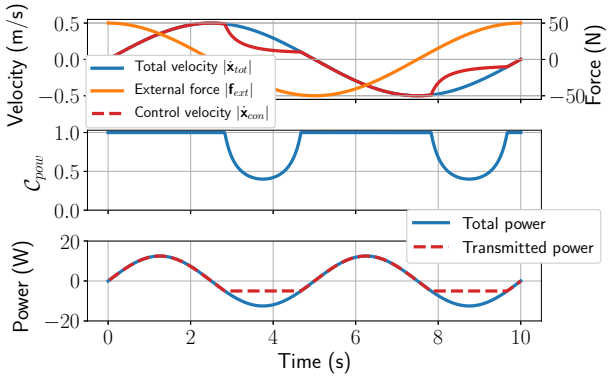


Figure 8: Evolution of the velocity, external force and transmitted power with $P_{max} = 5$ W.

6.5 Force limitation

Force limitation is the third and last constraint imposed by the ISO 10218:2011 and ISO/TS 15066. Actual force limitation is a very challenging problem, since it requires a complete knowledge of the environment, including present humans. While for the environment a complete map (position, materials, etc.) can be obtained, it is nearly impossible to have the same knowledge about humans, as this would require estimation of their motion and body impedance parameters, which change over time (e.g. fatigue or muscular co-contraction may stiffen a joint) and from a person to the other. Hence, we decided to adopt a reactive approach that does not rely on an environment model. By doing so, if the external force instantaneously passes the limit $F_{max} > 0$, the robot reacts to quickly move away from the impact and reach a safe state. Our approach has two steps. The first step consists in generating a velocity in the direction opposite to the external force, to move away from the collision. This velocity, noted $\dot{\mathbf{x}}_{F_{lim}}$, is added to set $\mathcal{V}$. The second step consists in including one or more constraints in $\mathcal{C}$, to slow down the robot, according to (7) and guarantee that it behaves safely while executing $\dot{\mathbf{x}}_{F_{lim}}$. For example, to respect the ISO 10218:2011, both velocity and power limitations must be applied:

$$\mathcal{C}_{force} = \min(\mathcal{C}_{vel}, \mathcal{C}_{pow}). \tag{10}$$

6.6 Kinetic energy limitation

When robot and human collide, the level of injury endured by the latter can be related to the robot 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 joints 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 scale factor. As such, constraint $\mathcal{C}_{kin}$ is equivalent to $\mathcal{C}_{vel}$, with $V_{max} = \sqrt{2E_{kmax}/m}$.

6.7 Separation distance

When the separation distance between the robot and near operators is monitored, it can be used to adapt the aforementioned limits (e.g., on velocity, power).

It is also required to fullfil the *Speed and separation monitor* mode of the 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 working near or in collaboration with humans. A simple example is depicted in Fig. 9. To comply with this, we use fifth-order polynomials, implemented in OpenPHRI[5], 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 (for instance, V_{max}) will vary from a low value at a fixed minimal distance, to a large value at a higher distance, and vary smoothly in between.

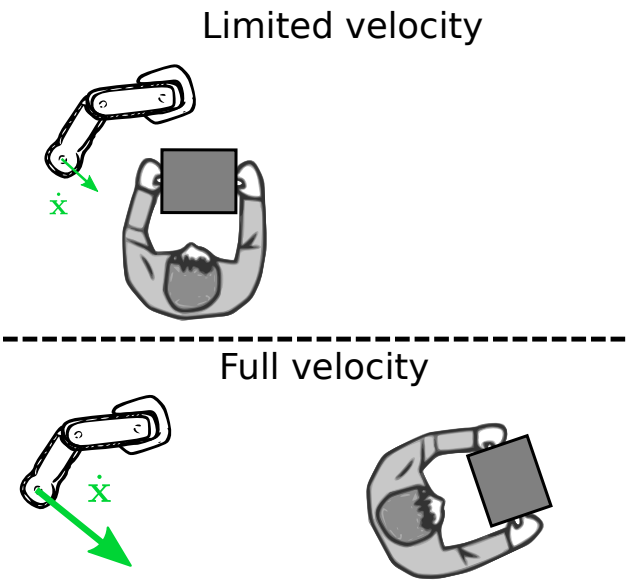


Figure 9: The velocity limitation varies smoothly in function of the separation distance.

7 Benchmark tests

In physical human-robot interaction, in order 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 minimum 1kHz. It is therefore crucial that the implementation of our controller in OpenPHRI is fast enough to comply with this timing constraint. To assess the performance of our library, we have run some benchmark tests on a computer equipped with an Intel i7-6700HQ @ 2.6GHz running Linux 4.11.

In Fig. 10, we present the results of the benchmark tests for the controller associated with different constraints, force and velocity inputs, and running on a 7 degrees of freedom manipulator. At each iteration, the controller is run 10000 times to get meaningful results, and the average computation time is logged. In Figures 10a-10e, we give the average computation time $\bar{t}$ and the standard deviation σ over 1000 iterations. The computation of the forward and inverse kinematics is not included in these results, in order to focus on the control computation time overhead. Also, the current controller implementation is single-threaded but, given the very low computational time ($\bar{t} < 4\mu s$ in the most complex scenario presented in 10e), a multi-threaded version does not seem necessary. Figure 10f shows that the memory usage[6] stays very low, with a peak at 186 KiB (here, the abscissa indicates snapshots taken regularly during execution).

8 Experiments

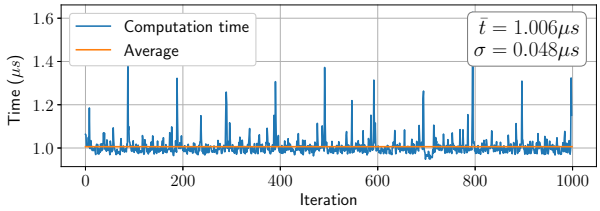
Let us now present the results of a full-featured experiment using the framework described in this chapter.

The experiment is split in two phases, a *teaching-by-demonstration phase*, and a *replay phase*, where the robot operates autonomously, in the presence of an obstacle and near the human operator. Figure 11 shows the setup, consisting in a Kuka LWR4+ arm, with external force $\mathbf{f}_{ext}$ estimated through the FRI interface[7]. All the code was written in C++ using the OpenPHRI library and integrated inside the Knowbotics Framework, currently under development at LIRMM, to interface with the hardware. The FRI library was used to communicate with the Kuka arm. The controller sample time was T=1 ms. To manage

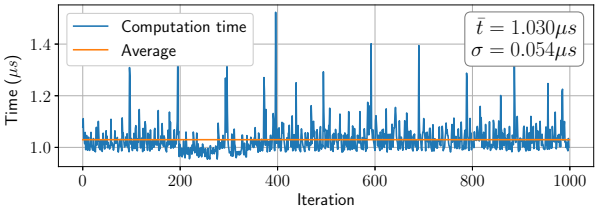
[5] OpenPHRI/utilities/fith_order_polynomials.{h,cpp}

[6] Measured using the Massif tool from the Valgrind software.

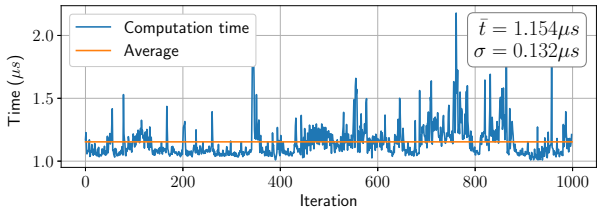
[7] `http://cs.stanford.edu/people/tkr/fri/html/`



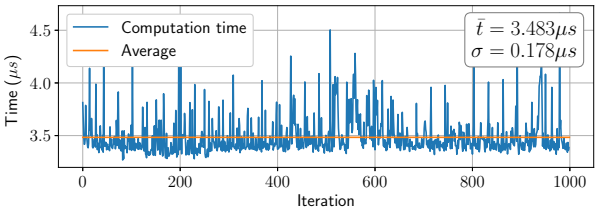
(a) Controller with no constraints and no inputs.



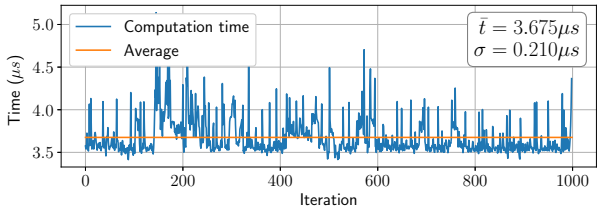
(b) Controller with a velocity constraint ($\mathcal{C} = \{\mathcal{C}_{vel}\}$).



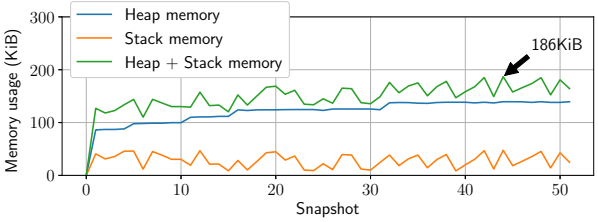
(c) Controller with velocity and power constraints ($\mathcal{C} = \{\mathcal{C}_{vel}, \mathcal{C}_{pow}\}$).



(d) Controller with velocity, power and kinetic energy constraints ($\mathcal{C} = \{\mathcal{C}_{vel}, \mathcal{C}_{pow}, \mathcal{C}_{kin}\}$).



(e) Controller with velocity, power and kinetic energy constraints and with a potential field, a virtual stiffness and a force controller in the task space ($\mathcal{C} = \{\mathcal{C}_{vel}, \mathcal{C}_{pow}, \mathcal{C}_{kin}\}$, $\mathcal{F} = \{\mathbf{f}_{t,stiff}, \mathbf{f}_{rep}\}$), $\mathcal{V} = \{\dot{\mathbf{x}}_{fc}\}$.



(f) Memory usage while executing the controller benchmarks ((a)-(e)) sequentially.

Figure 10: Benchmarks of the controller running on a 7 degrees of freedom manipulator.

the robot behavior, we designed in OpenPHRI the finite state machine shown in Fig. 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"[8]. The whole application has less than 600 lines of code: 125 for the main file and 440 for the finite state machine (header + source).

[8] `https://github.com/BenjaminNavarro/OpenPHRI/tree/master/apps/demo`

The *teaching phase* consists in manually guiding the robot, by applying $\mathbf{f}_{ext} \in \mathcal{F}$, to teach it the waypoints where it should later realize a force control task (apply $f_r = 30N$ 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 more seconds.

Once the operator has specified all the desired points, the *replay phase* is triggered. The trajectory generator is used to output the control point

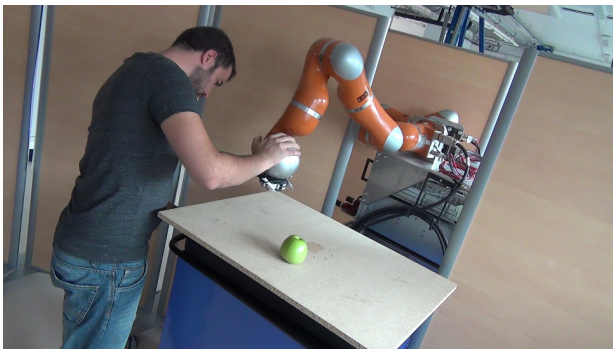


Figure 11: Setup for the experiment.

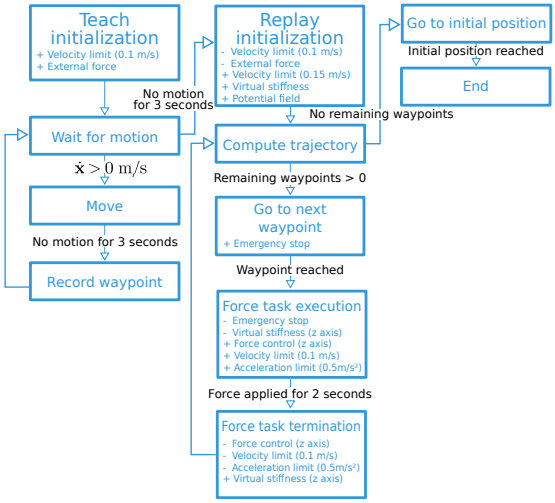


Figure 12: Finite state machine used for the experiment. A + sign indicates an addition to the controller (new constraint or new input) while a - indicates a removal.

(end effector) reference velocity ($\dot{\mathbf{x}}_r \in \mathcal{V}$) for reaching each waypoint. When a waypoint is reached, the task space force controller is activated ($\dot{\mathbf{x}}_{fc} \in \mathcal{V}$). After the force has been correctly applied, the robot moves to the next waypoint. Once all the force control tasks have been performed, the robot returns to its original position using the trajectory generator. 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, as explained in Sect. 6.1. Motion is resumed only when the external force is lower than 1 N. Additionally, potential fields ($\mathbf{f}_{rep} \in \mathcal{F}$) are used to avoid a known object (here, an apple) in the environment.

Throughout the experiment, the joint velocities sent to the robot are output by (8), with scaling factor α computed with the constraints in (7). 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 ($\mathcal{C}_{acc} \in \mathcal{C}$) of $A_{max} = 0.5$ m/s^2 is applied, to avoid abrupt motions. During the replay phase, a virtual stiffness $\mathbf{K}_t = \text{diag}(1000, \ldots, 1000)$, described in Sect. 4.2, is added to compensate 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 Fig. 13 while the results are shown in Fig. 14. A video of the experiment is attached to this paper[9]. The teaching phase takes place during the first 36 s. Then, the replay phase starts. It can be seen from Fig. 14c that the scaling factor is decreased multiple times to comply with the constraints. For example, during manual guidance, the applied external forces, visible in Fig. 14a, would have led to velocities above the limit if $\mathcal{C}_{vel}$ was not present. The same occurs when the obstacle is being avoided between seconds 73 and 74. At 70 s, as Fig. 14a and snapshot 13g show, 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 Fig. 14b and 14d. Finally, Fig. 14e shows how scaling from the total ($\mathbf{v}_{tot}$) to the applied ($\mathbf{v}_{con}$) translational velocity norms, complies with the imposed limit V_{max}.

[9] And also available at: `http://bit.do/openphrivideo`

(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 robots 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 13: Snapshots of the experiment: teaching (a-c) and replay (d-h) phases.

9 Conclusions

This article introduces OpenPHRI, a new software library intended for physical human-robot interaction 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, hence not be fully realized. This behavior can be sometimes desirable (e.g., collision avoidance while following a trajectory) or unwanted, and caused by controller misconfiguration. Nevertheless, the constraints applied at the lower layer of OpenPHRI, guarantee safe robot behavior at all times. Aside safety, a real life experiment also demonstrates 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). Besides, OpenPHRI controllers can be executed at very high rates (>1kHz) or on low end machines, while still achieving excellent performances. Finally, the OpenPHRI library is easy to use, and 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, avoiding it to become outdated on the long term.

Acknowledgments

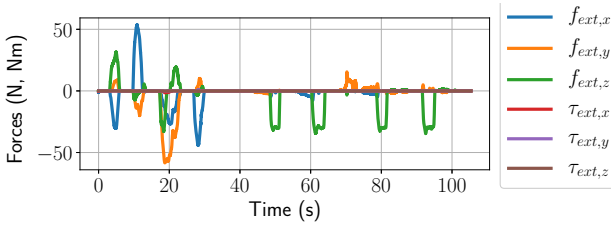
This work has been supported by the ANR (French National Research Agency) SISCOB project ANR-14-CE27-0016.

References

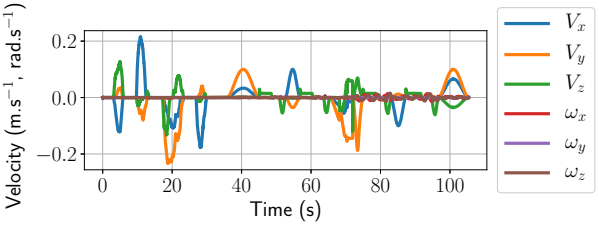
[1] N. Hogan. Impedance control: An approach to manipulation: Part II-Implementation. *Journal of Dynamic Systems, Measurement, and Control*, 107(1):8–16, 1985.

[2] S. Jung, T.C. Hsia, and R.G. Bonitz. Force tracking impedance control of robot manipulators under unknown environment. *IEEE Trans. on Control Systems Technology*, 12(3):474–483, May 2004.

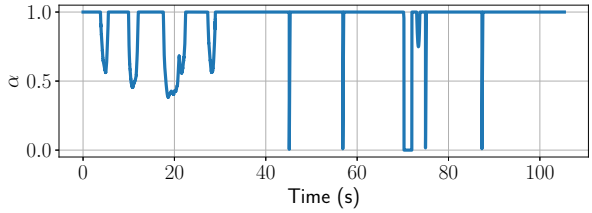
[3] F. Almeida, A. Lopes, and P. Abreu. Force-impedance control: A new control strategy of robotic manipulators. In *Recent Advances in*



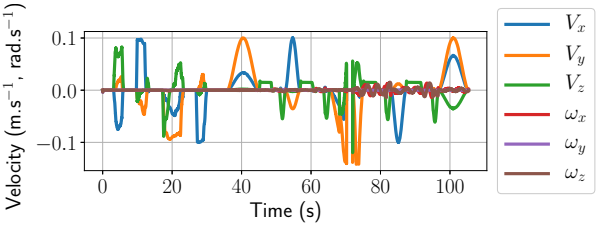
(a) Components of the external force $\mathbf{f}_{ext}$, applied by the human for teaching or upon collision (at $t = 70$ s), then by the robot during the four force control tasks.



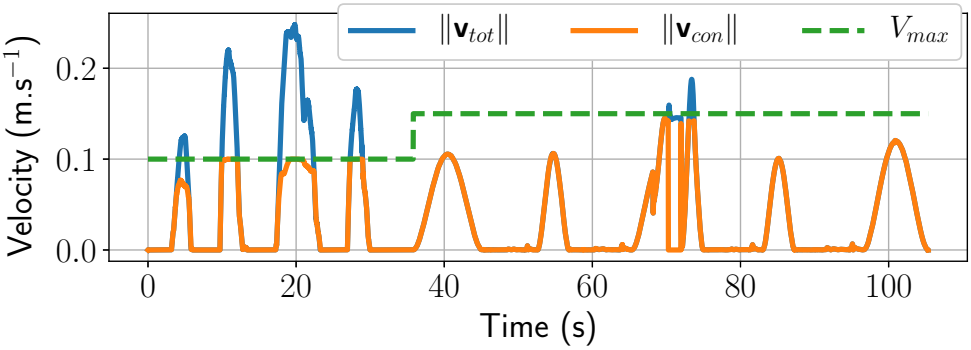
(b) Components of the control point total velocity $\dot{\mathbf{x}}_{tot}$.



(c) The scaling factor α, diminishing whenever the constraints are active.



(d) Components of the control point velocity applied after velocity reduction $\dot{\mathbf{x}}_{con} = \mathbf{J}\dot{\mathbf{q}}_{con}$.



(e) Comparison between the current velocity limit V_{max} and the total and applied translational velocity norms ($\|\mathbf{v}_{tot}\|$ and $\|\mathbf{v}_{con}\|$).

Figure 14: Relevant variables during the experiment.

Mechatronics, Springer, Singapore, pages 126–137, 1999.

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

[5] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, mar 1986.

[6] Torsten Kroger. Opening the door to new sensor-

based robot applications - the reflexxes motion libraries. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, may 2011.

[7] Y. Yamada, Y. Hirasawa, S. Huang, Y. Umetani, and K. Suita. Human-robot contact in the safeguarding space. *IEEE/ASME Transactions on Mechatronics*, 2(4):230–236, 1997.

[8] Alessandro De Luca and Fabrizio Flacco. Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration. In *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. IEEE, jun 2012.

[9] ISO/TS 15066:2016 robots and robotic devices – collaborative robots. Technical report, International Organization for Standardization, Geneva, Switzerland, 2016.

[10] ISO 10218-1:2011 Robot for industrial environments - Safety requirements - Part 1 : Robot. Technical report, International Organization for Standardization, Geneva, Switzerland, 2006.

[11] Sami Haddadin, Simon Haddadin, Augusto Khoury, Tim Rokahr, Sven Parusel, Rainer Burgkart, Antonio Bicchi, and Alin Albu-Schaffer. A truly safely moving robot has to know what injury it may cause. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, oct 2012.

[12] Anis Meguenani, Vincent Padois, and Philippe Bidaud. Control of robots sharing their workspace with humans: An energetic approach to safety. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2015.

[13] O. Khatib. Inertial properties in robotic manipulation: An object-level framework. *The International Journal of Robotics Research*, 14(1):19–36, feb 1995.