Task-Space Control Interface for SoftBank Humanoid Robots and its Human-Robot Interaction Applications

Anastasia Bolotnikova, Pierre Gergondet, Arnaud Tanguy, Sébastien Courtois, Abderrahmane Kheddar

Task-Space Control Interface for SoftBank Humanoid Robots and its Human-Robot Interaction Applications

Anastasia Bolotnikova[1,2], Pierre Gergondet[4], Arnaud Tanguy[3], Sébastien Courtois[1], Abderrahmane Kheddar[3,2,4]

***Abstract*— We present an opensource software interface, called `mc_naoqi`, that allows to perform whole-body task-space Quadratic Programming based control, implemented in `mc_rtc` framework, on the SoftBank Robotics Europe humanoid robots. We describe the control interface, associated robot description packages, robot modules and sample whole-body controllers. We demonstrate the use of these tools in simulation for a robot interacting with a human model. Finally, we showcase and discuss the use of the developed opensource tools for running the human-robot close contact interaction experiments with real human subjects inspired from assistance scenarios.**

I. Introduction and Background

Quadratic Programming (QP) task-space control has become a golden standard in humanoid robot control [1], [2], [3]. One of the most powerful software control frameworks that implements QP task-space control, called `mc_rtc`[1], is now publicly available. This framework provides developers with useful tools for writing complex controllers for either individual or multiple interacting robots to perform wide variety of experiments, e.g. in aircraft automation [4] or in physically interacting with humans [5], [6] that we technically explain and extend in this work.

In order to control any given complex robots with this framework, an interface must be developed to allow communication between the control framework and the robot's low-level controllers, sensors and devices. One example of such opensource interface is the `mc_openrtm`[2] software that is used to control Humanoid Robotics Project robots from Kawada Robotics [7]. However, for different types of robots a specific interface, as well as robot description and robot module, need to be developed in order to adapt to particularities of robot brand and low-level control strategies, devices and onboard operating system (OS). The idea of task-space control is to lie exactly between low-level control and high-level planning with somewhat adjustable frontiers.

In this work, we present the opensoure software interface, called `mc_naoqi`[3], that enables running the controllers implemented with `mc_rtc` framework on widely used SoftBank Robotics Europe (SBRE) humanoid robots [8], [9], running NAOqi OS and a customly developed local module `mc_naoqi_dcm` (Sec. II). Along with the interface, we opensource the robot modules and description packages that serve as a representation of SBRE robots in the `mc_rtc` framework (Sec. III). We also provide a basic sample Finite State Machine (FSM) `mc_rtc` controller for Pepper robot and its interaction with a human model in simulation (Sec. IV). Finally, we present our current instance use of the developed tools for running the `mc_rtc` controllers developed for Human-Robot interaction (HRI) experiments with human subjects (Sec. V).

Fig. 1 is an overview of the developed tools and their interconnection, described in detail in the rest of the paper.

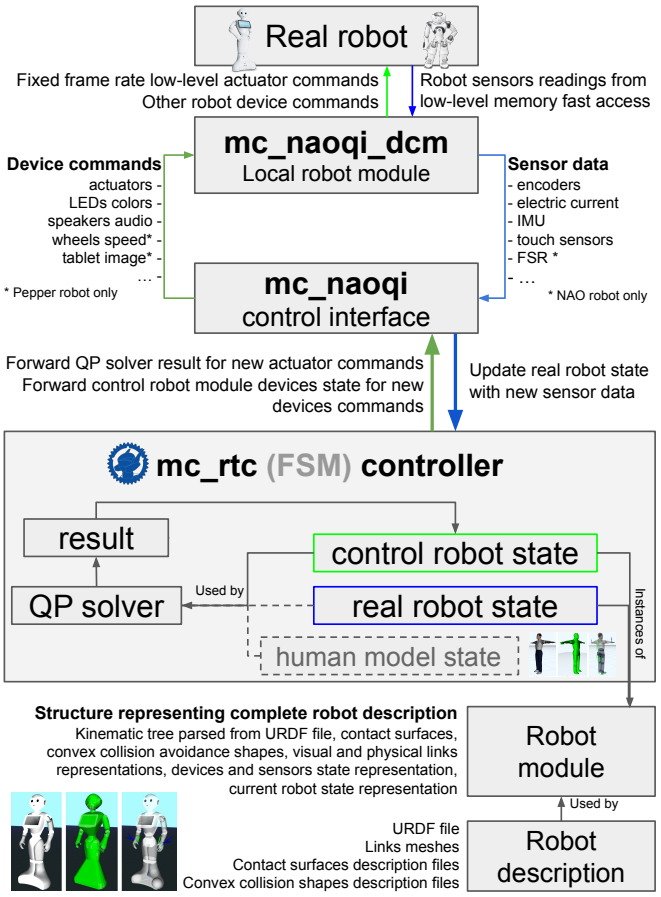


Fig. 1: `mc_naoqi` interface enables communication between SBRE humanoid robots and `mc_rtc` control framework. It can be used to steer the robot behaviour in HRI experiments.

[1]SoftBank Robotics Europe, Paris, France
[2]University of Montpellier–CNRS LIRMM, Interactive Digital Humans, Montpellier, France
[3]CNRS-AIST Joint Robotics Laboratory, IRL 3218, Tsukuba, Japan
[4]Beijing Advanced Innovation Center for Intelligent Robots and Systems, Beijing Institute of Technology, Beijing, China

[1]`https://jrl-umi3218.github.io/mc_rtc`
[2]`https://github.com/jrl-umi3218/mc_openrtm`
[3]`https://github.com/jrl-umi3218/mc_naoqi`

II. Control Interface

Figure 2 gives a schematic overview of the mc_rtc control framework and its connection to the simulation and control interfaces, such as mc_naoqi. Interested reader is invited to visit mc_rtc project website for more information, installation instructions and tutorials.

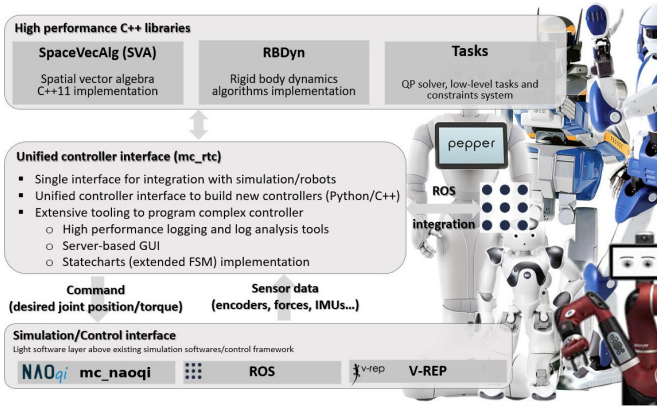


Fig. 2: mc_rtc control framework architecture.

Figure 1 illustrates schematically the role of the mc_naoqi interface as a communication layer between mc_rtc control framework and NAOqi OS running onboard SBRE humanoid robots, such as NAO, Pepper or Romeo. In the following subsections, we describe the entire system in detail.

A. Forwarding device commands from controller to the robot

The main role of the mc_naoqi interface is to forward the fixed frame rate control commands computed by the QP solver of mc_rtc whole-body QP controller to the onboard low-level robot actuators control. The decision variable of the mc_rtc QP controller is robot joint accelerations, which is integrated once to get desired velocity (e.g. for Pepper mobile base command), and then once more to get desired robot joint angles for joint actuator commands.

For social, user-friendly and interacting humanoid robots, such as Pepper or NAO, it is highly beneficial to endow mc_rtc controller with the functionality to also forward other device commands, such as sentence to play from the speakers, desired tablet screen image or eye led color, from the mc_rtc controller to the robot devices via mc_naoqi interface. This functionality allows mc_rtc framework users to develop controllers which can provide a richer interaction experience for HRI applications. For instance, when a certain mc_rtc controller FSM state is terminated, the robot can indicate this event (and that it is goint to transition to the next FSM state) with a comprehensive verbal message, illustrative tablet image and/or a specific LED color.

The mc_naoqi interface is thus developed with rich HRI consideration in mind, and allows mc_rtc controller to forward commands to all the robot devices, not only the joint actuators. We describe how it is implemented on the robot module side in detail in Sec III.

B. Forwarding sensor data from the robot to the controller

The mc_naoqi interface is also responsible for getting the most up-to-date sensor readings from a robot low-level memory in real-time and forwarding sensor measurements in a suitable form to the mc_rtc controller real robot state representation as a feedback. This way, the task-space QP controller keeps track of the real robot state and can use it to perform closed-loop QP control computations.

Besides common sensor readings, such as encoder values, force sensors or Inertial measurement unit (IMU) measurements, mc_naoqi interface also allows to forward from the robot to the mc_rtc controller such sensor readings as electric motor current and touch sensor readings (from tactile or bumper sensors). We describe how custom robot sensors are implemented on the robot module side in detail in Sec III.

For HRI applications, the touch sensor readings are especially beneficial to be forwarded to the controller, as they allow to detect when a human touches the robot. This signal can then be used inside the mc_rtc FSM controller, for instance, to trigger an appropriate reaction of the robot to the touch or to trigger a transition to a specific FSM state of the controller. We demonstrate this use-case in our sample HRI experiment, presented in detail in Sec. V.

C. Local low-level robot module

A customized local robot low-level module, called mc_naoqi_dcm[4], is cross-compiled for NAOqi OS and is set to run onboard the robot to read sensor values and set device commands via Device Communication Manager (DCM) every 12 ms, fastest update rate currently available for SBRE humanoid robots.

When a user connects to the robot via mc_naoqi interface, and commands to turn on robot motors, a preprocess function is connected to DCM loop, to start setting the actuator commands at a fixed time rate, synchronized with the other DCM operations. After the user commands to turn off robot motors via mc_naoqi interface, the preprocess actuator command update function is disconnected from the DCM loop. This way, the user can safely use other control applications, such as *Choregraphe*, with the robot right after using mc_naoqi interface, even though mc_naoqi_dcm module is still active on the robot.

In order to prevent the default high-level robot behaviours to interfere with the commands forwarded to the robot via mc_naoqi from the mc_rtc controller, the default robot safety reflexes are disabled when robot motors are turned on via mc_naoqi. Once the motors are turned off via mc_naoqi, the default robot safety reflexes are re-enabled. This has to be taken into account by an mc_rtc controller designer, to ensure that the developed controller is safe to be run on the real robot, through extensive testing in simulation.

A fast access to the low-level robot memory is initialized when mc_naoqi_dcm starts to run on the robot. This allows to read a predefined set of sensor values from robot memory in the fastest way.

[4]https://github.com/jrl-umi3218/mc_naoqi_dcm

III. Robot representation in mc_rtc framework

A. Robot description packages

To control any robot with mc_rtc framework, a basic description of this robot needs to be provided. Such robot description includes robot kinematic tree, dynamic properties of the links, description of robot contact surfaces (i.e. covers), and convex (eventually strictly convex [10]) anti-collision shapes (these shapes can be automatically generated from existing robot links graphic representation). Examples of these robot description elements for SBRE humanoids are shown in Fig. 3. With this work, we release these robot description projects, implemented as Robot Operating System (ROS) packages, for NAO[5] and Pepper[6] robots.

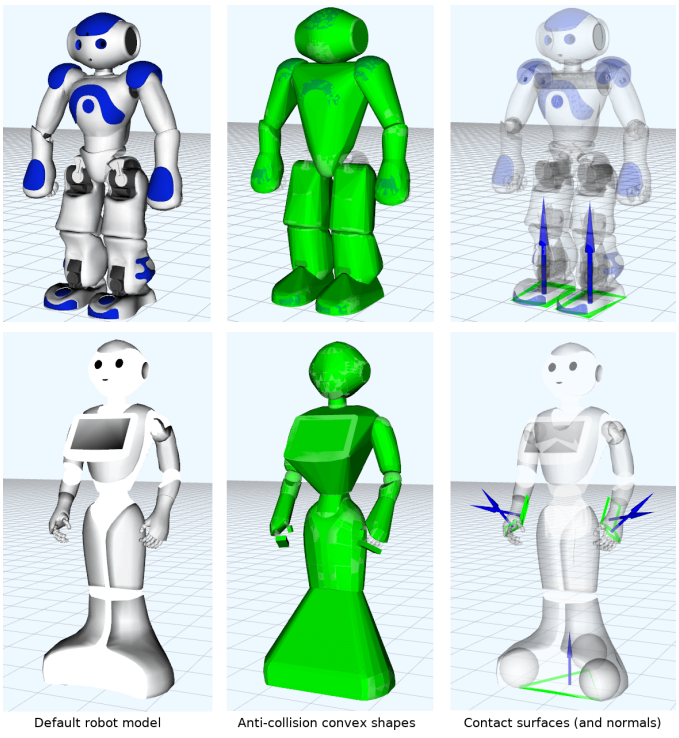


Fig. 3: NAO and Pepper robot descriptions for the robot representation in the robot module of the mc_rtc framework.

B. Robot mc_rtc modules

As illustrated in Fig. 1, the robot description packages are used by a *robot module* software layer to create a structure that provides a complete description of the robot: kinematic tree parsed from URDF files, visual and physical representations of robot links, surfaces attached to the robot bodies, sensors and other devices, strictly convex hulls and primitive shapes for collision avoidance, etc. The instance of this structure is used by the mc_rtc framework, as control robot state representation, to formulate the QP control problem (objectives and constraints), that is then passed to the solver to compute the next desired robot state.

We make the robot modules for NAO[7] and Pepper[8] publicly available with this work. For fast prototyping and experiments, the robot description package and module can easily be augmented with any new robot hardware elements, e.g. new onboard camera, which we demonstrate in Sec. V.

The Pepper robot module exploits a recently introduced new mc_rtc framework feature - generic robot devices. This feature enables developers to implement any kind of robot custom device representation as part of the robot module, which can then be used in the mc_rtc controller. Currently implemented devices in Pepper robot module are:

- *loud speaker*: to forward speech commands to the robot directly from the mc_rtc controller via mc_naoqi;
- *visual display tablet*: to set robot tablet image from the mc_rtc controller via mc_naoqi interface;
- *touch sensor*: to forward tactile sensor readings from the robot to the mc_rtc controller via mc_naoqi

Wheeled mobile base of Pepper is modeled as a floating base, constrained to move on a plane (i.e. the room ground), with limits imposed to the mobile base body maximum speed and acceleration according to physical hardware limitations.

In mc_rtc framework, two main hierarchies for the robot control are the following:

- *Tasks*: that are objectives 'describing' what the robot should do at the best it can;
- *Constraints*: that are limits under which the tasks are to be performed, i.e. what the robot should always fulfill strictly.

Many tasks and constraints are already implemented as robot-independent templates in mc_rtc. For instance *PostureTask*, *CoMTask*, *EndEffectorTask*, *KinematicsConstraint*, *ContactConstraint*, etc. However, in some cases it might be desirable to design and implement new custom tasks or constraints, not yet implemented in mc_rtc. Such new tasks and constraints might be specific to a robot, use-case or research topic. In the Pepper robot module mc_pepper, we provide an example of the implementation of a custom *CoMRelativeBodyTask* QP control objective. This task allows to specify the desired Pepper center of mass (CoM) target relative to the robot mobile base frame (as opposed to world frame in mc_rtc *CoMTask*). The implementation of this Pepper specific objective was necessary to allow controller to simultaneously compute new mobile base position and keep CoM objective as part of QP computations. An example of a Pepper specific QP constraint implementation is a custom *BoundedAccelerationConstr* constraint, included in the Pepper robot module. This constraint allows to impose acceleration bounds for Pepper mobile base link.

An example of how these custom Pepper robot specific tasks and constraints are loaded and used in a sample mc_rtc controller can be seen in *PepperFSMController* opensource project, which we describe in detail in Sec. IV. In an analogous way, many other novel tasks and constraints can easily be implemented and tested.

[5] https://github.com/jrl-umi3218/nao_description
[6] https://github.com/jrl-umi3218/pepper_description
[7] https://github.com/jrl-umi3218/mc_nao
[8] https://github.com/jrl-umi3218/mc_pepper

IV. Sample Pepper mc_rtc FSM controller

A. Individual robot controller

To facilitate the process of writing a new mc_rtc controller, especially for new potential users of the framework, we provide a basic sample *PepperFSMController*[9]. It can be used as a starting point for new controller development or as an example of how similar projects should be implemented.

The sample controller includes Pepper as the main controller robot, associated default posture task, kinematics and dynamics constraints, self-collision avoidance constraints, robot-ground contact constraint. The controller also includes implementation of few sample FSM states that allow to control robot posture, mobile base, camera orientation and right and left hand end-effector positions, either through predesined setpoints or interactively through RViz [11] (Fig. 4).

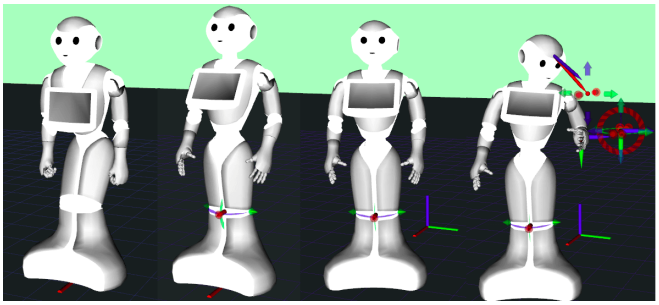


Fig. 4: RViz scenes of sample FSM Pepper controller states.

B. Controller for HRI including a human model

A direct extension of the master branch of the sample controller project, is a branch called topic/withHumanModel. On this branch, a multi-robot QP (MQP) control feature of the mc_rtc framework is exploited by adding a human model and its state as part of the mc_rtc controller (recall Fig. 1). A human model is integrated into mc_rtc exactly the same way that any other robot model, by providing a description ROS package[10] and implementing a corresponding robot module[11]. Fig. 5 shows the human model description used in our projects.

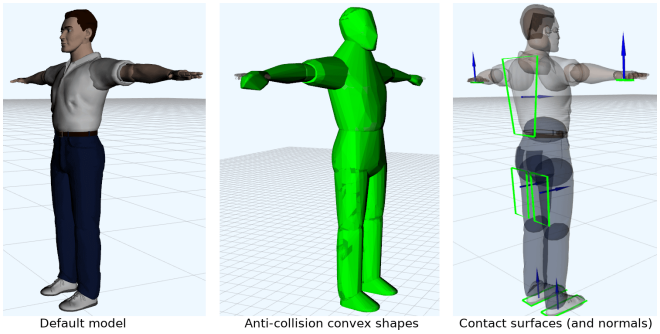


Fig. 5: Simulation human model used in mc_rtc framework.

[9] https://github.com/jrl-umi3218/pepper-fsm-controller
[10] https://github.com/jrl-umi3218/human_description
[11] https://github.com/jrl-umi3218/mc_human

This MQP controller can be used to develop and simulate a wide variety of HRI scenarios including pHRI ones, e.g. using human model and robot contact surfaces description to define contact tasks and constraints.

In the opensource sample MQP controller project, we provide an example of HRI controller state called *NavigateToHuman*. In this state, a Position Based Visual Servoing (PBVS) task of mc_rtc framework is used to control Pepper robot mobile base to navigate in closed-loop to a set-point defined w.r.t human model torso frame (it can be any other human model frame), while using simulation data as a virtual visual feedback signal. At the same time, Image Based Visual Servoing (IBVS) task is used to control robot camera orientation to look at the human head. Fig. 6 illustrates the simulation of the *NavigateToHuman* FSM state at the start, middle and the end of this state.

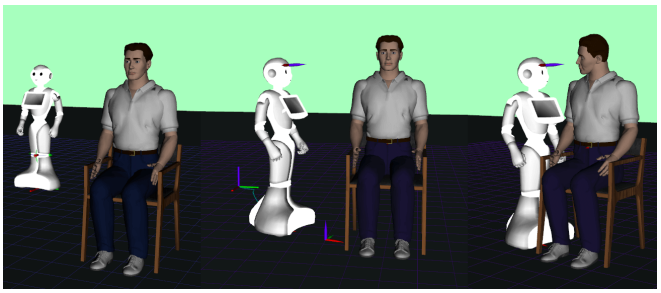


Fig. 6: Simulated *NavigateToHuman* state of a sample MQP mc_rtc FSM controller with a human model included.

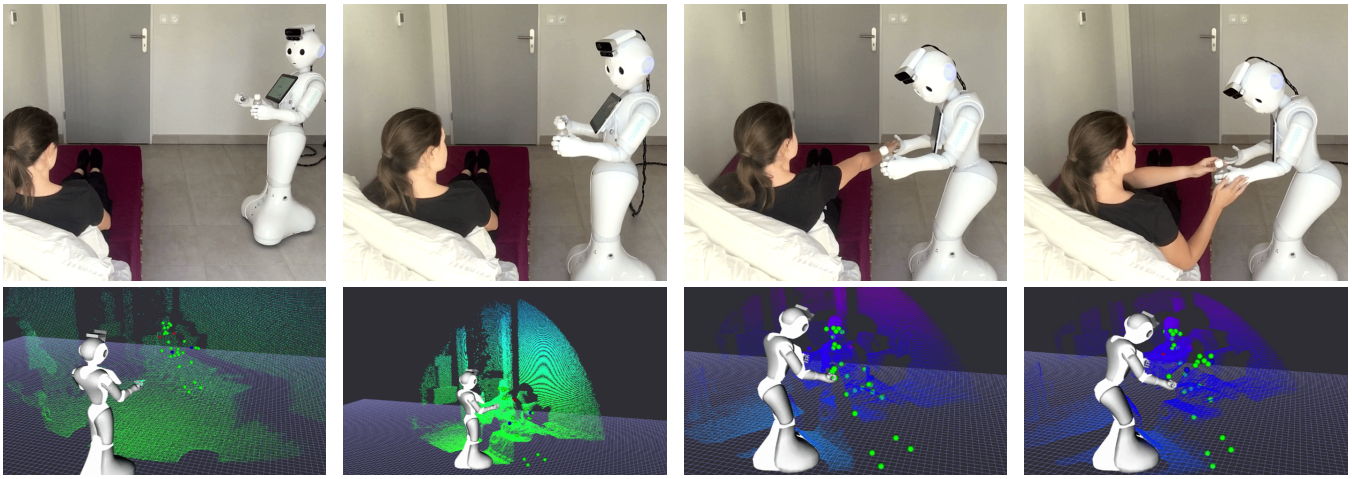
The sample controller project can also efficiently serve as a starting point for developing controllers for various real HRI applications, which we showcase in Sec. V

V. Experiments with real human subject

Recently, the Covid-19 pandemic has put attention into robotics, which can help to deal with the problematics of human virus transmission, by transferring some risky labours and non-added value tasks of care-givers to robotic systems. Examples come from different companies, like disinfecting ultraviolet (UV) rovers and ground drones for hospitals, or even beaming videos to connect patients to their relatives. There are more examples in other fields, like flagging patients with suspected pneumonia during their hospital admission, or using robots and AI in logistics to transport daily groceries.

In our recent publication, we have already demonstrated how the developed tools, presented in the current work, can be used to enable Pepper robot to perform autonomous initiation of human physical assistance [6]. We also made the controller code[12] publicly available to serve as a software reference to our work and as an advanced FSM mc_rtc HRI controller example. This controller shows how our developed software components allow to easily create complex controllers for rich, intuitive and efficient HRI. This includes closed-loop navigation toward human, verbal, visual and

[12] https://github.com/anastasiabolotnikova/autonomous_phri_init



(a) Navigation to the human (b) Verbal communication (c) Human takes the pills (d) Human takes the bottle

Fig. 7: Experiment screenshots (top), perceived robot state, Azure Kinect point cloud and human body markers (bottom).

body language communication, and physical interaction with a human for an assistance process initiation.

In the present work, we develop another HRI application –inspired from the Covid-19 outbreak, where Pepper robot behaviour is regulated, with the help of the `mc_naoqi` and the developed robot modules, to autonomously deliver medication to a human lying on the bed. In this scenario a bed height conforms that of Pepper for the given tasks.

The aim here is also to demonstrate how the controllers for new HRI scenarios can be easily prepared using our developed tools and re-using states and elements of the controllers written for other HRI scenarios. As such, for instance, the navigation toward a human state could be re-used from our previous work in [6], for this new scenario with almost no modifications, despite the person is lying in the bed instead of sitting on a chair. Other parts of this new controller FSM also needed only slight adjustments w.r.t our existing work (states present in other HRI controller) to create a controller for this new HRI application.

Fig. 7 shows excerpts screenshots from the experiment video. In the bottom row images, the scenes are visualized in RViz. Note, that additional hardware, namely Azure Kinect, which is used for human state feedback, and RealSense camera, which is included in the robot prototype, but not used in this experiment, are easily included in the robot description (Fig. 8) and processed by the robot module, and therefore are also included in the QP problem formulation (e.g. for more accurate robot center of mass computation). Full experiment can be seen in the accompanying video.

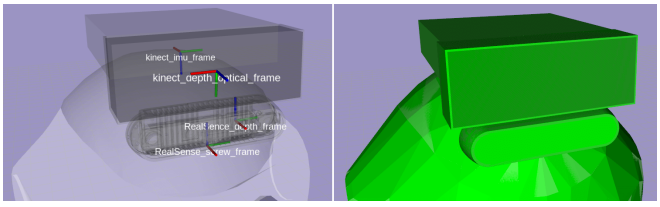


Fig. 8: Extra hardware included in the prototype Pepper robot

Once the robot reaches a position nearby the person, it communicates verbally its intention to pass the medicine (Fig. 7b). Then it proceeds to open its right gripper for the person to take the pills (Fig. 7c). The passing of the bottle with liquid is arranged with the help of the robot module feature, described in Sec. III, that allows to forward robot tactile sensor data to the `mc_rtc` controller, which then triggers robot left hand gripper to open slightly to allow the person to get out the bottle more easily (Fig. 7d).

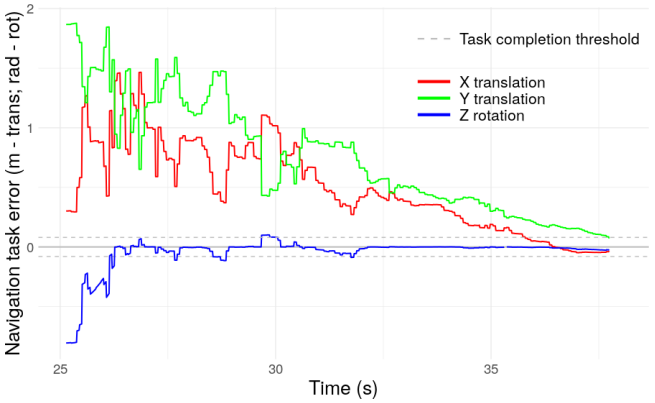


Fig. 9: Evolution of Position Based Visual Servoing closed-loop Pepper mobile base navigation to human task errors

Fig. 9 shows the progress of the closed-loop PBVS task, that uses Azure Kinect body tracking for feedback to make the robot navigate to the person (Fig. 7a). Task errors eventually converge near zero, although in a not very smooth way. This is due to the low quality of human detection (that prohibits constant usage in a continuous closed-loop way), and the low detection frame rate and latency issues (that limits the speed in reaching the person). Indeed, the existing human motion trackers are not robust enough for robotic usage. It is important to underline some shortcomings that prohibit current spreading of HRI in real practice:

- reliable human posture and body detection algorithms

(even those advanced) often fail in robotics because of the usage perspective. In assistive robotics scenarios, it is the robot which moves toward a human that can be static (i.e. not moving much), e.g. lying in a home or hospital bed or sitting on a chair. Most of the human pose detection algorithms are trained with rather static camera and moving human and not with a moving camera and static (or moving) human. We have witnessed a considerable effects on the robustness of the human pose acquisition by a moving robot with all the algorithms we tried. As a consequence, it is difficult to use them continuously in closed-loop control.

- lack of ground-truth: most of the human pose detection matches well in an augmented-reality, their image or video counterpart, but no one provides measurement ground-truth concerning the pose results that are returned. That is to say, it is difficult to assess the precision of the 6D pose returned by most algorithms with a ground-truth 6D measurements. In robotics, it is important to know precisely and in real-time the exact joint and floating base values of the human posture and configuration, namely when it comes to contact a person or to manipulate a given limb of a person [6].
- last but not least, as it is a most critical issue: close-contact interaction with human (i.e. when the robot reaches a person) tracking is lost even when wide fish-eye is used. This becomes more critical when manipulating a human will generate obstructed view due to the robot body. This means that human pose estimation approaches in the framework of human-robot close contact interactions have to be deeply reviewed.

This being said, the approach we adopt is rather modular enough to live with such shortcoming without recalling into question the controller. Any improvement in human pose estimation w.r.t the performances mentioned previously will straightforwardly result in better robustness and performance of the controller, because of its conceptual simplicity.

VI. Conclusion

For robotics to gain more insight, trust and meet the challenge of future human societal stakes, such as home daily assistance for frail or elderly persons, or to be efficiently used in future outbreaks, research efforts shall be paired with important integration development ones. Any advances made in critical human-robot interaction technological bricks such as human perception, artificial intelligence chatbots, 5G, advanced SLAM... should be integrated in a readily available and sustained task-space control frameworks. This is the very reason of this work: we do not only aim at sharing an opensource code for the robotic community in general and the HRI in particular, but also sharing experiences that led to existing developments which are made to be further used, improved and sustained. Our methodology in terms of control can be seen now under a different philosophy.

In computer science, algorithms, simple instructions such as if-then-else, while-for loops, variables... together with basic well-agreed routines and functions constitute the basic bricks of any modern algorithm that solve more and more complex problems. Our robot control framework should be seen under this spectrum: we shall provide elementary controllers and controller "routines", templates that form the common "instructions", and functions of more complex controllers that solve more and more complex tasks. In this paper, we thoroughly exemplify what `mr_rtc` may bring under a unified framework in terms of task specification, embedding straightforwardly the constraints... how multi-(sensory,objectives,robots) task-space controller can be used to build sustainable controllers, and show that what we propose is consistent, as all these tasks are defined exactly the same way for any robot or multi-robot system. We hope that users of SBRE robots could assess, enrich and use our developments in even more complex scenarios.

In future work, we are aiming to use this framework to conduct *in-situ* assistance scenarios in a retirement house with few real patients. This needs to be complemented with considerable software engineering efforts.

Acknowledgment

We thank Sébastien Barthelemy and Sébastien Dalibard from SBRE for providing useful information for this work.

References

[1] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[2] R. Cisneros, M. Benallegue, M. Morisawa, and F. Kanehiro, "Qp-based task-space hybrid/parallel control for multi-contact motion in a torque-controlled humanoid robot," in *IEEE-RAS 19th International Conference on Humanoid Robots*, pp. 663–670, IEEE, 2019.

[3] K. Bouyarmane, K. Chappellet, J. Vaillant, and A. Kheddar, "Quadratic programming for multirobot and task-space force control," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 64–77, 2019.

[4] A. Kheddar, S. Caron, P. Gergondet, A. Comport, A. Tanguy, C. Ott, B. Henze, G. Mesesan, J. Englsberger, M. A. Roa, *et al.*, "Humanoid robots in aircraft manufacturing: The airbus use cases," *IEEE Robotics & Automation Magazine*, vol. 26, no. 4, pp. 30–45, 2019.

[5] K. Otani, K. Bouyarmane, and S. Ivaldi, "Generating assistive humanoid motions for co-manipulation tasks with a multi-robot quadratic program controller," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3107–3113, IEEE, 2018.

[6] A. Bolotnikova, S. Courtois, and A. Kheddar, "Autonomous initiation of human physical assistance by a humanoid," in *IEEE International Conference on Robot and Human Interactive Communication*, (Naples, Italy), 31 August–4 September 2020.

[7] H. Hirukawa, F. Kanehiro, K. Kaneko, S. Kajita, K. Fujiwara, Y. Kawai, F. Tomita, S. Hirai, K. Tanie, T. Isozumi, K. Akachi, T. Kawasaki, S. Ota, K. Yokoyama, H. Handa, Y. Fukase, J. ichiro Maeda, Y. Nakamura, S. Tachi, and H. Inoue, "Humanoid robotics platforms developed in hrp," *Robotics and Autonomous Systems*, vol. 48, no. 4, pp. 165–175, 2004.

[8] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "Mechatronic design of nao humanoid," in *IEEE International Conference on Robotics and Automation*, pp. 769–774, IEEE, 2009.

[9] A. Pandey and R. Gelin, "A mass-produced sociable humanoid robot: pepper: the first machine of its kind," *IEEE Robotics & Automation Magazine*, vol. 25, no. 3, pp. 40–48, 2018.

[10] A. Escande, S. Miossec, M. Benallegue, and A. Kheddar, "A strictly convex hull for computing proximity distances with continuous gradients," *IEEE Transactions on Robotics*, vol. 30, pp. 666–678, June 2014.

[11] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, "Rviz: a toolkit for real domain data visualization," *Telecommunication Systems*, vol. 60, no. 2, pp. 337–345, 2015.