Software Development for Collaborative Robots

Final Project Assignment v1.0.0

Edoardo Lamon, edoardo.lamon@unitn.it

Academic Year 2025-26

1 Project Overview

This is the final project assignment for the Software Development for Collaborative Robots course in the academic year 2025-26. The goal of the project is the development of a collaborative human-robot application. The key challenges include human intent recognition, handling a dynamic environment, timing and coordination, safety, adaptability to human variability, and acceptance and trust in the robotic system. In this project, you will have the opportunity to apply your knowledge of C++ and other programming languages, robotics frameworks and libraries, and collaborative robot technologies to create a sophisticated software system.

Learning Outcomes

1. Develop a collaborative robotics application that includes interaction, motion, and task planning and execution;
2. Integrate advanced programming aspects in C++ and Python to promote code modularity and reusability;
3. Manage and maintain software in a team setting, including documentation, testing, continuous integration, and version control;
4. Present and highlight, within a group, the distinctive features of their own work.

1.1 Project Description

This topic allows students to select an area of interest and develop a collaborative human-robot application in this area. It might be related to your personal interest or the possibility of exploiting the expertise acquired within other courses. For example, it might be interesting to include standard image processing in robotised medical examinations or deep learning algorithms for human motion prediction in industrial settings. A **brief proposal** (1 page max) describing the scope, objectives, and expected outcomes of the project must be submitted by email for evaluation and approval before starting the work. Feel free to discuss it with the lecturer to ensure that all aspects of the project are considered.

1.2 Project Hardware

The targeted hardware is:

- the UR5e, with maximum payload of 3 kilograms. It is a 6-DoFs position controlled arm.With respect to the old UR5, the UR5e integrates a 6-axis force-torque sensor at the end-effector;
- a KUKA LBR iiwa with maximum payload 14 kilograms. It is a 7-DoFs torque controlled arm which features torque sensing at the joint level and also estimates the force applied onto the end-effector. Its technical features are really close to the KUKA LBR Med, which is certified for medical applications.
- a tool to be mounted at the robot end-effector. This depends a lot on the targeted task. For example, for object grasping a suitable tool can be a 2-fingers gripper Robotiq 2F-85.

Both simulation scenes, which need to be developed and modified, can reflect the lab setup. The UR3e robot is mounted on a support structure on top of a table, with the base link facing downwards. The KUKA, instead, is mounted on its pedestal. In this way, the transition from the simulation to the real world becomes straightforward.

1.3 Project Examples

The following examples illustrate the type of applications that meet the project requirements. You do not need to replicate them as they are; use them as inspiration and adapt or expand them to suit your own project idea.

Project Example 1 – Human-Robot Handover Framework This example presents a human-robot handover application in an industrial assembly simulation. In this case, the focus is not limited to one aspect of the system; instead, students may develop a full pipeline, including baseline algorithms for control, motion planning, and task planning. Here, the robot collects the pieces that the human operator is assembling. The robot should be able to grasp some objects and hand them over to the coworker, according to the plan requirements and the human proximity to the robot. An example scenario could be a simulated human approaching the table from different angles. The robot, which knows the approaching direction of the human, hands over the object in the direction of the human. If the human changes direction, the robot is able to correct his trajectory and adapt to the modified human trajectory. The objects need to be handed out in such a way that they are easily graspable by humans when they are passed. During motion adaptation, the robot must not collide with humans. Native objects, such as glass, can be used as test objects. Humans can be introduced into the simulation with predefined or random motions.

In the examples below, the contribution targets a single topic of interest, explore it in depth, and develop a state-of-the-art algorithm in C++.

Project Example 2 – Control Algorithm for Collaborative Object Transportation This example focuses on the development of a control algorithm for collaborative object transportation, where a human and a robot jointly carry a long object such as a panel or beam. The

robot must adapt its behaviour based on the forces and torques measured at the end-effector, allowing the human to guide the motion intuitively while the robot provides compliant support. The students are expected to implement a suitable control strategy in C++, for example Cartesian impedance or admittance control, ensuring that the robot follows the human's intended motion while keeping interaction forces within safe limits. An RGB-D camera is used to monitor the human posture and motion, enriching the control loop with perceptual information. For instance, the robot can adjust stiffness or damping according to the human's configuration or detect obstacles in the environment to avoid collisions during cooperative motion. When the human applies smooth, sustained forces, the robot should move compliantly; when sudden peaks or unsafe proximities are detected by the camera, the robot should increase damping or execute a controlled stop. The final controller should generate smooth references or torque commands compatible with ROS 2 control and ensure safe, predictable and intuitive behaviour during the collaborative transportation task.

Project Example 3 – Motion Planning Algorithm for Breast Biopsy This example focuses on the development of a planning algorithm for semi-autonomous breast biopsy, where the robot assists a clinician by positioning a biopsy needle toward a target lesion identified from a 3D CT scan. CT scans can be taken from publicly available open datasets, allowing students to work with realistic anatomical data. Students are required to implement in C++ a motion-planning strategy that computes a safe and feasible needle insertion trajectory starting from the robot's current configuration. The algorithm should extract the lesion pose from the CT data, compute candidate entry points on the skin surface and plan collision-free paths that respect anatomical constraints. The 3D CT volume is used to segment critical structures such as ribs, vessels or glands, enabling the planner to avoid unsafe regions and select an insertion path that minimizes tissue deformation. During the approach, the robot must ensure smooth and precise movements, integrating safety constraints such as speed limits or proximity checks. The resulting planner should generate Cartesian or joint-space trajectories compatible with ROS 2 control, providing accurate, safe and robust guidance for the biopsy procedure.

2 Project Objectives

2.1 Application Objectives

- **Simulation Environment**: Use the CoppeliaSim simulation environment for testing and validating the software without the need for a physical robot (see Appendix A for more details). Examples of scene files (with extension .ttt) and additional simulation data can be downloaded at this link. To allow control commands to be sent to the simulation, a hardware interface, compatible with ROS Control is required (follow this tutorial).

- **ROS 2 Controller Integration**: Choose a suitable robot control strategy to be used in human-robot interactive tasks. Compliant controllers (impedance or admittance) should be preferred over position controllers. Such a controller can be picked from this Cartesian controller repository, or among the standard ROS Controllers. Custom ROS controllers, if needed, can be used or developed.

- **Safety Features Implementation**: Design and incorporate safety features described by

the technical specification ISO/TS 15066 to ensure the collaborativeness of the application. This includes collision detection and avoidance, speed and separation monitoring (SSM), power and force limiting (PFL), and an emergency stop. The implementation of such strategies becomes paramount even if collision avoidance strategies are implemented.

1. While avoidance can be integrated at the motion planning level, detection can exploit the force/torque sensor mounted at the end-effector of the robot. This way, we ensure that external forces[1] are measured only when the human is grasping the object, while, in other cases, this situation represents an unforeseen, undesired contact. Ensure that, in these contacts, the robot exerts, at maximum, a limited value of force (PFL). According to the technical specification ISO/TS 15066, the maximum value depends on the body region hit by the robot. However, a conservative value of 100 N for all the body regions will be considered sufficient for the project.
2. According to speed and separation monitoring (SSM), we need to ensure that the speed of the robot's motion decreases as the human and the robot get closer.
3. Finally, a mechanism for an immediate and controlled stop of the robot's movements in case of an emergency.

Additional details on the safety features are available in the technical specification ISO/TS 15066 at this link.

- **Motion Planning Integration**: If the task requires autonomous motion planning, several open-source libraries are available for this goal. The trajectory needs to be updated dynamically if the target changes. Features like collision avoidance can also be integrated.
  - You can use simple motion planning libraries to enable efficient movement of the robotic arm, such as 3rd (for kinematically-controlled robots) or 5th-order (for torque-controlled robots) polynomials for point-to-point motions or splines for interpolating smoothly multiple points. For example, splines are available in the *ros2_control* package (see here[2]).
  - Complex motion planning libraries like MoveIt 2 or cuRobo can be also used. While they offer several off-the-shelf motion planning algorithms, their integration for first-time users is not always straightforward.
- **Task Planning Integration**: If the task requires autonomous task planning, you can prescribe a reactive plan to regulate robot behaviour according to potential faults, emergencies, sensor readings or human behaviour. This can be used to trigger different operational modes according to the safety features implemented. Behavior Trees or Finite State Machine libraries (e.g. SMACC2, yasmin, etc.) and others can be used.

Not all higher-level application components are mandatory if not required by the task/scenario (it needs to be specified in the one-page report). See also the examples.

[1] Consider that the force generated by the grasped object load is also measured.

[2] In this page they assume you are planning trajectories at the joint level but they can be used also at the Cartesian level.

2.2 Tools Objectives

- **Continuous Integration**: We expect not only issues and pull requests, but also setting up a Continuous Integration (CI) pipeline using the GitHub CI tool (GitHub Actions workflows) to automate the process of building and testing the software.
- **Documentation**: Create comprehensive documentation for the project, including README.md, user guides, system architecture, and code documentation. One way is to use Doxygen to extract the API documentation from C++ headers and use Sphinx to customise it using the Breathe python package.

2.3 Additional Objectives

Additional objectives are optional (not critical to passing the exam but can greatly improve the final evaluation) in the case of the plenary session, while mandatory for later sessions. The (+) symbol represents how much the objective adds to the evaluation (only in the plenary session):

- **User Interface Development** (+): Design and implement an intuitive graphical user interface (GUI) to be used as input/feedback for the human worker. Ensure ease of use, incorporating features such as feedback for the coworker, real-time visualisation, and task monitoring and interaction.
- **Unit Tests** (++): Implement unit tests using the Google Test (GTest) to ensure the reliability and correctness of individual code units. Make sure that you achieve high code coverage to validate the functionality of critical components.
- **Laboratory Settings Validation** (+++): Conduct tests on a physical robot in the laboratory environment to validate the software's performance in real-world scenarios. Address challenges related to hardware integration, sensor feedback, and environmental variations.

3 Software and Tools

Here is a short list of tools and libraries that can be used for the project. However, I encourage students to look themselves because there might be newer/more suitable tools online:

- **Programming Languages**: C++ (main), Python/Lua (when or if needed);
- **Robotics Middleware**: ROS 2;
- **Simulation Environment**: CoppeliaSim, EDU version (free for university);
- **Control Algorithms**: Student's choice (needs to be motivated). Examples: Cartesian controllers or standard ROS 2 controllers;
- **Motion Planning**: Student's choice (needs to be motivated). Examples: Splines(different libraries are avalable) Motion Planning Templates, MoveIt 2, cuRobo, etc.;
- **Task Planning**: Student's choice (needs to be motivated). Examples: Behavior Trees 4, Finite State Machine libraries (e.g. SMACC2, yasmin, etc.), or similar libraries;

- **Version Control and Remote Hosting**: Git and GitHub, you will update the code on your repo in the SwDev4Cobots organization.
- **Hardware-related packages**: For the UR3e, you can refer to the UR description and simulation (here the documentation). For the KUKA LBR iiwa while this repo with its documentation. For the Robotiq 2f-85 gripper, refer to this repo.

4 Evaluation Criteria

Your project will be evaluated on the following criteria:

- **Functionality** (20%): Does the software meet the specified objectives and perform the required tasks effectively?
- **Code Quality, Modularity and Robustness** (20%): Is the project written effectively? Is code reused when needed? Is the project divided into reusable components? What clever C++ features are you proud of?
- **Documentation and Usability** (20%): Is the documentation comprehensive and clear, aiding in understanding and using the application? Can a third person use it easily? Is the user interface intuitive and user-friendly? Can users easily use the application? Do you follow the standard ROS and C++ code styles, naming, and package structure?
- **Quality of the Oral Presentation** (15%): Slide design (images, videos, etc.), speaker in command of the presented topics, impactful demonstration of the project functionality.
- **Collaboration between Group Members** (15%): design awareness, usage of tools for cooperative development of software. All team members are expected to understand the project's content and the design choices made during development, and they should be able to justify and defend these decisions if asked.
- **Intuitive and Safe Human-Robot Collaboration** (10%): Have safety features been successfully implemented? Is the human co-worker aware of the robot action? How did you ensure human-robot collaboration?

5 Project Components and Deliverables

1. **One-page Report**: This report should outline the selected topic along with the main features and software components that each group plans to develop. It does not need to be final, as additional components may be introduced during the project. However, it should clearly convey the project's goals and development plan. The report must be submitted at the end of the course, regardless of the chosen presentation session. Iterative discussions with the lecturer are encouraged to refine and enhance the project's scope and relevance.
2. **Source Code Repository**: Upon communication (if you have not done it already, send it to edoardo.lamon@unitn.it) of the GitHub profile name, you will be granted access to an empty repository in the GitHub organisation SwDev4Cobots. You are supposed to use it for project development, where you will periodically push the commits to your project,

comprising at least the main branch and each student's developer branch. Don't wait until the project is finished to push the code there. Once it is finished, just add a commit to the repository with the commit message FINAL COMMIT in the main branch.

3. **Documentation**: Provide README.md and comprehensive documentation in digital format;

4. **Project Presentation**: Prepare a comprehensive presentation (maximum 20 minutes) that effectively showcases the key features and functionalities of your application. This presentation forms an integral part of the evaluation, so take the time to make it clear, engaging, and interactive. Focus on demonstrating the strengths and distinctive aspects of your software. Prefer visual elements, such as figures, videos, and diagrams, over text to enhance clarity and impact.
   You don't need to share this presentation when submitting the FINAL COMMIT.

5.1 Instructions for the project submission

Plenary session The recommended format for presenting your project is the **plenary session**. During this session, all groups will present their work to the lecturer and peers, who will also take part in a **peer review** process focused on clarity, originality, and overall impact. The most outstanding project will receive **extra points for the grade** and the **Best Project Award**. To participate in the plenary session, each group must register for the exam and finalise their GitHub repository by pushing a commit with the message *FINAL COMMIT* to the main branch no later than the day before the presentation.

Other sessions Groups that are unable to complete their project in time for the plenary session may still present and pass the exam during one of the subsequent exam sessions. In this case, the group must finalise the GitHub repository by pushing a commit with the message *FINAL COMMIT* to the main branch and notify the lecturer by email, both at least two weeks before the selected exam date. This period is necessary to allow sufficient time for code evaluation and functionality testing. The presentation will then take place on the scheduled exam date.

5.2 Important Dates

- **Team Formation Deadline**: 16 November 2025.
- **One-page Report Submission Deadline**: 18 December 2025.
- **Project Submission Deadline (Plenary sessions)**: 10 February 2026. Although 20 January 2026 is also listed in the exam calendar, the later date has been chosen to give you more time to work on the project.
- **Project Submission Deadline (other sessions)**: see other exam dates.

5.3 Important Notes

- Projects are to be completed in teams of 2 students.

- Collaboration among team members is fundamental, but individual contributions should be clearly defined and documented.
- Regular check-ins with the lecturer are encouraged to monitor progress and provide guidance.
- In case of issues faced in project development, we suggest opening threads in the dedicated Moodle section *Teacher-student discussion forum*, so that all students could benefit from the proposed solutions. Students are also encouraged to respond to the requests of colleagues if they think they know the solution!

Best of luck with your SDCR project!

Appendix A CoppeliaSim

A.1 Installation

It can be manually downloaded from the CoppeliaSim website. Otherwise, for Ubuntu 22.04 [x86_64], you can download it using the following bash command in the terminal.

```
wget https://downloads.coppeliarobotics.com/V4_8_0_rev0/
↪ CoppeliaSim_Edu_V4_8_0_rev0_Ubuntu22_04.tar.xz
```

In the case of a newer release, please update the path to the file accordingly. To extract it:

```
tar -xf CoppeliaSim_Edu_V4_8_0_rev0_Ubuntu22_04.tar.xz
```

Install the dependencies (at the moment this step needs to be performed every time the container starts):

```
sudo apt update
sudo apt install xsltproc
python3 -m pip install pyzmq cbor2 xmlschema
```

Then, to run the application, just execute inside of the extracted folder:

```
./coppeliaSim.sh
```

A.2 Enable the ZeroMQ remote API

The ZeroMQ remote API is one way to control a simulation from an external application or a remote hardware. One example is reading the robot state or the writing a target joint torque or configuration (see the documentation). To compile them:

```
export COPPELIASIM_ROOT_DIR=[path-to-CoppeliaSim-folder]
# e.g. COPPELIASIM_ROOT_DIR=~/ros2_ws/CoppeliaSim_Edu_V4_8_0_rev0_Ubuntu22_04/
cd $COPPELIASIM_ROOT_DIR/programming/zmqRemoteApi/clients/cpp
mkdir build
cd build
cmake ..
cmake --build . --config Release
```

Informations about the ZeroMQ remote API and examples are available here.

A.3 ROS2 Bridge

To enable the communication between CoppeliaSim and ROS2, in `CoppeliaSim_{version}/programming/ros2_packages/` two ROS2 packages are available: the bridge package (sim_ros2_interface) and the example package (ros2_bubble_rob). You can copy them into your workspace and then compile the workspace. Please carefully follow ROS 2 tutorial that explains it in detail.

```
export COPPELIASIM_ROOT_DIR=[path-to-CoppeliaSim-folder] # You can skip this if you set
↪  COPPELIASIM_ROOT_DIR in the previous step
ulimit -s unlimited #otherwise compilation might freeze/crash
colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release
```

Don't forget to add in `sim_ros2_interface/meta/interfaces.txt` the ROS2 messages, services, etc., you want to use to communicate with CoppeliaSim (most of them are already in the list, but others might be missing, e.g., `sensor_msgs/msg/JointState`).

A.4 Useful Links

- Examples of ROS2 publisher and subscriber scripts in Python/Lua (to be used in CoppeliaSim);
- Examples to control a robot simulation in CoppeliaSim via an external application (ZMQ Remote API and ROS2);
- Examples of ROS2 hardware interfaces implemented with *ros2_control*.

A.5 Troubleshoot

A.5.1 Blank Scene/CoppeliaSim crashing during startup in WSL

This is a known issue with CoppeliaSim, which opened in WSL. It appears that some CPUs and GPUs have issues with OpenGL software; hence, this issue does not affect only CoppeliaSim. One way to overcome this issue is to add this to the .bashrc file: `export GALLIUM_DRIVER=llvmpipe`.

Optional To only allow the rendering of the scene in the web browser, as suggested by this guide, enable first the visualisation stream add-on via [Modules > Connectivity > Visualisation stream], then open a web browser and type `http://127.0.0.1:23020`. The browser needs to be opened inside WSL (here is a guide to installing Chrome and other GUI apps on WSL).