Internship report

Autonomous Robotic System for Laptop Disassembly

Tom Voisine

tom.voisine@ensta.fr

Autonomous robotics

Mai - September 2026

Acknowledgements

First of all, I would like to thank my tutor, Atle Aalerud, for his guidance and trust throughout my internship, as well as for his availability and support during the project.

I would also like to thank Hege Indresand for her valuable help in coordinating the project and facilitating its progress. I am also grateful to the staff at the Mechatronic Innovation Lab (*MIL*) for their availability and for taking the time to answer my questions whenever needed. I would also like to thank Nabil Belbachir for the interesting talks and discussions throughout my internship.

Finally, I would like to thank all the members of the NORCE office in Grimstad for their kindness and for making me feel welcome from my arrival.

Résumé

Ce stage, réalisé au sein du CARM Center du laboratoire de recherche NORCE de Grimstad, en Norvège, porte sur le développement d'un système robotisé autonome destiné à l'ouverture d'ordinateurs (laptops) en fin de vie, dans le contexte de la récupération de matières premières critiques et de composants valorisables issus des déchets électroniques. L'objectif est de développer une solution permettant d'ouvrir les appareils avant leur broyage, tout en conservant leurs composants et en limitant les opérations manuelles.

Le système repose sur un bras robotisé UR10e, piloté avec ROS2 et MoveIt2, ainsi que sur deux outils interchangeables : une broche rotative équipée d'un foret afin de percer les têtes de vis et un préhenseur à vide pour retirer le couvercle situé à l'arrière de l'appareil. Une caméra fixe fournit les positions des vis et du point de préhension. Une architecture logicielle basée sur différents services ROS2 a été développée afin de contrôler les mouvements du robot, la broche, le préhenseur et les changements d'outil. Des fonctions spécifiques de détection de contact et de contrôle en force ont également été développées afin d'améliorer la précision et la sécurité du perçage.

L'approche proposée cherche à dépasser le dévissage séquentiel traditionnel en perçant directement les têtes de vis. Cette méthode présente plusieurs avantages, notamment une meilleure tolérance aux vis endommagées et un besoin moindre de précision lors du positionnement de l'outil. Les essais réalisés montrent que l'ouverture du laptop est mécaniquement réalisable avec cette approche. Cependant, la séquence reste principalement séquentielle. Les travaux de recherche sur le sujet doivent continuer d'être développés afin d'envisager une utilisation industrielle directe à l'avenir. Ce travail fournit néanmoins une architecture logicielle et matérielle réutilisable pour de futurs développements visant à améliorer la vitesse et l'autonomie du processus de désassemblage.

Abstract

This internship, carried out at the CARM Center of the NORCE research institute in Grimstad, Norway, focuses on the development of an autonomous robotic system for opening end-of-life computers (laptops), in the context of recovering critical raw materials and valuable components from electronic waste. The objective is to develop a solution capable of opening the devices before shredding, while preserving their components and reducing the need for manual operations.

The system is based on a UR10e robotic arm controlled using ROS 2 and MoveIt 2, together with two interchangeable tools: a spindle equipped with a drill bit to drill the screw heads and a vacuum gripper to remove the cover located on the back of the device. A fixed camera provides the positions of the screws and the gripping point. A software architecture based on several ROS 2 services was developed to control the robot movements, spindle, gripper, and tool changes. Specific contact detection and force control functions were also developed to improve the accuracy and safety of the drilling process.

The proposed approach aims to go beyond conventional sequential unscrewing by directly drilling the screw heads. This method offers several advantages, including better tolerance to damaged screws and lower positioning accuracy requirements for the tool. The tests carried out show that the laptop can be mechanically opened using this approach. However, the sequence remains mainly sequential. Further research on the subject is still needed to consider direct industrial use in the future. Nevertheless, this work provides a reusable hardware and software architecture for future developments aimed at improving the speed and autonomy of the disassembly process.

Table of contents

Acknowledgements **1**

Résumé **2**

Abstract **3**

1 Introduction **5**
1.1 Internship context and work environment 5
1.2 Problem statement and topic 5

2 Activities undertaken during the internship **7**
2.1 Strategy considerations 7
2.2 UR10e robot overview 7
2.2.1 General description 7
2.2.2 Hardware setup and ROS2 integration 8
2.3 First simulations and ROS2 control 9
2.3.1 Portable software development with docker 9
2.3.2 Trajectory planning and visualization 9
2.3.3 ROS2 cartesian control 10
2.3.4 Trajectory planning issues 11
2.3.5 Tests 12
2.4 Spindle integration 13
2.4.1 Hardware specifications and setup 13
2.4.2 ROS2 spindle control 14
2.4.3 Spindle integration into trajectory planning 14
2.5 Gripper integration 15
2.5.1 Setup 15
2.5.2 Mesh decimation 17
2.6 Automatic tool changing 17
2.6.1 Principle 17
2.6.2 Software management 18
2.7 Full opening 19
2.7.1 Height calibration 19
2.7.2 Drilling the screw heads 20
2.7.3 Pulling the cover 21
2.8 Complete sequence 22

3 Results and analysis **23**

Conclusion and perspectives **24**

References **25**

A Annexe : ROS2 commands guide **26**

1. Introduction

1.1 Internship context and work environment

NORCE is an independent research and innovation institute conducting research in severals fields, including energy, health, climate, the environment, social sciences, and technology. The institute brings together several research teams and facilities located across Norway, working on both fundamental and applied research projects. NORCE is structured into three main divisions: Health & Social Sciences, Climate & Environment, and Energy & Technology. These divisions bring together research projects and dedicated research centers with specific areas of expertise. My internship was carried out within the CARM Center (***C**ompetency Centre in **A**gder for **R**obotics Powered by AI and Data for Automated Recovery of Critical **M**aterials*), which is part of the Energy & Technology division.

1.2 Problem statement and topic

The amount of electronic waste generated worldwide has been increasing steadily year after year, making its treatment and valorization an important issue, as shown in Figure 1.1. This is also a significant concern in Norway : « Norway, with highest generation of electronic waste and low recovery of valuable non-ferrous CRMs, faces a growing challenge that manual processes can no longer address.» [1]

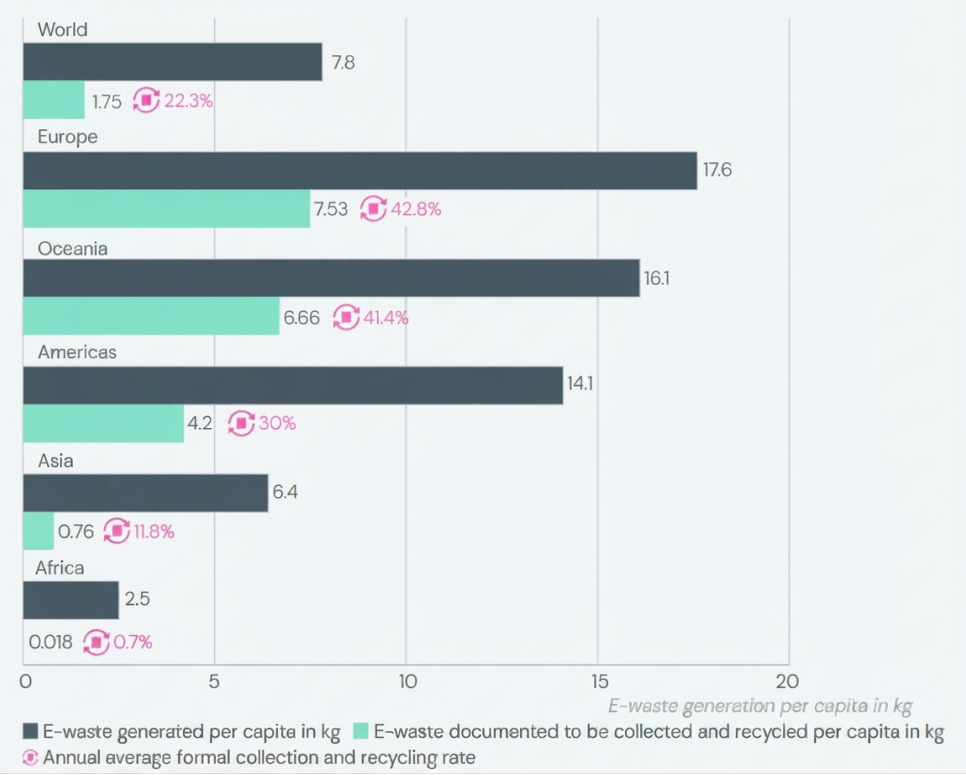


Figure 1.1: E-waste generated and Documented as formally collected and recycled by region in 2024 - [2]

The challenge is therefore to increase the capacity to process electronic waste in order to recover a greater proportion of Critical Raw Materials (CRMs) and other valuable materials. Improving the recovery of these resources is important for two main reasons. First, it can help reduce the environmental impact associated with the extraction of new raw materials, particularly by limiting the need for mining. Second, it can contribute to reducing Europe's dependence on countries that currently control a significant share of the extraction and supply of these critical resources, thereby strengthening its resource security and technological sovereignty.

Currently, a large proportion of electronic waste is still processed manually. After a rough sorting, the waste is generally shredded, with subsequent chemical treatments used to separate certain materials. However, this process can lead to significant losses of valuable components and materials.
Laptops actually contain several valuable components to be recovered, as summarized in the table 1.1.

Table 1.1: Valuable components to be recovered in a laptop - [3]

Component	CPU	RAM stick	Battery	Heat pipe
Value	Precious metals	Palladium	Cobalt & Lithium	Copper

The objective of this internship is thus to develop an autonomous robotic system able to open a laptop quickly and cleanly in a single sequence, in order to guarantee a clean access of the electronic components for further recovering. Several robotic systems already exist for automated laptop disassembly, with sequential screw removal being one of the most studied approaches [4]. However, existing solutions still face challenges in terms of processing speed, access to components, and adaptability to different laptop models.

The main challenge is to find an alternative to sequential screw removal while keeping the electronic components intact after opening. Researches are currently at an early stage in the development of projects related to these topics, providing an opportunity to build a solid foundation for future work. My role is to develop the software architecture of the robotic sequence using ROS2, as well as the simulations required for its development, mainly with RViz. I am also responsible for integrating the different hardware components, including the robot arm and its tools, into this architecture. The goal is to create a solid and reusable basis for this project and for similar projects in the future.

2. Activities undertaken during the internship

2.1 Strategy considerations

Cutting the back of the laptop with a CNC tool has been considered, but this approach presents several risks. The components located directly behind the cover could be damaged, and the cut could leave edges that would make the later extraction of components more difficult. This solution was therefore not selected.

The screws at the back of the laptop hold the cover firmly in place. A simple pulling motion would not be reliable, as the plastic or aluminium could crack instead of separating cleanly. The screws therefore need to be addressed directly. To avoid removing the screws one by one with a screwdriver, the chosen approach is to drill directly through the screw heads, using camera-based vision to detect their positions. Although this still requires processing the screws individually, it offers several advantages over sequential unscrewing. The drilling tool does not need to be carefully aligned with the screw head, making the operation faster. It can also be used when the screw head is damaged, which is particularly relevant for end-of-life laptops, and it is suitable for different types of screws.

The final opening sequence therefore consists of three main steps: drilling the screw heads, changing the tool, and pulling the back cover off.

2.2 UR10e robot overview

2.2.1 General description

The robot used for this project is the UR10e, developed by Universal Robots. It is a six-axis articulated robotic arm with six revolute joints, as shown in Figure 2.1.

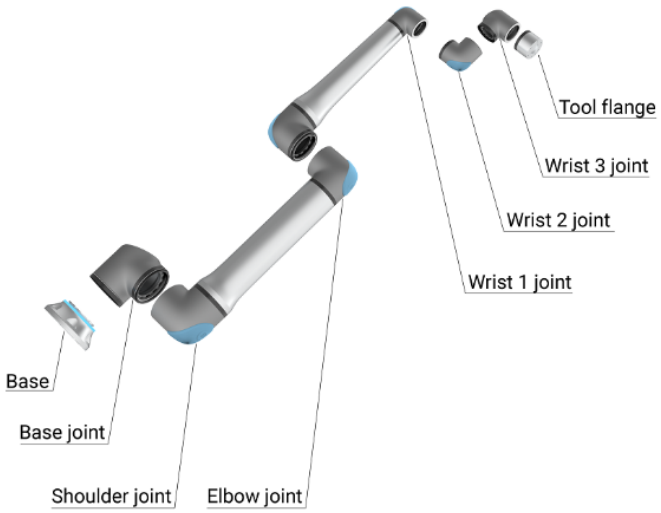


Figure 2.1: UR10e articulations overview - [5]

The Base, Shoulder, and Elbow joints are mainly used for larger movements, while Wrist 1, Wrist 2, and Wrist 3 provide finer movements and control the orientation of the tool. The tool is attached to the Tool Flange at the end of Wrist 3. Each joint can rotate through a full 360°. The Base and Shoulder joints have a maximum speed of 120°/s, while the other joints can reach up to 180°/s. The robot has a 1300 mm reach and can handle a maximum payload of 12.5 kg. This payload is sufficient for the project, as the tools used are mainly a spindle for drilling the screws and a vacuum gripper, both of which are well below this weight limit.

The UR10e is mounted on a table, with the working area surrounded by plexiglass panels to protect the operator from possible material projections during drilling. The work environment is shown in Figure 2.2.



Figure 2.2: UR10e arm used

2.2.2 Hardware setup and ROS2 integration

The PC is connected via Ethernet to the UR10e control cabinet (the UR Controller), which is responsible for controlling the robot's motors in real time and managing its safety functions. Universal Robots provides a ROS2 driver, **ur_robot_driver**, which connects the PC to the controller. It communicates with it using the RTDE (*Real-Time Data Exchange*) protocol, allowing commands to be sent and robot data such as joint positions, forces, and I/O states to be received at high frequency.

When the driver is launched, it starts a central node called **controller_manager**, which loads a set of controllers. In the **ros2_control** architecture, a controller is a software module that converts high-level commands into low-level commands sent to the robot through the hardware interface. Each active controller then provides its own ROS interfaces, such as topics, services, or actions, which allow other nodes to interact with it.

Two controllers are particularly useful. The **scaled_joint_trajectory_controller** provides an action server (FollowJointTrajectory) that allows a complete joint trajectory to be sent to the robot. The **io_and_status_controller** provides services to control the robot's digital and analog I/O, as well as topics that provide information about the robot's current state. A simplified system architecture is shown in Figure 2.3

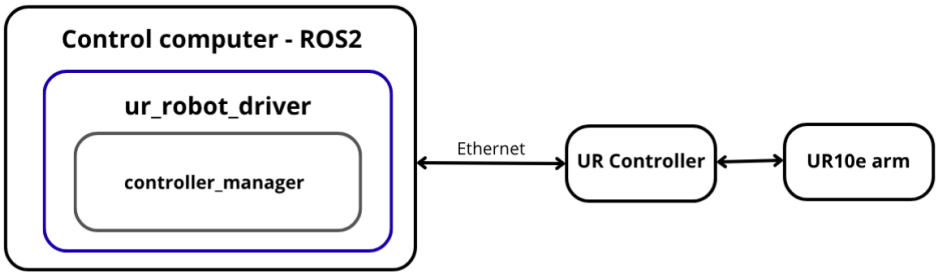


Figure 2.3: Simplified system architecture

2.3 First simulations and ROS2 control

2.3.1 Portable software development with docker

The software developed for this project was containerized using **Docker** to make it easier to use on different machines and to allow the project to be continued in the future. Docker is a tool that packages an application and its dependencies into a container, making it portable across different machines.The development environment is based on ROS2 Jazzy.

2.3.2 Trajectory planning and visualization

The first step of the project was to control the robot arm in position and move its TCP (Tool Center Point) to a desired cartesian position. This cartesian control is based on MoveIt2, a ROS2 framework for robot motion planning. MoveIt2 can plan a trajectory and send it to the controller provided by the robot driver. It also performs collision checking with the robot itself and with obstacles defined in its environment. A motion planner is then used to find a valid trajectory between the current and desired positions.

Before creating the first ROS2 position control commands, a 3D visualization environment was set up using RViz (ROS Visualization), as shown in Figure 2.4. RViz allows the robot and its environment to be visualized and obstacles to be defined. These obstacles can then be taken into account by MoveIt2 when planning trajectories, helping to avoid collisions during real robot movements.

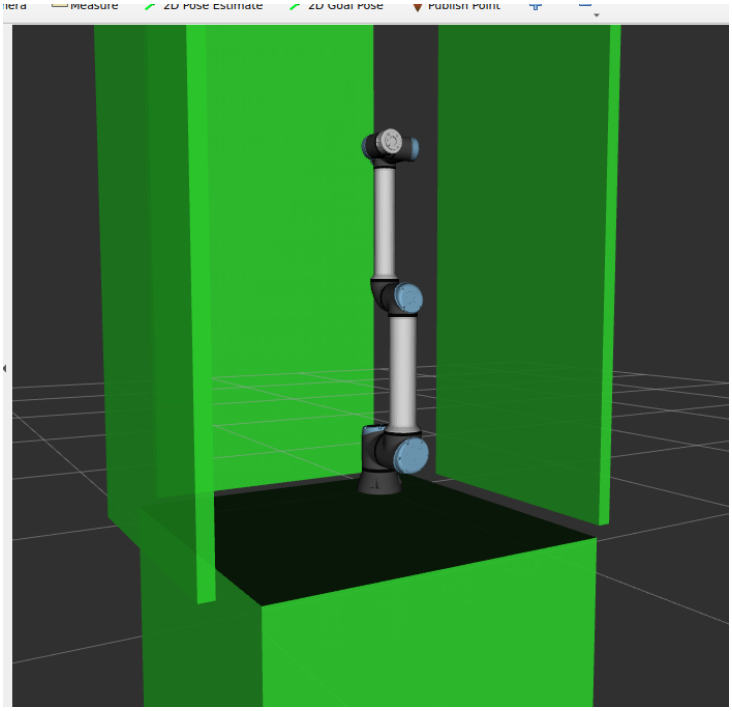


Figure 2.4: UR10e visualization with walls and table in RViz. The arm is in *home* position

2.3.3 ROS2 cartesian control

For position control, it was natural to create a ROS2 service. A service is useful here because it provides a clear result indicating whether the requested movement was successfully completed or not. This feedback will be important in the final drilling sequence, as it will allow the sequence to stop if a movement fails. The `move_to_pose` service was therefore developed in the **ur_motion** package (see the ROS2 commands guide in the appendix for an overview of the complete ROS2 architecture).

The service request contains a target pose expressed in the robot's base frame (see Figure 2.5). It includes a cartesian position (x, y, z) and an orientation represented by a quaternion (qx, qy, qz, qw). Quaternions were chosen instead of Euler angles to avoid orientation singularities and because they are directly supported by MoveIt2. The command to call this service is given by :

```
ros2 service call /move_to_pose ur_motion/srv/MoveToPose \
        "{x: -0.30, y: -0.85, z: 0.07, \
      qx: 0.69, qy: 0.71, qz: -0.01, qw: 0.01}"
```

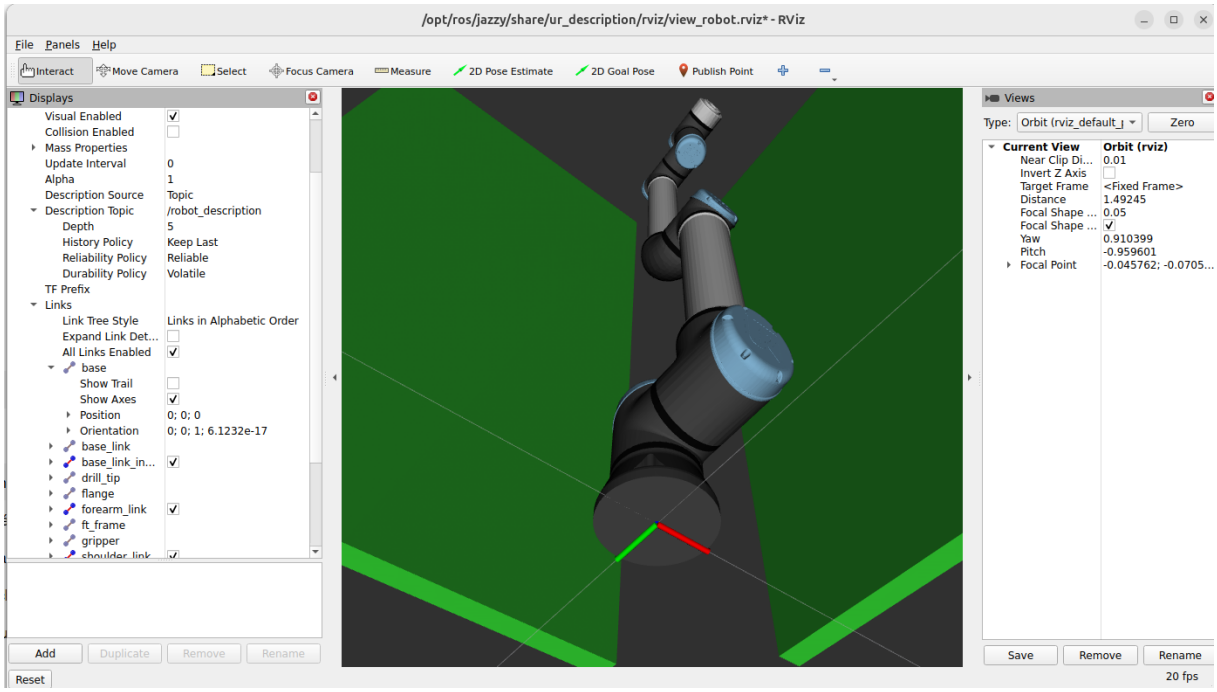


Figure 2.5: UR10e's base frame (z axis being hidden by the arm)

After conversion to the planning frame used by MoveIt2, the target pose is passed to a **MoveGroupInterface**, the C++ interface provided by MoveIt2 to control a specific kinematic group. Here, the group is **ur_manipulator**, which includes the six joints of the robot. MoveGroupInterface handles the main steps: it sends the target pose to the MoveIt2 planner, generates a collision-free joint trajectory, and executes it. MoveGroupInterface does not directly control the robot motors. Instead, it sends the planned trajectory to the active controller through the **FollowJointTrajectory** action. The **scaled_joint_trajectory_controller** receives the trajectory and executes it on the robot while respecting the speed limits defined in the controller.

2.3.4 Trajectory planning issues

During the first tests of the `move_to_pose` service, many planning errors occurred. Starting from a home pose and trying to reach a target pose close to the table, MoveIt2 sometimes failed to find a valid trajectory, even though the target pose was reachable.

The problem comes from the default planner used by MoveIt2, **OMPL**, which uses algorithms from the RRT (*Rapidly-exploring Random Trees*) family. This type of planner is not deterministic. It progressively builds a tree in the robot's joint configuration space, starting from the initial pose. At each iteration, a random configuration is selected and the tree is extended towards it. The new path is kept only if it does not cause a collision. The tree continues to grow until it reaches the target area or the allowed time and number of attempts are reached.This approach can become inefficient when the target is located in a narrow area between an obstacle and the robot's joint limits, as is the case close to the table. This is known as the *narrow passage problem*. The valid space around the target is small, so the probability of randomly reaching this area is low, especially when the starting pose is far away. The planning principle is shown in Figure 2.6. Initial and

target poses are respectively marked in green and red.

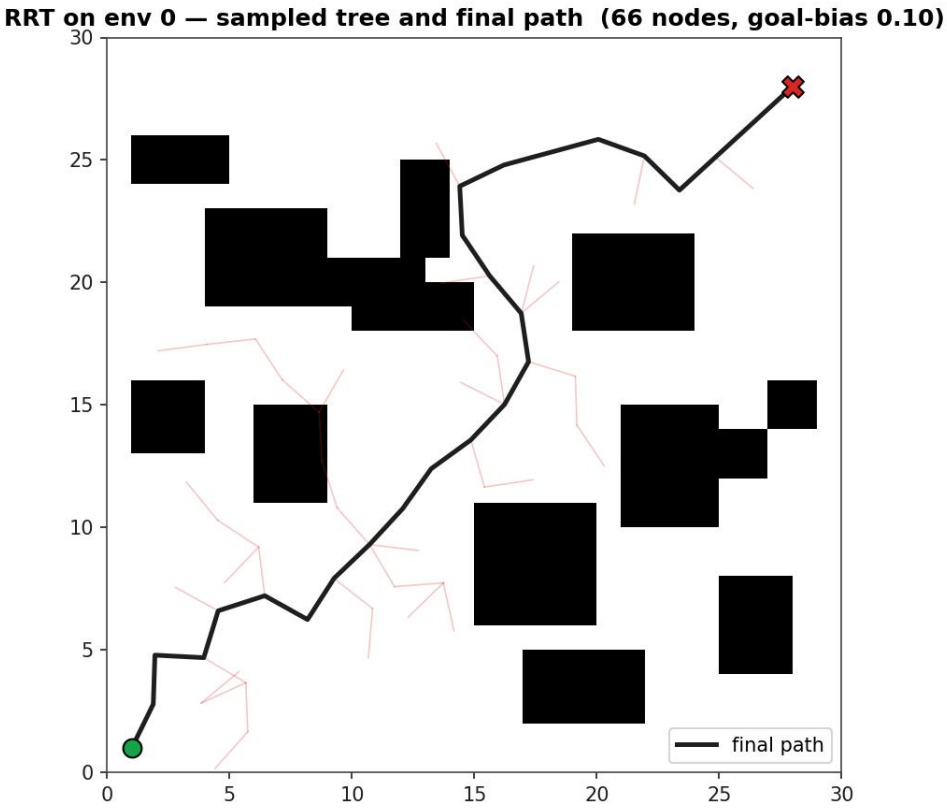


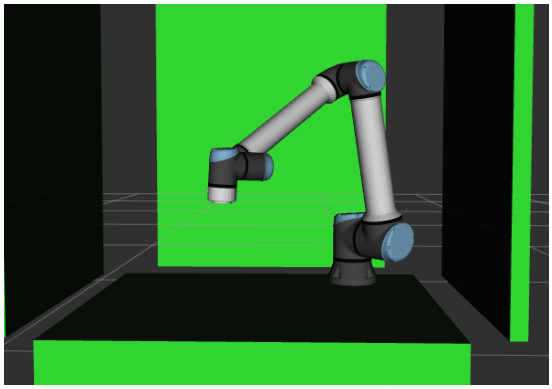
Figure 2.6: Planning tree principle - [6]

To overcome this limitation, the chosen solution was to divide the movement into intermediate poses located closer to the final target before reaching the final pose. This reduces the distance that the planner needs to explore at each step and increases the probability of finding a valid trajectory within the allowed time.

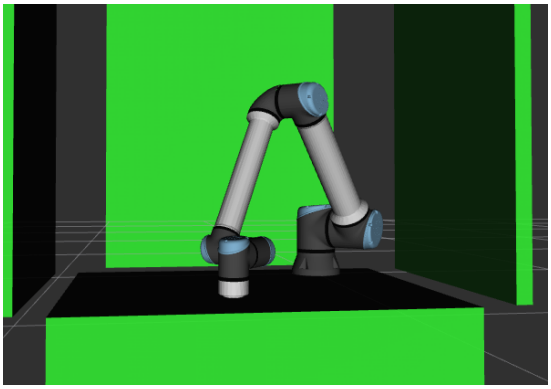
2.3.5 Tests

To test the reliability of the `move_to_pose` service, 1000 attempts were performed from the home position (see Figure 2.4) to a target position (see Figure 2.7b) where the TCP is close to the table. Two cases were tested:

- **Without an intermediate position**: direct movement from the home position to the target position.
- **With an intermediate position** (see Figure 2.7a): movement from the home position to an intermediate position, then to the target position.



(a) Intermediate pose



(b) Target pose

Figure 2.7: Intermediate and target poses in RViz

The rates of successful attempts for each case are presented in the following table :

Table 2.1: Trajectory planning success rates

Configuration	Without intermediate pose	With intermediate pose
Success rate	28%	99%

According to [5], the UR10e has a motion repeatability of ±0.05 mm. To verify this value, a test was performed with 1000 calls to the same target position. After each movement, the reached position was measured using the `get_current_pose` service (see the ROS2 commands guide), and the average positioning error was +0.051 mm. This result is satisfactory for the next steps of the project, particularly for approaching the screws to be drilled.

2.4 Spindle integration

2.4.1 Hardware specifications and setup

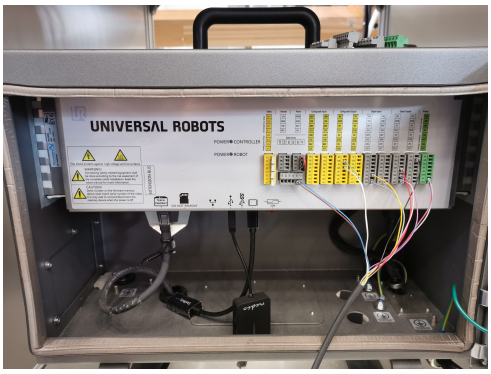
As explained previously, the laptop opening strategy is based on drilling through the screw heads rather than removing the screws using a standard screwdriver. To drill the screw heads on the back of the laptop, the robot arm will manipulate a spindle equipped with a drill bit. The spindle used is the ER20 Compact Line Module from ToolDrives (see Figure 2.8a). It has a peak power of 4 kW and a maximum speed of 18,000 rpm. The spindle is controlled using a dedicated control box (see Figure 2.8b), which connects directly to the pins of the UR Controller (see Figure 2.8c).



(a) Spindle



(b) Control box



(c) UR Controller pins setup

Figure 2.8: Spindle hardware setup

2.4.2 ROS2 spindle control

For the same reason of requiring feedback, the spindle was controlled using a ROS2 service called `set_spindle`. The spindle of the drilling system is controlled entirely through the I/O of the UR, which are exposed by the driver's **io_and_status_controller** through the `set_io` (digital) and `set_analog_output` (analog) services.
The `spindle_node` uses these two services to control the spindle from a single user request, `set_spindle`, which contains a target speed in RPM and an activation boolean.

The spindle is started and stopped using a digital output (DO3). A high state closes the circuit and activates the spindle, while a low state stops it, similarly to a simple relay. The speed is controlled using an analog output (AO1). The spindle drive interprets a DC voltage as a speed command, proportional to the desired speed. The node therefore converts the requested RPM into a ratio between 0 and 1, corresponding to the requested speed divided by the maximum spindle speed of 18,000 RPM. This ratio is then sent as a normalized voltage command to the spindle drive. An important sequencing step must be respected: the speed command is always sent before activating the spindle, followed by a short stabilization delay. This prevents the spindle from starting at no load and then suddenly increasing to the requested speed. When stopping the spindle, the order is reversed: the spindle is first switched off by setting DO3 to a low state, and the speed command is then set to zero. This prevents a speed command from remaining active while the spindle is stopped.

2.4.3 Spindle integration into trajectory planning

Once the spindle is mounted on the robot arm, it must be added to the MoveIt2 model for two different reasons. First, the planner needs to take it into account for collision detection. Second, the robot's kinematics, and therefore the planned trajectories, need to correctly account for its presence at the end of the wrist.

This is done using a file called `drill.stl`, which is an STL (STereoLithography) file. STL is a standard format used to represent a 3D mesh by describing the geometry of an object as a set of triangular faces. MoveIt2 also uses this format as a simplified geometric representation of attached tools, allowing collisions between the tool and the environment

to be checked efficiently.This geometry is referenced in the robot description, which is defined in a URDF (Unified Robot Description Format) file. In this project, the URDF is generated from a Xacro (XML Macros) source file. This file defines the robot's links and joints. A new link containing the `drill.stl` mesh must therefore be added and rigidly attached to the last link of the robot arm, `tool0`. Adding this link directly affects the TCP, which is the reference point used by MoveIt2 to define the target poses sent through `move_to_pose`. Once the spindle is mounted, this point no longer corresponds to the end of the robot wrist. It must instead be moved to the useful end of the tool, which is the tip of the drill bit. The visualization of the spindle in RViz is shown in Figure 2.9

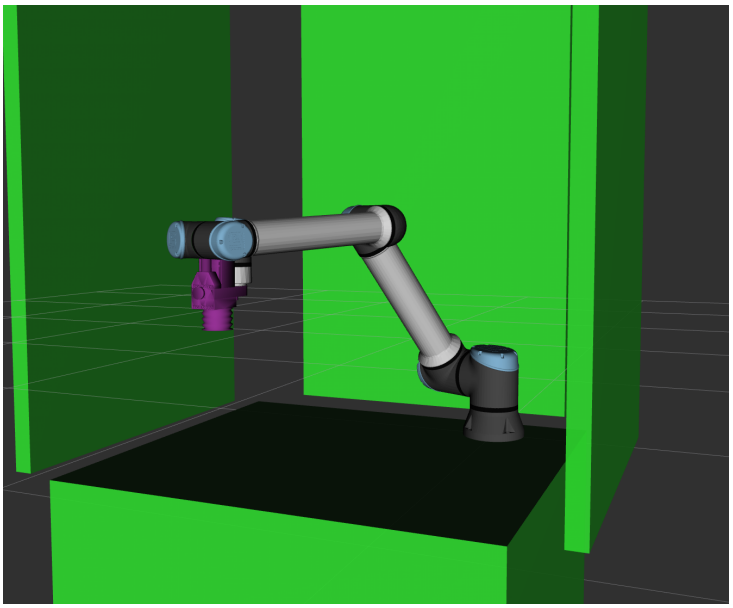
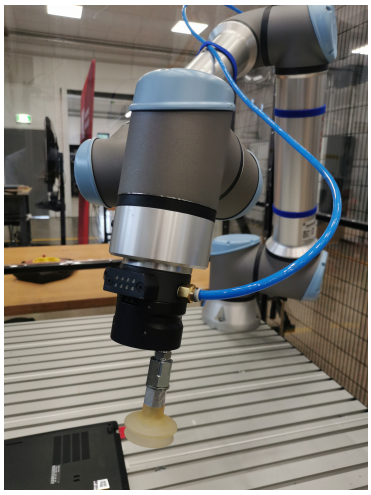


Figure 2.9: Spindle mesh in RViz

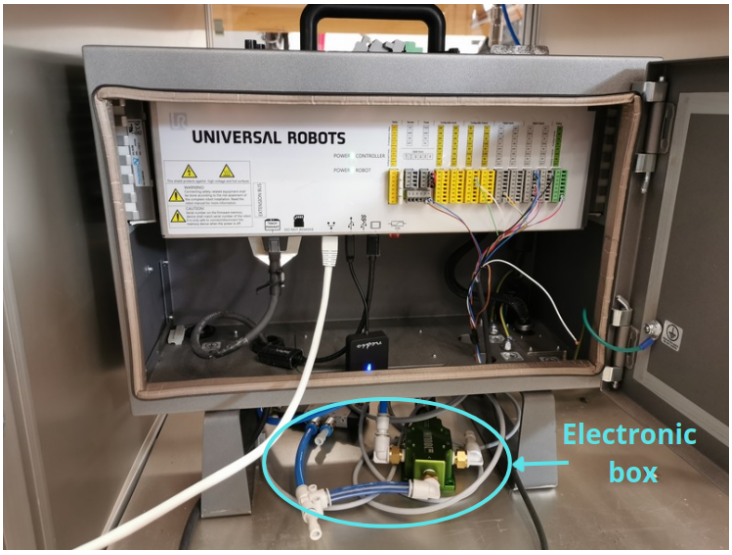
2.5 Gripper integration

2.5.1 Setup

Once the screw heads have been drilled, the robot arm will manipulate a vacuum gripper to remove the laptop cover, which is now separated from the rest of the chassis. The required hardware is simpler than for the spindle: an electronic box connected to one of the pins of the UR Controller controls an air outlet connected to the gripper, generating the vacuum required for gripping (see Figure 2.10b). The suction pressure is adjusted directly on the air outlet and is independent of the software control.



(a) Vacuum gripper



(b) Control box

Figure 2.10: Gripper setup

As with the spindle, the gripper must be added to the MoveIt2 model so that it is taken into account during collision detection. This is done using a `gripper.stl` file referenced in the robot's Xacro file.
The visualization of the gripper in RViz is shown in Figure 2.11 :

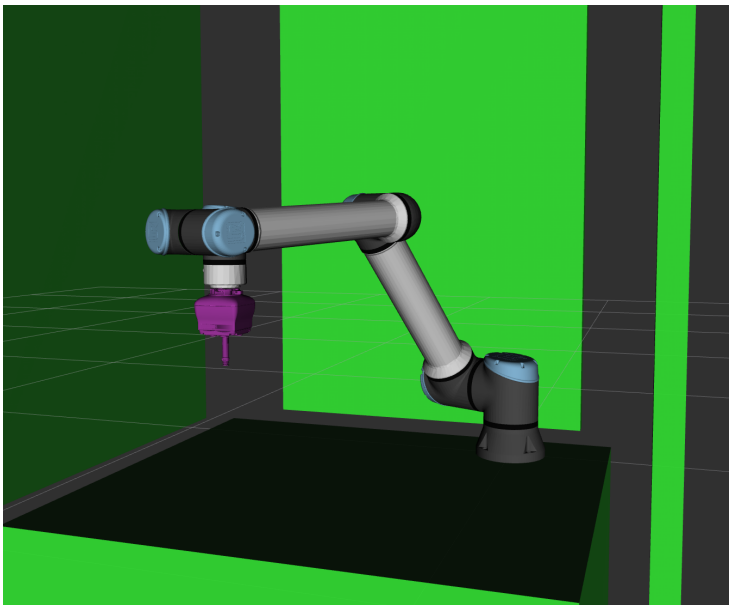


Figure 2.11: Gripper mesh in RViz

The gripper is also activated in the same way as the spindle, by setting the digital output to which it is connected. This is done using the `set_io` service.

2.5.2 Mesh decimation

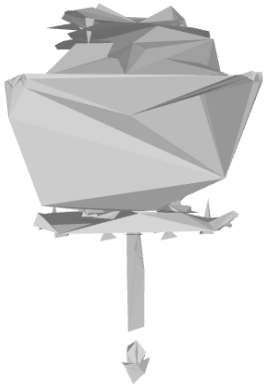
During the first trajectory tests with the gripper mounted, we encountered a performance issue. Trajectory planning with MoveIt2 became significantly slower and sometimes failed because of a timeout.

The problem came from the original `gripper.stl` file. The mesh was exported from the official maker website, with a number of triangles much higher than necessary for collision checking. The planner therefore had to check collisions against an unnecessarily complex mesh at each iteration of its search. This significantly increased the computation time and, in some cases, caused the planning time limit to be exceeded.

We therefore had to perform mesh decimation, which consists of reducing the number of triangles while keeping a sufficiently accurate geometry to represent the actual size and shape of the gripper. Collision meshes were decimated with **PyMeshLab**, cutting the drill and gripper triangle counts by 90x and 400x. For example, the gripper mesh went from 794 348 to 1988 triangles. This simplification restored satisfactory planning times without significantly affecting the reliability of collision detection. See Figure 2.12 for the gripper meshes.



(a) Gripper mesh before decimation



(b) Gripper mesh after decimation

Figure 2.12: Gripper setup

2.6 Automatic tool changing

2.6.1 Principle

As explained previously, the robot arm needs to use the spindle to drill the screws and then the gripper to remove the back cover of the laptop. These two tools cannot be mounted on the robot at the same time. We therefore use a tool changing system based on the SmartShift Tool Changers from SMARTSHIFT Robotics. The spindle and the gripper are mounted on mechanical parts that can be inserted into racks. The robot can then pick up a tool from its rack and place it back when it is no longer needed. Figure 2.13 shows the racks installation

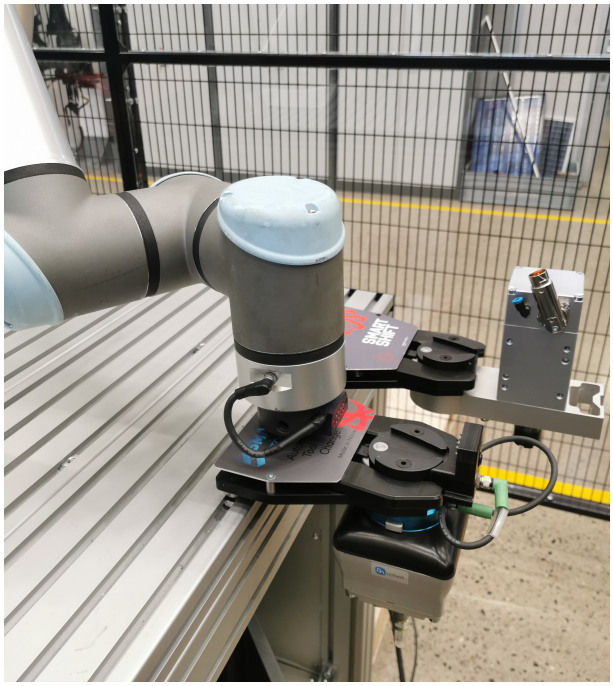


Figure 2.13: Racks installation

To perform this tool change, the robot must be able to move in a straight line. However, as explained previously, the trajectory planning with MoveIt2 uses a probabilistic planner that does not specifically consider the shape of the trajectory, as long as it can reach the target position. Therefore, there is no guarantee that the robot will follow a linear path when picking up or placing a tool. The default planner used by MoveIt2 is OMPL, but another planner called Pilz can be used to guarantee linear motion. A `move_linear` service was therefore implemented using this planner. Its request and operating principle are the same as those of the `move_to_pose` service, but it guarantees a linear movement. By defining target poses before and after the rack, the robot can therefore move in a straight line to pick up and place the tool.

2.6.2 Software management

The mounting and removal of the different tools (drill and gripper) during the same run requires a dedicated service called `set_active_tool`. This service dynamically updates the planning state to match the tool physically mounted on the robot at a given time. Without it, the node would have to be restarted after each tool change. More importantly, MoveIt2 could plan trajectories without knowing the actual geometry of the robot, creating a risk of undetected collisions.

The service takes the name of the tool to activate as a parameter (`drill`, `gripper`, or `none` when no tool is mounted) and performs three main operations each time it is called. First, the previously mounted tool is always detached before attaching a new one. This prevents two tool geometries from being present at the same time in the planning scene. Second, the collision mesh of the selected tool is attached to the last link of the robot, `tool0`, in the MoveIt2 planning scene. This ensures that the planner takes the dimensions of the mounted tool into account during collision detection, just as it does for the table or the walls of the environment. At the same time, the reference point used for trajectory

commands, the TCP, is switched to the useful end of the active tool: the tip of the drill bit or the gripping point of the gripper. This ensures that the target poses sent through `move_to_pose` correctly position the tool rather than the bare robot wrist.

Finally, the mass and center of gravity of the tool, expressed in the `tool0` frame, are sent directly to the robot controller as a short script through the URScript interface. This information is not used by MoveIt2, but by the low-level controller of the UR10e. Its dynamic model uses these values to compensate for the weight of the tool, calculate the correct motor torques, and adjust its safety limits according to the actual payload.

You can watch the tool-changing sequence at the following link : https://youtu.be/8BMfl3dX_uk

2.7 Full opening

This section describes in detail the complete strategy used to open the laptop.

The laptop to be opened is placed flat and upside down on the table. A previously calibrated camera is fixed on a stand above the table. Its video stream is processed by a machine learning algorithm that detects the positions of the screws. A so-called *hand-eye calibration* of the camera allows the system to calculate the Cartesian positions of the screw heads in the robot's *base* frame. This work was carried out by other members of the laboratory. We therefore have a vision node that provides, at the beginning of the sequence, the positions of the screw heads to be drilled, as well as the position for the gripper.

2.7.1 Height calibration

The `drill_descent` service is used to improve the accuracy of the actual position of the screw to be drilled. The drill bit length defined in the URDF is only a nominal value. Changing the drill bit can introduce a small difference, which may be enough to shift the actual contact point from the theoretical position calculated by MoveIt2. The solution is therefore to calibrate this position by performing a controlled descent until actual contact is detected, instead of relying only on the theoretical geometry.

A linear downward trajectory is first planned by MoveIt2, but it is not executed by the standard trajectory controller. The controllers are then switched:
the `scaled_joint_trajectory_controller` is disabled and replaced by two dedicated controllers running in parallel. The first controller detects contact by continuously monitoring a contact signal based on the forces measured at the tool. The second is a `passthrough` controller, which executes the descent while allowing the movement to be interrupted immediately, unlike the standard controller.

The descent and contact monitoring are then performed in parallel. As soon as contact is detected, the current trajectory is immediately cancelled, stopping the robot as close as possible to the actual surface. If no contact is detected before the end of the trajectory, the descent finishes normally. A safety timeout is also used to handle

the case where neither event occurs. The exact robot pose at the moment of stopping is returned by the service, and control is then systematically given back to the `scaled_joint_trajectory_controller`.

You can watch the height calibration sequence at the following link : https://youtu.be/0E74iO7Nqok

2.7.2 Drilling the screw heads

Unlike `move_to_pose` or `drill_descent`, the `drill_hole` service does not move the robot to a fixed Cartesian pose. Once the contact point is known from `drill_descent`, the objective is no longer to reach a precise position, but to apply a controlled pushing force along the drill axis, while allowing the robot to move further as the material is removed. This is the purpose of the UR10e force mode, which makes the robot compliant along certain axes instead of strictly controlling its position.

As with `drill_descent`, the `scaled_joint_trajectory_controller` is replaced by two controllers. The `passthrough_trajectory_controller` executes a deliberately excessive downward trajectory, going beyond the actual required drilling depth.
The `force_mode_controller` then changes how this trajectory is followed by the robot controller. It receives a task frame centered on the contact point, as well as a selection vector defining which axes remain position-controlled and which become compliant. In this case, only the Z axis is made compliant, while the other axes remain rigid to keep the drill aligned. On the compliant axis, the robot no longer follows a position target but a force command corresponding to the pushing force applied by the drill, expressed in newtons. A maximum feed speed, allowed deviations on the non-compliant axes, a damping factor, and a gain are also defined to control the responsiveness and stability of the compliant motion.

The execution is continuously monitored using two criteria. The first is the drilling depth, calculated from the current robot pose relative to the contact point. This is the normal stopping condition once the requested drilling depth has been reached. In parallel, the force measured along the Z axis by the force-torque sensor is monitored as a safety condition. If an excessive force is detected, the operation is immediately stopped, which may indicate that the drill is blocked or that a mechanical problem has occurred. In all cases, the trajectory is cancelled, the force mode is stopped, and control is returned to the `scaled_joint_trajectory_controller` to restore standard control through MoveIt2.

The `drill_hole` service requires a pushing force value as an input parameter. Several isolated tests were performed to drill screw heads using increasing pushing forces. A sufficiently high force is required: if the force is too low, the screw heads may not be completely drilled and can instead be pushed into the laptop. These tests led to a selected pushing force of **30 N**. Once the screw head is drilled, it looks like Figure 2.14



Figure 2.14: Drilled screw

You can watch the drilling sequence at the following link : https://youtu.be/tXdqRCm5UCM

2.7.3 Pulling the cover

Once the laptop cover is separated from the screws, the gripper moves into position above it. The `set_gripper` service is then activated, followed by a vertical linear movement to remove the cover. It is important to note that after drilling the screws, the cover remains attached to the laptop by several small clips. Tests with an upward pulling motion were performed and showed that the cover could be removed successfully. The laptop cover after being removed by the vacuum gripper is shown in Figure 2.15 :

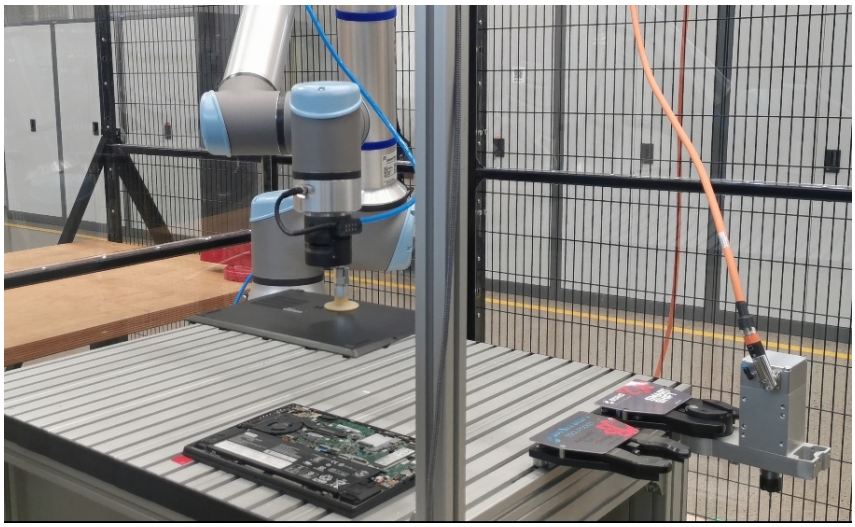


Figure 2.15: Pulled cover

2.8 Complete sequence

The complete sequence is controlled by a client node called `sequence_orchestrator`, which calls the different services described previously in a fixed and predefined order: moving to an intermediate pose, picking up the spindle (`set_active_tool`), drilling each screw (`move_to_pose`, `drill_descent`, `set_spindle`, `drill_hole`), placing the spindle back, picking up the gripper, opening the laptop, and finally placing the gripper back.

Each step is executed through a synchronous and blocking service call. The orchestrator always waits for the response before moving to the next step and immediately stops the sequence if an error occurs, such as a timeout or a `success=false` response. This avoids continuing the sequence in an uncertain state, which is important because an undetected error, such as incorrect contact, a collision, or insufficient vacuum, could cause material damage.

Between each main action, such as drilling or gripping, the robot always moves through intermediate safety poses (`intermediate_tool0`, `intermediate_drill`, and `intermediate_gripper`). These poses are located sufficiently far from the work surface and the tool racks. They act as mandatory intermediate points between two independently planned movements. This avoids requiring MoveIt2 to directly plan a trajectory between two very different poses that may be close to obstacles, as in the narrow passage problem described previously. They also ensure that the robot has a known and repeatable joint configuration before each new step.

All poses, expressed in the robot's *base* frame, as well as the main operating parameters, are stored in a configuration file called `poses.yaml`, which is loaded as ROS2 parameters when the node starts. This makes it possible to adjust the sequence without recompiling the code. For example, the rack poses and the cover placement position can be recalibrated after a physical change to the setup. Drilling parameters, such as spindle speed, pushing force, safety force, drilling depth, and the additional descent distance used for contact detection, can also be adjusted without modifying the source code. The file also contains test flags (`enable_spindle`, `enable_drill_hole`, `enable_gripper`) that allow individual actions to be disabled, such as actual drilling or vacuum activation. This makes it possible to test the complete sequence in simulation before running it on the real hardware.

You can watch the full sequence (with height calibration only) at the following link : https://youtu.be/6gFiHYgxpn4

3. Results and analysis

The complete sequence developed during this internship demonstrates the feasibility of the proposed approach for opening a laptop with a robotic arm. The different hardware and software components were successfully integrated, and the robot is able to perform the main steps of the process: picking up the tools, drilling the screw heads, changing tools, and removing the laptop cover. The drilling approach also proved to work mechanically, provided that the drilling parameters are correctly adjusted.

However, the main objective of improving the processing speed was not fully achieved. As explained in the introduction, the robotic systems currently developed for electronic device disassembly mainly rely on sequential screw removal. The approach developed in this project also remains sequential, since the robot must still process each screw individually. The main difference is that the screws are drilled rather than unscrewed.

Nevertheless, this drilling approach offers several advantages compared with the use of a screwdriver. It can be used with different types of screws and does not require the screw head to be in good condition. It also requires less precise alignment between the tool and the screw, since the drill bit does not need to fit into the screw head in the same way as a screwdriver. This can make the operation more robust when dealing with end-of-life laptops, where screws may be damaged or have different shapes.

The approach also has some limitations. The drilling parameters, especially the pushing force, must be carefully adjusted to ensure that the screw heads are properly removed without pushing the screws into the laptop. The drilling operation also produces material projections, which requires a suitable and safe working environment. In addition, the complete sequence includes several tool changes and robot movements between the different operations, which significantly increases the total processing time.

Therefore, although the drilling approach works mechanically and provides some advantages over conventional unscrewing, the overall sequence remains too slow for direct industrial use. The main limitation is not the ability of the robot to perform the opening process, but the time required to perform all the individual operations sequentially. Further improvements would therefore be necessary to reduce the number of sequential operations and increase the overall processing speed.

Conclusion and perspectives

This internship focused on the development of an autonomous robotic system for opening end-of-life laptops before shredding, with the objective of improving the recovery of valuable components and critical raw materials from electronic waste. The work resulted in the integration of a UR10e robotic arm with a spindle and a vacuum gripper, together with a ROS2 and MoveIt2 software architecture for controlling the robot, tools, and complete opening sequence. The drilling approach developed during the project was successfully tested and demonstrated that a laptop can be mechanically opened using this method.

However, the complete sequence remains mainly sequential and requires several robot movements and tool changes. As a result, the processing time remains too high for direct industrial use. The results obtained during this internship should therefore be considered as an initial step towards the development of a more efficient automated recovery system. In particular, it would be interesting to investigate more destructive approaches in order to increase the processing speed. For example, an approach based on crushing or boxing could potentially reduce the number of individual operations required to open the devices. Such approaches would nevertheless require further study to determine their impact on the preservation and recovery of valuable components.

Another important perspective would be to extend the project beyond the opening of the laptop to include the identification and recovery of its components. Research on component recognition using machine learning and artificial intelligence is developing rapidly and could complement the work carried out during this internship. Combining automated opening with component recognition and recovery could ultimately lead to a complete robotic recovery station for end-of-life electronic devices.

More generally, this project is still at an early stage in addressing the different challenges involved in the automated recovery of electronic waste. Many aspects remain to be investigated, particularly regarding processing speed, adaptability to different laptop models, component identification, and the optimization of the complete sequence. Continued research in this direction is therefore important to progressively develop more efficient and autonomous solutions.

Finally, this internship was a valuable experience from a technical and methodological point of view. I was confronted with many hardware and software difficulties and had to learn how to solve them in a structured and methodical way. This experience highlighted the importance of building a complex robotic system step by step and of thoroughly testing each new component or function before moving on to the next stage. Dividing the project into independent and testable parts made it possible to identify problems more easily and to ensure that each stage was built on reliable foundations. This approach will be particularly useful for future robotics projects involving the integration of multiple hardware and software components.

References

[1] CARM center official website : https://www.norceresearch.no/en/projects/carm

[2] UNITAR and ITU (2024). *The Global E-waste Monitor 2024*. Geneva/Bonn: United Nations Institute for Training and Research (UNITAR) and International Telecommunication Union (ITU).

[3] Priya, A., & Hait, S. (2018). *Comprehensive characterization of printed circuit boards of various end-of-life electrical and electronic equipment for beneficiation investigation.* Waste Management, 75, 103–123. https://doi.org/10.1016/j.wasman.2018.02.014

[4] Poschmann, H., Diels, J., Greschke, P., & Müller, F. (2020). Disassembly 4.0: *A Review on Using Robotics in Disassembly Tasks as a Way of Automation.* Chemie Ingenieur Technik, 92(3), 249–257. https://doi.org/10.1002/cite.201900107

[5] Universal Robots official website https://www.universal-robots.com

[6] D. Maldonado Naranjo, *Rapidly-Exploring Random Trees: Sample-Based Planning in Continuous Space, 2026*, Available: https://d8maldon.github.io/projects/path-planning/rrt/

[7] SMARTSHIFT Robotics official website : https://smartshift-robotics.com/

[8] J. Li, M. Barwood, and S. Rahimifard, Robotic disassembly for increased recovery of strategically important materials from electrical vehicles, *Robotics and Computer-Integrated Manufacturing*, vol. 50, pp. 203–212, 2018. https://doi.org/10.1016/j.rcim.2017.09.013

[9] M. E. Asif, A. Rastegarpanah, and R. Stolkin, Robotic disassembly for end-of-life products focusing on task and motion planning: A comprehensive survey, *Journal of Manufacturing Systems*, vol. 77, pp. 483–524, 2024. https://doi.org/10.1016/j.jmsy.2024.09.010

A. Annexe : ROS2 commands guide

The following guide provides a detailed description of the ROS2 services and overall software architecture developed in the project, with the aim of facilitating the continuation and further development of the project.

UR10e project - ROS2 commands guide

Author : Tom Voisine
tom.voisine@ensta.fr

Mai - September 2026

1 Introduction

This document presents the ROS2 commands for the features developed for the laptop disassembly project using the UR10e arm. All the source code is available in the following GitHub repository : `https://github.com/tomvoisine/ur_project`.
The entire environment is containerized with Docker and is based on ROS2 Jazzy.

2 Setup

To control the UR, you can connect your laptop directly to the UR controller using an Ethernet cable. Check the IP address configured (by default 192.168.1.200) on the teach pendant and make sure you are able to ping the UR using :

```
ping 192.168.1.200
```

If you have trouble connecting to the robot, try the following commands :

```
sudo nmcli connection add type ethernet \
    ifname enx78542ee6076c \
    con-name robot-connection \
    ipv4.addresses 192.168.1.200/24 \
    ipv4.method manual

sudo nmcli connection up robot-connection
```

Once you have cloned the repository and built the Docker image, name your image ur_project to use the following commands without modification. You can then launch a container using the following command (make sure you are in the folder where you cloned the repository) :

```
docker run -it \
    --network host \
    -e DISPLAY=$DISPLAY \
    -v /tmp/.X11-unix:/tmp/.X11-unix \
    -v $(pwd)/ros2_ws:/workspace/ros2_ws \
    --name ur10_dev \
    ur_project
```

Then, you must build and source the workspace :

```
cd ros2_ws
colcon build
source install/setup.bash
```

3 Controlling the arm - All features

The first thing to do is to launch the UR driver. You can either run it on the real hardware or in simulation. For further information on arguments, check the official documentation : `https://docs.universal-robots.com/Universal_Robots_ROS_Documentation/jazzy/index.html`

On the real robot :

```
ros2 launch ur_robot_driver ur_control.launch.py     ur_type:=ur10e
   robot_ip:=192.168.1.200     headless_mode:=true     launch_rviz:=true
        description_launchfile:="/workspace/ros2_ws/src/ur_motion/launch
   /ur_rsp_drill.launch.py"
```

In simulation mode only :

```
ros2 launch ur_robot_driver ur_control.launch.py     ur_type:=ur10e
   robot_ip:=192.168.1.200     use_mock_hardware:=true     launch_rviz:=
   true     description_launchfile:="/workspace/ros2_ws/src/ur_motion/
   launch/ur_rsp_drill.launch.py"
```

Then, in a second terminal, you have to launch MoveIt2 :

```
ros2 launch ur_moveit_config ur_moveit.launch.py     ur_type:=ur10e
   launch_rviz:=false
```

(**launch_rviz** parameter is set to false because RViz has already been launched by the first command)

The repository includes several packages and nodes. They are listed below.

3.1 ur_motion package

This package contains one major node : **motion_executor** (*motion_executor.cpp*)

To run it, use the following standard ROS2 command :

```
ros2 run ur_motion motion_executor
```

This node exposes several services. They are listed below.

3.1.1 MoveToPose

Move the end-effector to a target position expressed in the robot's cartesian coordinate frame. Command to call the service :

```
ros2 service call /move_to_pose ur_motion/srv/MoveToPose \
        "{x: -0.30, y: -0.85, z: 0.07, \
      qx: 0.69, qy: 0.71, qz: -0.01, qw: 0.01}"
```

Note : the orientation must be expressed as a quaternion. This service uses the OMPL planner.

Warning : To avoid potential planning issues, avoid making large movements that require the robot to move close to obstacles defined in the MoveIt planning scene in one single movement. It is recommended to use intermediate poses that are easier for the planner to reach before moving to the final pose. This can help simplify the planner's calculations and increase the chances of successful planning. For example, if the robot is in the home position and you want to move it to a pose where the end-effector is less than 10 cm from an obstacle, it is preferable to first move the robot to an intermediate pose farther away from the obstacle, and then command the final pose.

3.1.2 MoveLinear

Similar to the MoveToPose service but forces the motion to be linear.
Command to call the service :

```
ros2 service call /move_linear ur_motion/srv/MoveLinear \
        "{x: -0.30, y: -1.15, z: 0.07, \
      qx: 0.69, qy: 0.71, qz: -0.01, qw: 0.01, speed_scaling: 0.06}"
```

Notes : This service uses the Pilz planner in order to force linear motions. The **speed_scaling** parameter can take values from 0.0 to 1.0, but it is preferable no to exceed 0.15.

3.1.3 DrillDescent

Perform a linear downward motion until contact is detected. The target position is expressed in the robot's coordinate frame and should intentionally be set beyond the estimated contact point. The robot stops as soon as contact is detected. If no contact is detected, the robot stops at the target position. The service response includes a contact_detected boolean indicating whether contact was detected, as well as the robot's position at the time of stopping, expressed in MoveIt's world frame.
Command to call the service :

```
ros2 service call /drill_descent ur_motion/srv/DrillDescent "{x: -0.38,
   y: -0.55, z: 0.01, \
      qx: -0.96, qy: -0.25, qz: 0.00, qw: -0.01, speed_scaling: 0.06}"
```

3.1.4 DrillHole

This service takes as input the position returned by the DrillDescent service upon contact detection, along with a target depth in mm, a downward pushing force in N, and a maximum safety force. The service is aborted if the safety force is exceeded. The response includes booleans indicating the success or failure of the operation, whether the target depth was reached, and whether the operation was aborted due to excessive force.
Command to call the service :

```
ros2 service call /drill_hole ur_motion/srv/DrillHole "{contact_x: 0.50,
    contact_y: 0.70, contact_z: 0.12, qx: 0.71, qy: -0.69, qz: -0.00, qw
   : -0.00, depth_mm: 4.0, push_force_n: 5.0, safety_force_n: 15.0}"
```

3.1.5 SetActiveTool

This service updates the active tool mounted on the robot's end-effector. The TCP (Tool Center Point) is adjusted through a custom XACRO file to match the actual tip of the tool, ensuring that position targets are correctly reached with respect to the tool tip. The service also loads the tool's 3D mesh into MoveIt, allowing the planner to account for the tool geometry and avoid collisions when computing trajectories.
Command to call the service :

```
ros2 service call /set_active_tool ur_motion/srv/SetActiveTool "{
   tool_name: 'drill'}"
```

The **tool_name** parameter can take 3 values : **'drill'**, **'gripper'** and **'none'** (no tool mounted)

3.1.6 GetCurrentPose

This service simply returns the current TCP position in both coordinate frames. Command to call the service :

```
ros2 service call /get_current_pose ur_motion/srv/GetCurrentPose {}
```

3.2 ur_spindle package

This package contains one major node : **spindle_node** (*spindle_node.cpp*)

To run it, use the following standard ROS2 command :

```
ros2 run ur_spindle spindle_node
```

This node exposes one service : SetSpindle

This service controls the spindle by starting or stopping it and setting its speed between 0 and 18,000 RPM. Command to call the service :

Start the spindle at a given RPM :

```
ros2 service call /set_spindle ur_spindle/srv/SetSpindle \
    "{enable: true, speed_rpm: 1000.0}"
```

Stop the spindle :

```
ros2 service call /set_spindle ur_spindle/srv/SetSpindle \
    "{enable: false, speed_rpm: 0.0}"
```

Spindle reference : ToolDrives ER20. Check documentation for the connecting procedures to the robot.

3.3 ur_gripper package

This package contains one major node : **gripper_node** (*gripper_node.cpp*)

To run it, use the following standard ROS2 command :

```
ros2 run ur_gripper gripper_node
```

This node exposes one service : SetGripper

This service allows you to activate or desactivate the gripper. Command to call the service :

Activate the gripper :

```
ros2 service call /set_gripper ur_gripper/srv/SetGripper \
    "{enable: true}"
```

Desactivate the gripper :

```
ros2 service call /set_gripper ur_gripper/srv/SetGripper \
    "{enable: false}"
```

Note : The pressure level is adjusted directly at the air inlet to which the air hose is connected.

3.4 ur_sequence_orchestrator package

This package serves as the high-level orchestrator responsible for executing the complete sequence. It calls the services described above as a client. Several configurable parameters are available for testing and adjusting the sequence :

The robot executes the sequence without activating the spindle or the gripper. Only the Drill-Descent service is called ; DrillHole is not :

```
ros2 launch ur_sequence_orchestrator sequence_orchestrator.launch.py
```

The robot executes the sequence, with the spindle activated above each screw. However, the DrillHole service is not called, so no drilling is performed :

```
ros2 launch ur_sequence_orchestrator sequence_orchestrator.launch.py
    enable_spindle:=true
```

The robot executes the sequence with the spindle activated, and the DrillHole service is called. Drilling is performed :

```
ros2 launch ur_sequence_orchestrator sequence_orchestrator.launch.py \
  enable_spindle:=true enable_drill_hole:=true
```

Full sequence with the gripper :

```
ros2 launch ur_motion sequence_orchestrator.launch.py \
  enable_spindle:=true enable_drill_hole:=true enable_gripper:=true
```

Note : The package ur_sequence_orchestrator includes a .yaml file *poses.yaml* with all the intermediate poses, as well as some parameters for the different tasks performed during the sequence. (depths, forces, ...). Fell free to adjust their values.

4 Test sequence commands

The following section summarizes the commands to be executed in separate terminals in order to launch a test sequence.

Launch the driver :

```
ros2 launch ur_robot_driver ur_control.launch.py     ur_type:=ur10e
   robot_ip:=192.168.1.200     headless_mode:=true     launch_rviz:=true
        description_launchfile:="/workspace/ros2_ws/src/ur_motion/launch
   /ur_rsp_drill.launch.py"
```

Run MoveIt2 :

```
ros2 launch ur_moveit_config ur_moveit.launch.py     ur_type:=ur10e
   launch_rviz:=false
```

Run the motion_executor node :

```
ros2 run ur_motion motion_executor
```

Run the spindle node :

```
ros2 run ur_spindle spindle_node
```

Run the gripper node :

```
ros2 run ur_gripper gripper_node
```

Finally launch the full sequence :

```
ros2 launch ur_motion sequence_orchestrator.launch.py \
  enable_spindle:=true enable_drill_hole:=true enable_gripper:=true
```

ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* NORCE Research AS

Adresse / *Address* Jon Lilletuns vei 9H, 4879 Grimstad, Norway

Tél / *Phone (including country and area code)* +47 56 10 79 02

Nom du superviseur / *Name of internship supervisor* Atle Aalerud

Fonction / *Function* Senior Researcher

Adresse e-mail / *E-mail address* ataa@norceresearch.no

Nom du stagiaire accueilli / *Name of intern* Tom VOISINE

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
*Please attribute a mark from **A (excellent)** to **F (very weak).***

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ? A (B) C D E F
  Was the initial contract carried out to your satisfaction?

- Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☒ non/*no*
  Was the intern lacking skills?

  Si oui, lesquelles ? / *If so, which skills?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)*

  (A) B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here*_____

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?

Did the intern live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

(A) B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

_____

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? A B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? A (B) C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Tom works well independently. However, it would help if he is more active in steering the project and asking questions.

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? A (B) C D E F
Was the intern open to listening and expressing himself /herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

_____

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : A (B) C D E F
Please evaluate the technical skills of the intern:

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

Would you be willing to host another intern next year? ☒ oui/*yes* ☐ non/*no*

Fait à _____, le _____
In Grimstad, on 22/9-26

Signature Entreprise _____ Signature stagiaire
~~Company stamp~~ _____ *Intern's signature*

Merci pour votre coopération
We thank you very much for your cooperation