POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Modular software architecture for assistive and rehabilitative exoskeletons: a proof of concept

Laurea Magistrale in Biomedical Engineering - Ingegneria Biomedica

Author: Dario Comini, Daniele d'Arenzo

Advisor: Prof. Marta Gandolla

Co-advisors: Stefano dalla Gasperina, Mattia Panzenbeck

Academic year: 2021-22

Introduction

Over 1 billion people in the world are estimated to live with some form of disability, about 15% of the global population [10]. People with disability experience poorer health outcomes, have less access to education and work opportunities and are more likely to live in poverty than those without a disability [10]. There are two main interventions to contrast the impact of disability in daily life: rehabilitation and/or assistance.

Rehabilitation is a field that favors the development of the potential of people with disability and its main goal is the recovery of a physiological function. When coming to motor rehabilitation, the effectiveness of a rehabilitation treatment and the consequent motor relearning depend on several factors [5], such as repeatability of the exercises, intensity and dosage of the training, adaptation of the treatment to the patient (needs, status, recovery stage), customization of the therapy and individualization, direct involvement of the patient and motivation.

The purpose of assistance is complementary, as it regards the support in activities of daily living [8]. This introduces another important topic in the context of disability: autonomy. The lack of assistive services can make people with disability overly dependent on family members and caregivers, causing economical and social inclusion difficulties[3].

The continuous increase of people with disabilities results in higher requests for rehabilitation and assistance [10]. Although the help of specialists and caregivers for the care of people with disability is always needed, the workload and cost required for rehabilitation and assistance can be reduced through technological support. Assistive technology has been demonstrated to significantly improve the ability to perform activity of daily living [3] alleviating the issue of lacking autonomy. Rehabilitation can benefit of technology as well, since it allows to assess patients' performances more accurately than standard therapy, while providing a potentially adaptable and repeatable environment [5]. Particularly for motor disabilities, rehabilitative and assistive robots are increasingly integrated into clinics [12] and industrially produced. In fact, robots can be designed in order to adapt to the patient needs while providing an effective treatment and support.

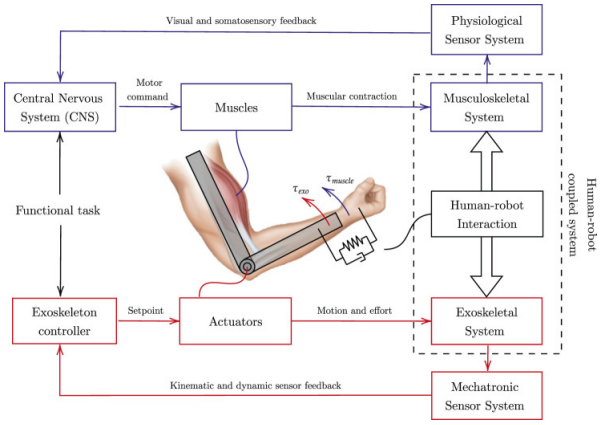

Figure 1: Human-exoskeleton system [5]

In medical robotics, exoskeletons are of particular interest, since they enable users to perform an action with their own body. An exoskeleton system is a complex human-robot coupled system (Figure 1) in which the wearer and its exoskeleton interact both physically (physical human-robot interaction) and cognitively (cognitive human-robot interaction) [4].
There are many elements that compose an exoskeleton system, depending on how much complexity is considered in the design phase.

The hardware of an exoskeleton system is mainly composed by actuators, sensors, passive components and transmissions. Actuators are the effectors of the system, and the means by which exoskeletons are able to actively move. Sensors constitute the perceptual interface between the exoskeleton and its environment. The selection of hardware elements can be highly user-specific [4], depending on users' biomechanics, financial resources and even their preference.

To make the exoskeleton perform as expected, control solutions are implemented. Controllers represent one of the cores of a robot: they enable the system to compensate for errors and also act as an interface between physical hardware and higher-level functions, such as planning. Many different controllers are used in rehabilitation and assistance, both at low and high level, and their implementation should adapt to the user progress [5].

User interfaces and intention-detection strategies are other crucial elements of an exoskeleton. Interfaces enable users to interact with the system, while intention detection strategies represent what is used by the exoskeleton control core to derive the willingness of the user to perform a task [7]. Both these functionalities come with a great variety of alternatives, depending on user capabilities and preferences.

Finally, higher-level control systems are necessary for more complex functions. One example is motion planning and execution of planned trajectories in the workspace of the exoskeleton, which are useful for repeated tasks like eating or rehabilitation exercises [11][9]. However, when possible, users often prefer to directly control their actions, and this can be achieved by coupling intention detection mechanisms and end-effector or single joint control. These functionalities require kinematic algorithms to pass between an environmental or user-centered reference system and a joint-space reference (configuration space).

To merge all these functionalities, a complex architecture is required, often built on top of a specific robotic platform. Architectures for human-robot systems are usually considered fixed and implemented as a whole for the particular scenario. However, this design practice can load the development process, making it hard to use already implemented solutions or to select technologies from different manufacturers. Completely custom solutions may also lower the compatibility of the system with mainstream and widespread technologies for people with disability, like smartphones and the internet. Moreover, both in rehabilitation and assistance, an high level of customization of the human-robot system is needed to adapt or update a previously implemented functionality to the evolution of user needs.

Aim and structure of the work

The purpose of this work is to provide a proof of concept for the construction of a modular and adaptable software architecture, in particular addressed to an upper-limb assistive exoskeleton.
First, the requirements of such architecture will be described (Materials), focusing on how they can be accomplished using standard and

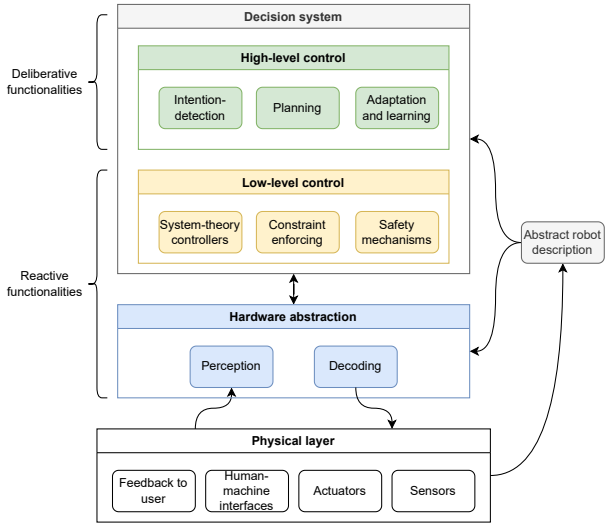

Figure 2: Robotic system architecture

widespread robotic frameworks. The architecture will then be implemented (Methods) using the described materials, and each of its elements will be tested to assess its modularity, both independently and collectively, using a test-bench and the BRIDGE exoskeleton (Results).

Materials

One of the cores of a modular architecture is the software: a software architecture is a methodology to structure, organize and assemble all elements that compose a system. A software architecture for robots interacting with people has many requirements: i) it must be able to provide an abstract description of the physical world of the exoskeleton system, ii) to connect hardware with different characteristics, iii) to switch between different control modalities, iv) to provide planning functionalities, v) to be able to implement several different user interfaces and intention - detection mechanisms, and vi) to be able to easily allow adaptation to different and changing user's conditions. The architecture should allow to run all these components independently, in parallel and asynchronously. That's why, rather than a simple messaging protocol, a robust high level framework has been selected: ROS2.

ROS is an open-source collection of libraries, drivers and tools for building robot applications [6]. Each single module of the ROS infrastructure constitutes a node: nodes are processes that perform computation and that can be dynamically reconfigured by means of parameters. Nodes communicate with one another through ROS interfaces, that work by exchanging messages.

The general software architecture can be divided into different elements and layers of abstraction (Figure 2).
First, a robot description is required to generalize the physical model of the robot (Figure 2, on the right), providing information that can be interpreted at all levels of the architecture – from low-level to derive the single-joint specifications, to high-level where kinematic chains, joint groups or visual information are used. In ROS2, the description is arranged using the Unified Robotic Description Format (URDF), which consists of a set of link and joint elements with kinematic and dynamic properties, geometrical information and collision models.

At the lowest level of the software architecture, the hardware abstraction layer – implemented by the ROS2 control framework – translates information at the control level to hardware instructions and vice-versa (Figure 2, in blue). The hardware abstraction consists of several abstract components running in parallel: each component interacts with a set of hardware elements that must be made compatible with the architecture, ensuring hardware modularity.

To achieve modularity and adaptation of the control modality, also controllers' implementation should be system-independent (Figure 2, in yellow). Controllers in ROS2 communicate with the hardware abstraction layer through specialized interfaces. Internally, they perform the operations that implement the required transfer function (e.g., PID control) between inputs and outputs, as in a theoretical control system. A controller manager node synchronizes the different controllers by maintaining a unique clock that determines the control-loop update frequency; also, it manages the controller access to hardware interfaces.

The higher-level control functionalities (Figure 2, in green) like planning and kinematic transformations are provided by MoveIt2 [1],

which is compatible with ROS2 out of the box. MoveIt functionalities include local minima avoidance for planning, kinematic solvers, collision and singularities checking and time-parameterization of trajectories.
MoveIt2 architecture is composed by a central node which can access environment characteristics and its constraints, but can be extended with plugins thus giving modularity to the architecture.
The first high-level control modality is offline planning, that is performed before the movement execution. To demand an offline planning, a motion request is sent through user interfaces (GUIs or external hardware inputs). The motion plan request includes planner preference and parameters, motion constraints and final pose of the robot. Offline planning is implemented with a specialized node which combines a path with planner adapters; the result is a trajectory – a path bound by specific time constraints, joints velocity and acceleration for each way-point (single trajectory points in the joints space).
Another high-level control method is online planning, which attempts to solve dynamic planning problems during movement execution. It is implemented through an hybrid planning architecture, combining a pair of global and local planners that run in parallel and recurrently.
The global planner builds an offline trajectory and shares the solution with the local planner for further processing; the local planner manages the offline trajectory, extracting way-points and, with a local constraint solver, it tries to identify obstacles. If no constraint is detected, the movement toward the current way-point is assigned to low-level controllers. Otherwise, the logic consists in asking the global planner to calculate a new trajectory.
A third method consists in the direct control of the exoskeleton. This is done with a specialized ROS node, which allows to stream end-effector cartesian paths or joint space movements. Other features include collision checking and singularity avoidance.

Using ROS interfaces, users can interact with the robotic system through a variety of human-machine interfaces, such as robot visualizations, simulations, GUIs and physical inputs.

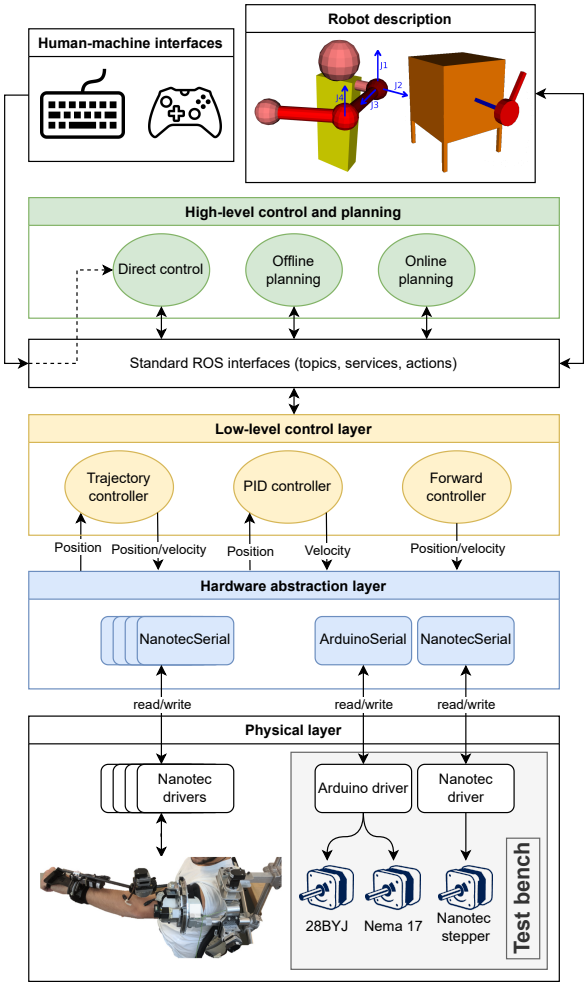

Figure 3: Overall architecture implementation

Methods

A software architecture was implemented from the selected materials to satisfy the requirements of assistive and rehabilitative robotics (Figure 3). We verified our designed architecture by means of two sets of tests.

1. *Horizontal experiments* – designed on a test-bench with three actuators (Figure 3, bottom-right), or a simulation environment (Figure 3, top-right) – consist in testing each layer of the architecture independently. The aim is to verify the ability of the architecture to customize and adapt each of its primary components (robot description, hardware, controllers, planning, human-machine interfaces...), while still conserving the remaining functionality of the system.
2. *Vertical experiments*, designed on an assistive exoskeleton (Figure 3, bottom-left) [3] to test the whole architecture at once.

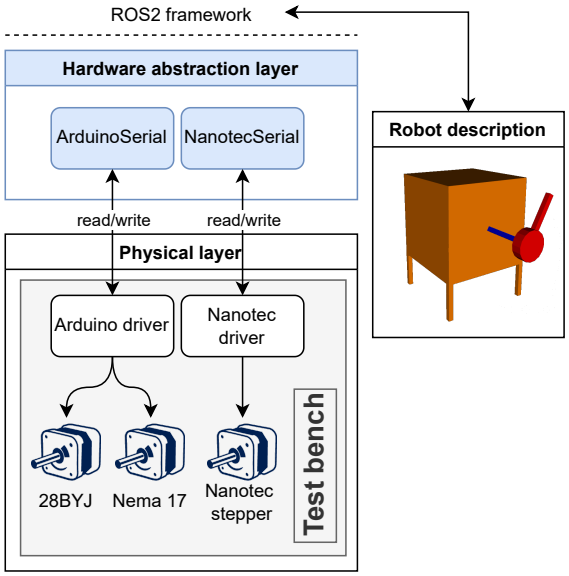

Figure 4: Hardware abstraction layer – horizontal experiments

Two abstract robotic descriptions were implemented (Figure 3, top-right): i) a simple 1-degree-of-freedom URDF model consisting of a single continuous rotational joint has been used to represent each stepper of the test-bench, used for experiments on the hardware and control layers; ii) the BRIDGE upper-limb assistive exoskeleton description, used for high-level control experiments. It has four revolute joints: shoulder horizontal abduction/adduction (Joint - J1), shoulder vertical flexion/extension (Joint - J2), humeral rotation (Joint - J3) and elbow flexion/extension (Joint - J4). Each joint has also software-based customizable ranges of motion (position limits) and velocity limits, plus reduction ratios and offsets which will be interpreted and used by the hardware abstraction.

Hardware abstraction Two different types of drivers were used as interface with the hardware: a Nanotec driver and an Arduino driver [2]. Therefore, two abstract components were designed to enable the architecture to communicate with the drivers, by means of USB serial protocol (Figure 4).
The first horizontal experiments assess whether the software architecture is able to support all two drivers and three steppers, even in parallel, demonstrating hardware modularity. Four experiments were chosen for this purpose - controlling the actuators at equal or different velocities

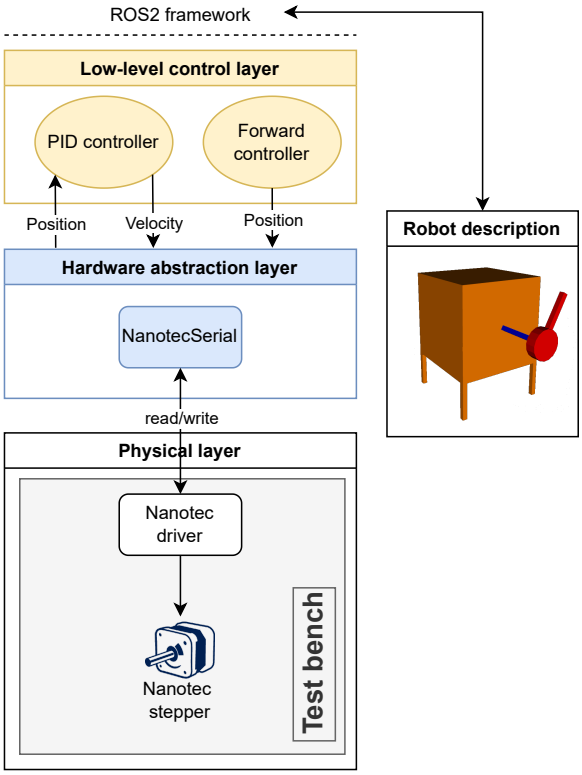

Figure 5: Control layer – horizontal experiments

and at equal or different positions. The evaluation consists in assessing the outcome qualitatively: the focus is not on performances but on the correctness of the execution.

Control layer Horizontal experiments for the control layer aim at verifying if different control modalities and schemes can be implemented. In particular two main control modalities are examined: i) an open-loop controller (Figure 5, forward controller) with position control; ii) a closed-loop controller (Figure 5, PID controller) with velocity control $\dot{x}(t)$ and position error feedback $e(t)$:

$$\dot{x}(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

Open/closed loop are tested with a single actuator (Figure 5, Nanotec stepper), keeping the hardware and hardware-abstraction fixed.

Planning layer For horizontal and vertical experiments on planning functions, MoveIt2 operating parameters have been set: i) the direct/inverse kinematic solver (KDL library, using a joint-limit-constrained pseudoinverse

Jacobian method); ii) the planning method (OMPL library, an open source implementation of more than 40 different sampling-based algorithms [13]); iii) planning adapters, which fix start state bounds and time-parameterize paths. Semantic description is then built with joint groups, virtual joints, Collision Matrix and robot poses.

All horizontal experiments for high-level control modalities (planning layer) implement a virtual robot (Figure 6, with fake hardware)

Offline planning tests are conducted receiving plan requests from user interfaces and calculating trajectories considering collisions and other constraints given by adapters. The trajectory is then sent to controllers and visualization tools which display the path in a three-dimensional virtual space. For the tests to be successful, the virtual robot must reach the correct end-configuration.

In direct control tests, a joystick is integrated in the architecture by means of a software driver that converts hardware inputs to adimensional commands in the range [-1,1]. Before converting Cartesian/joint commands from the joystick to states in the joints space, a ROS node deals with obstacles checking, singularity avoidance and joint limits by accessing the planning scene and joints current states. If everything checks out, the resulting joints configuration is sent to controllers as desired state. Finally, soft and hard thresholds ensure that the exoskeleton starts decelerating when a limit is about to be reached. Direct control horizontal test are passed if the joystick commands correspond to the correct virtual robot movements.

Online planning tests are done with a custom local constraint solver: the implemented new behaviour is comparable to a "fast switch" system. The new logic consists in switching from the execution of a planned movement to joystick commands if the local constraint solver detects a joystick input. Once joystick commands end, the global planner re-plans a trajectory from where the exoskeleton is located. The logic is implemented with three ROS nodes: two for motion planning (local

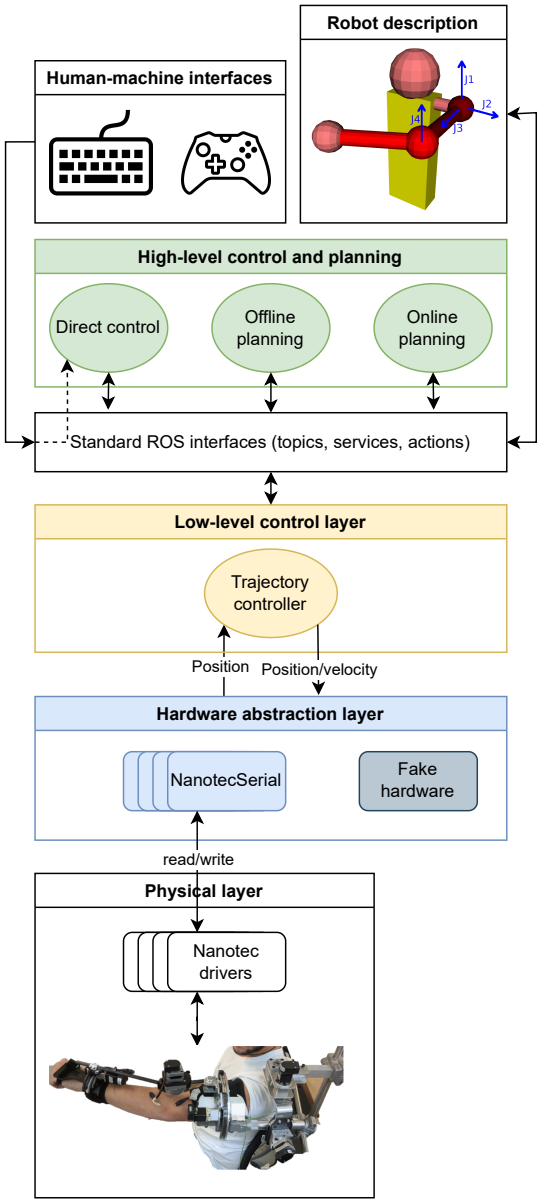

Figure 6: Architecture for vertical and high-level experiments

and global planner nodes) and one for handling joystick commands. Online horizontal tests are successful if the end-configuration is reached and the joystick successfully deviates the optimal trajectory as expected.

Vertical experiments In the vertical experiments all layers of the architecture are integrated (Figure 6, with real hardware). The BRIDGE robot description is selected; hardware abstraction layer implements four abstract components for the exoskeleton; low-level control uses a close-loop controller with position feedback, velocity/position commands, trajec-

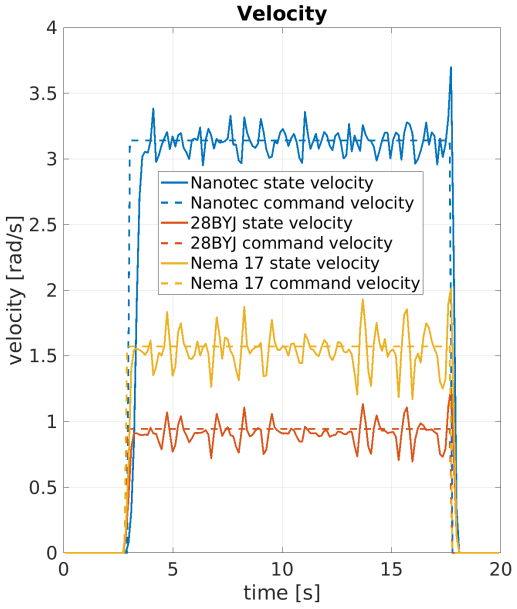

Figure 7: Control of different hardware, different commanded velocities

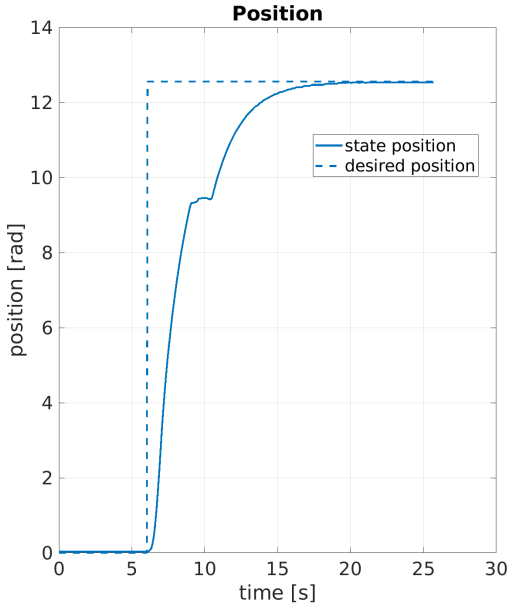

Figure 8: Close-loop control using PID transfer function, with an obstacle around 9.42 *rad*

tory interpolation capabilities and PID transfer function (if required); all high-level control modalities are used for the vertical experiments.

Results

Hardware abstraction Horizontal experiments on hardware abstraction layer consist in controlling the different actuators of the test-bench with velocity or position commands. The collected data shows that the 3 actuators were able to successfully reach the correct commanded velocities or positions, whether giving same signals to all actuators or a different signal for each of them (Figure 7).
Hardware layer results demonstrate how multiple hardware from different manufacturers can be connected and controlled in parallel by the same system.

Control layer To demonstrate the modularity of control approaches, two different control modalities, one open-loop and one close-loop, have been tested.
The first test is done with a feed-forward position controller and a temporary obstacle during the movement; since the controller cannot receive feedbacks from the physical layer, the actuator couldn't reach the proper position.
Subsequently, a feed-back PID controller is implemented, which sends a velocity command proportional to the error between current and reference positions. In this case, the actuator correctly continued the execution up to the desired position despite the obstacle (Figure 8).
These qualitative results demonstrate the possibility to modulate the control strategy according to the choice and requirements of the application.

Planning layer For horizontal planning and direct control experiments, the hardware was simulated and it behaves as an ideal sensor and actuator. In offline planning and direct control modalities, the attempts in achieving a goal state with and without an obstacle were all successful (Figure 9-A). For online planning, the correctness of switching between planning and direct control is evaluated. Tests show that, even with joystick and planner concurrency (fig. 9-B), the goal state is correctly reached.
In horizontal experiments on the direct control modality, results point out that the Cartesian path of the end effector (fig. 9-C) lies on the same vertical plane, even though in some experiments, the path deviates from the fixed coordinate. This behaviour can occur due to singularities avoidance. Indeed, in some situations the system seems to try bending the elbow to avoid

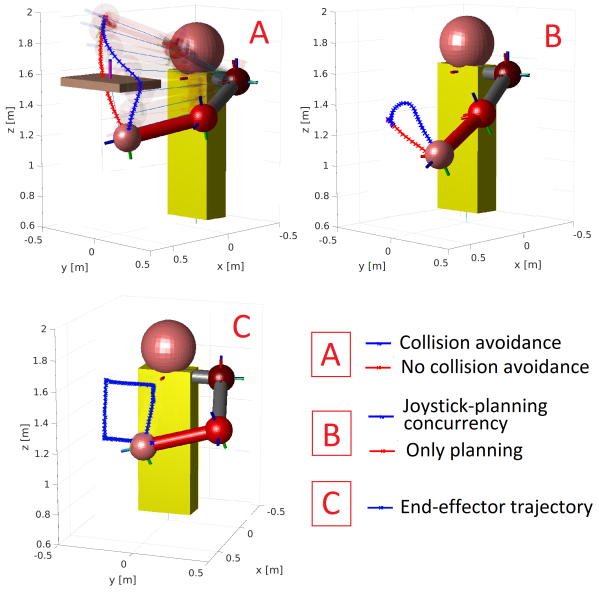

Figure 9: A: Offline planning; B: online planning with planner - joystick concurrency; C: direct control with joystick

outstretching the arm. Tests are performed both with keyboard and joystick, with similar performances.

Vertical experiments Vertical validation involves the use of the whole architecture on the BRIDGE exoskeleton, worn by a subject of 80 Kg. The subject was requested to interact with a computer running virtual environment, and with a joystick to directly control the exoskeleton. Vertical experiments include: i) offline planning with physical obstacles (table or bottle); ii) online planning with direct subject intervention through joystick; iii) drawing of a quadrilateral in the plane Y-Z with direct control modality.
All previous high-level control tests had positive results: in all experiments the goal state is reached or the movement functionality operates correctly.

Discussion

This project demonstrates the possibility to implement an architecture for exoskeletons, either rehabilitative or assistive, following recent robotic standards and an overall philosophy of system modularity. All concepts were followed while keeping the overall analysis tied to the biomedical field, in particular the topic of motor disability.
Specifically, this project focused on the customization and cooperation capabilities of the different elements that compose an exoskeleton or human-robot system, such as decision system, hardware, control system, high-level functionalities and planning, intention-detection mechanisms, human-machine interfaces and user-feedback. Qualitative results demonstrated that an overall customization is possible to obtain, still preserving a robust functionality for each of the modules of the system.

The adoption of a standard software and a modular design philosophy in rehabilitation and assistance can help different realities and fields of study to come together for a common objective; indeed, partitioning a complex problem into simpler subsystems allows people with different backgrounds and knowledge to work separately and concentrate their expertise on tasks they are proficient in.
The concept of a modular architecture is to fuse all these individual elements into a single system where they can be easily switched, composed together, customized and re-used, avoiding to redesign the whole system over and over, and to waste physical and mental resources.
However, systems that are designed to stay fixed may display a greater connection between the system components, as they can be optimally designed for that particular instance. In many cases, an integrated design can also help lowering the prices, mostly for hardware, and this is one of the main factors in the assistive scenario. Therefore, some trade-offs are always present between the different design philosophies, and the choice on how to implement each functionality should depend on the context.

In the future, this proof of concept and architecture design will be used by the Neuro-Engineering And medical Robotic Laboratory (NEARLab) for projects currently under development. The final hope is that this work could be an useful starting point for a more outlined methodology on how to construct a modular human-robot system, for both clinical and assistive scenarios.

References

[1] David Coleman, Ioan A. Şucan, Sachin Chitta, and Nikolaus Correll. Reducing the

barrier to entry of complex robotic software: a moveit! case study. *Journal of Software Engineering for Robotics*, 2014.

[2] Dario Comini and Daniele d'Arenzo. arduino_stepper_serial_driver, 2022. URL: `https://github.com/Assistive-Exoskeleton/arduino_stepper_serial_driver`.

[3] Marta Gandolla et al. An assistive upper-limb exoskeleton controlled by multimodal interfaces for severely impaired patients: development and experimental assessment. *Robotics and Autonomous Systems*, 143:103822, 2021. `doi:https://doi.org/10.1016/j.robot.2021.103822`.

[4] R.A.R.C. Gopura et al. Developments in hardware systems of active upper-limb exoskeleton robots: A review. *Robotics and Autonomous Systems*, 2016.

[5] Stefano Dalla Gasperina et al. Review on patient-cooperative control strategies for upper-limb rehabilitation exoskeletons. *Front Robot AI*, 8:745018, December 2021.

[6] Steven Macenski et al. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 2022.

[7] Joan Lobo-Prat, Peter N Kooren, Arno HA Stienen, Just L Herder, Bart FJM Koopman, and Peter H Veltink. Non-invasive control interfaces for intention detection in active movement-assistive devices. *Journal of neuroengineering and rehabilitation*, 2014.

[8] Abolfazl Mohebbi. Human-robot interaction in rehabilitation and assistance: a review. *Current Robotics Reports*, 2020.

[9] C. Nguiadem, M. Raison, and S. Achiche. Motion planning of upper-limb exoskeleton robots: A review. *Applied Sciences*, 2020.

[10] World Health Organization and World Bank. World report on disability, 2011.

[11] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson College Div, 3rd edition, 2010.

[12] Klamroth-Marganska V. Stroke rehabilitation: Therapy robots and assistive devices. *Springer*, 2018.

[13] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The open motion planning library. *IEEE Robotics and Automation Magazine*, 2012.