Report SYI700 Franka

1 Task description

A Franka Emika Panda robot is used to assemble containers by selecting and placing the required colored parts. The robot receives information from the vision system specifying which parts must be picked from the buffers for assembly or disassembly. The container is picked from the conveyor, assembled according to the provided requirements, and then placed back onto the conveyor once the process is completed for inspection.

1.1 Workstation setup

As shown in Figure 1, a collaborative Franka Emika Panda robot is used to perform the pick-and-place operations. It is mounted on a workspace table where the assembly and disassembly tasks are executed. The container is placed in the assembly position, where it is secured with two handles. The workspace sits directly beside the conveyor, allowing the robot to pick up containers and return them after processing. Several 3D-printed buffers are mounted on the workspace to hold the parts required for assembly. Parts that must be discarded—either because the buffers are full or because they were not reliably detected—are placed in a designated grey disposal box on the workspace. The robot is controlled via the FCI interface, which is connected to a Linux PC running the control software in real time. ROS is used as the overarching software framework, while MoveIt is employed to compute the robot's trajectories and coordinate its motion. For integration within the overall system, the Linux PC communicates with the Beckhoff PLC through ADS. The control setup is illustrated Figure 2.

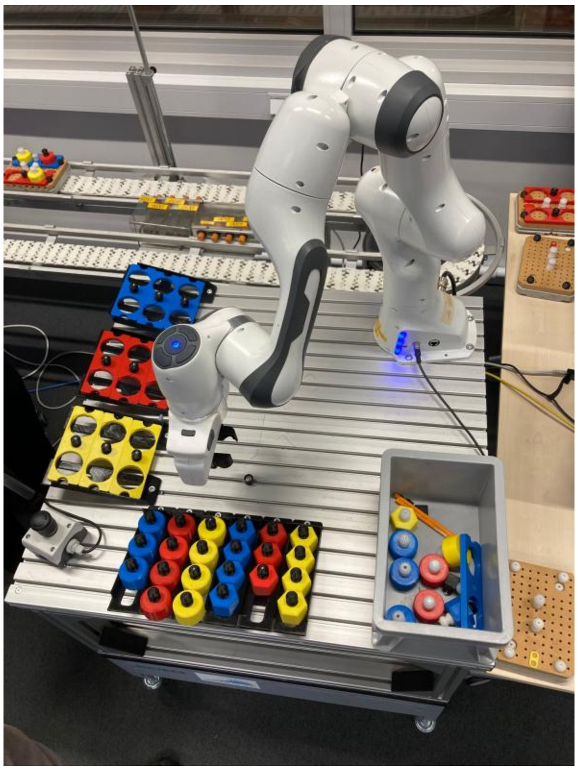

Figure 1: Workspace setup

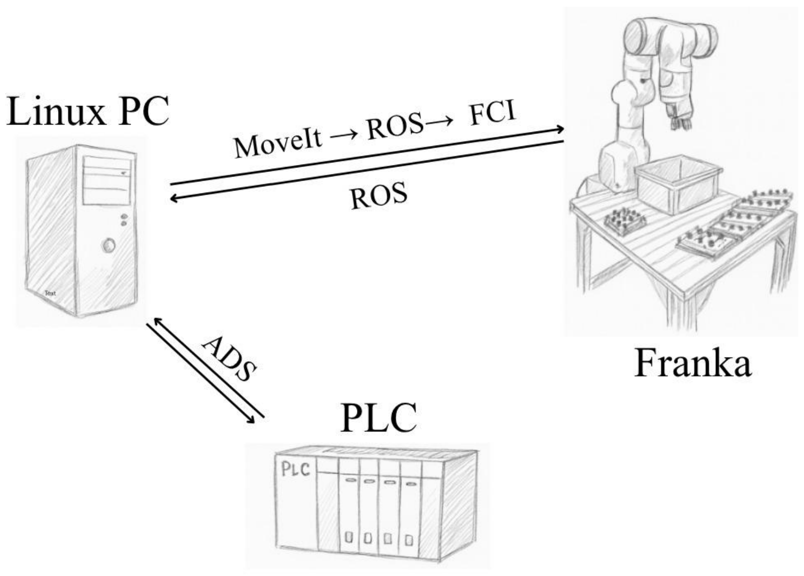

Figure 2: Franka control setup

1.2 Task sequence

The flowchart in Figure 3 illustrates the sequence of operations. Once the Franka station is ready, it waits for a start signal from the PLC. When the container is at the pick-up location, the PLC sets the start signal. The main program then reads the current assembled configuration, which has been detected by the vision system and the color which should be assembled. The Linux PC then determines the required steps to achieve the target container configuration. The program then initiates and issues the corresponding robot routines. Parts are assembled or disassembled from or to the buffers, and after processing, the container is placed back onto the conveyor and its state is set to "finished" for the PLC. If a buffer becomes empty, the assembly task is halted, and user input through the PLC SCADA system is required before the sequence can resume. In case of a crash, the robot goes into the home position and executes the next task. In case of an emergency, the enabling button in the workspace can be pressed, which then requires a restart of the system.

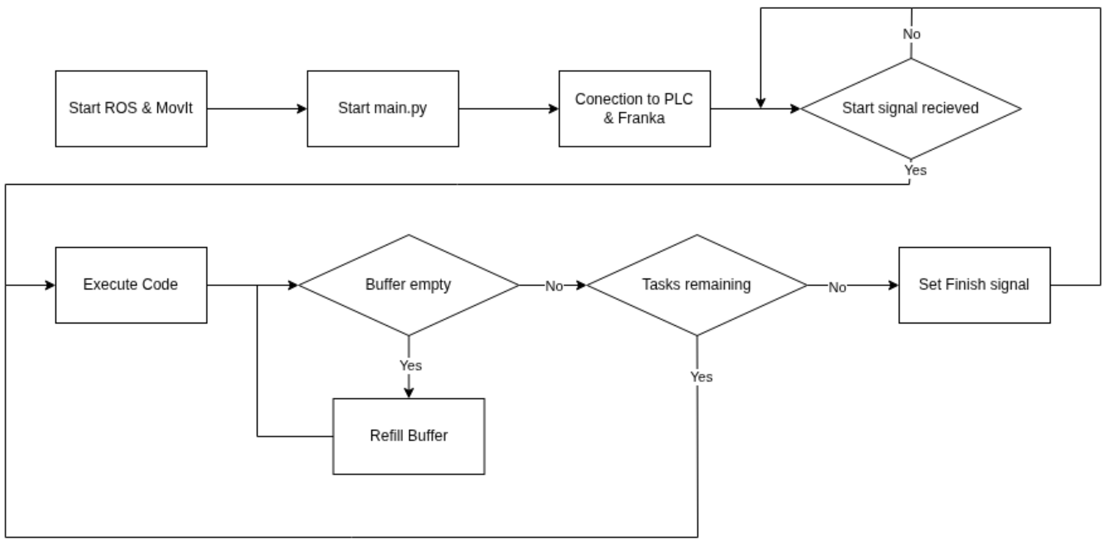

Figure 3: Task sequence

2 Linux-based control system

A Linux-based control computer is used to operate the Franka robot. The following chapter outlines the overall system structure, describes the components required for its operation, and presents the resulting program workflow.

2.1 System structure

The system structure consists of a Linux-based PC running Ubuntu, equipped with a real-time kernel to ensure deterministic communication with the Franka robot controller over Ethernet. MoveIt is used to visualize the robot in a virtual environment and to compute collision-free motion trajectories. ROS serves as the communication backbone for the entire system, operating through a publisher–subscriber model that links all software modules. The franka_state_controller continuously publishes joint states and sensor data, which MoveIt subscribes to in order to maintain an accurate representation of the robot during planning. Once MoveIt generates a trajectory, it publishes the result to the position_joint_trajectory_controller via its FollowJointTrajectory action interface. This controller converts the trajectory into real-time joint commands and forwards them through libfranka to the Franka Control Interface (FCI), which acts as the low-level communication channel to the robot. The task-level logic runs on the same Linux PC in a Python script that sends target poses to the planning pipeline and handles execution. This script also incorporates the ADS interface to the Beckhoff PLC, enabling communication and coordination with the broader automation system.

2.2 Realtime Kernel

A real-time kernel is required to ensure that communication with the robot controller occurs with consistent and predictable timing. Since the controller relies on time-critical data exchange, even small delays or scheduling variations can lead to unstable or unsafe robot behavior. A standard operating system cannot guarantee these timing constraints, which is why a Linux system with a real-time kernel was selected. This configuration allows the control processes to meet strict latency requirements and ensures reliable interaction with the robot controller.

2.3 Robot operating system (ROS)

ROS is used as the communication and coordination framework for the robot system. It manages data exchange between modules, initializes the robot control node, and provides the interfaces needed for motion planning and execution. In this project, ROS enables the integration of MoveIt for trajectory generation, handles the movement commands for the robot arm and gripper, and manages error recovery actions. Through these components, ROS supports the execution of the assembly workflow and ensures reliable interaction between the control software and the robot.

2.4 Libfranka

In this project, libfranka provides the direct communication link between the ROS–MoveIt planning pipeline and the Franka robot. Once a trajectory is generated, ROS sends the joint commands through libfranka, which forwards them in real time to the Franka Control Interface (FCI). This allows the planned motions to be executed accurately on the robot and ensures reliable low-level control during the assembly process.

2.5 MoveIt

MoveIt is used in this project as the motion-planning framework responsible for generating collision-free trajectories for the Franka robot. Based on the target poses provided by the Python control script, MoveIt computes feasible joint motions and maintains a synchronized virtual model of the robot using state information received from ROS. After planning, MoveIt sends the resulting trajectory to the ROS controller, which passes it through libfranka to the FCI for execution on the real robot. In this way, MoveIt provides the high-level planning capabilities that guide all pick-and-place and assembly actions.

2.6 Program structure and execution workflow

The main Python script is organized around a central loop that coordinates communication between the PLC, the vision system, and the robot control stack. At startup, it establishes the ADS connection to the Beckhoff PLC, initializes the ROS node and MoveIt commanders for the arm and gripper, loads all saved robot poses and joint configurations, and prepares the internal buffer state. During operation, the script continuously checks the PLC for a start signal and retrieves both the target assembly color and the vision data describing the current container configuration. Based on this information, it generates a task list specifying whether each part must be assembled, disassembled, rotated, or discarded. The robot is then guided through a sequence of high-level routines: picking the container from the conveyor, performing each task in the list, and placing the finished container back on the conveyor. All robot motions are executed by sending target poses to the planning pipeline, where MoveIt computes the trajectories and ROS/libfranka handle the execution on the physical robot. Throughout the process, the script updates the buffer state, sets the appropriate status flags on the PLC, and uses integrated error handling—such as motion recovery or buffer refill requests—to ensure the assembly sequence can proceed safely and reliably.

2.7 Communication

Communication within the system is realized through ROS for robot control and an ADS interface for coordination with the Beckhoff PLC. The PLC–robot interaction follows a defined handshake mechanism to ensure synchronized operation. The Python control script continuously monitors the PLC for a start signal. Only after this start signal is received does the robot begin the assembly sequence. During execution, the robot status is reported back to the PLC, indicating whether the system is idle, running, or in an error state. Once all tasks for a container are completed and the robot has placed it back on the conveyor, the script sends a finished signal to the PLC, confirming successful completion of the cycle.

In the event that a buffer becomes empty, the control script detects this condition internally and sends a refill request to the PLC. The robot then pauses operation and waits until the PLC confirms that the buffer has been refilled. Only after this confirmation signal is received does the program reinitialize the buffer state and resume the assembly process.

In addition to this basic signaling, the PLC–robot communication is organized as a cyclic handshake that ensures a reliable and deterministic exchange of commands and status information. The Python control program continuously reads all relevant PLC variables and evaluates them within the main control loop. Before any motion is executed, the robot explicitly reports a ready state to the PLC, indicating that initialization is complete and the system is awaiting a start command. After the PLC sets the start signal, the robot acknowledges this request by clearing the finished flag and switching its status to running. During task execution, the robot continuously updates its status, allowing the PLC to monitor the current execution state and detect faults. Error conditions, such as motion interruptions or safety stops, are directly reflected in dedicated status codes and trigger a controlled recovery sequence before the robot returns to a safe state. Buffer-related events are handled in the same handshake-based manner: when a buffer is detected as empty, the robot sends a refill request to the PLC and pauses execution until a confirmation signal is received. This structured exchange ensures synchronized operation, prevents undefined system states, and enables safe coordination between the robot and the PLC throughout the entire assembly process.

3 Mechanical components

In addition to the software components for programming and control, mechanical elements were also required for the project to function properly.

3.1 Robot gripper

The standard Franka Emika gripper was not suitable for the pick-and-place tasks, as it could not reliably grasp the part and container handles. For this reason, a custom gripper was designed using CAD software. The design was subsequently 3D-printed and mounted on the robot, as shown in Figure 4.

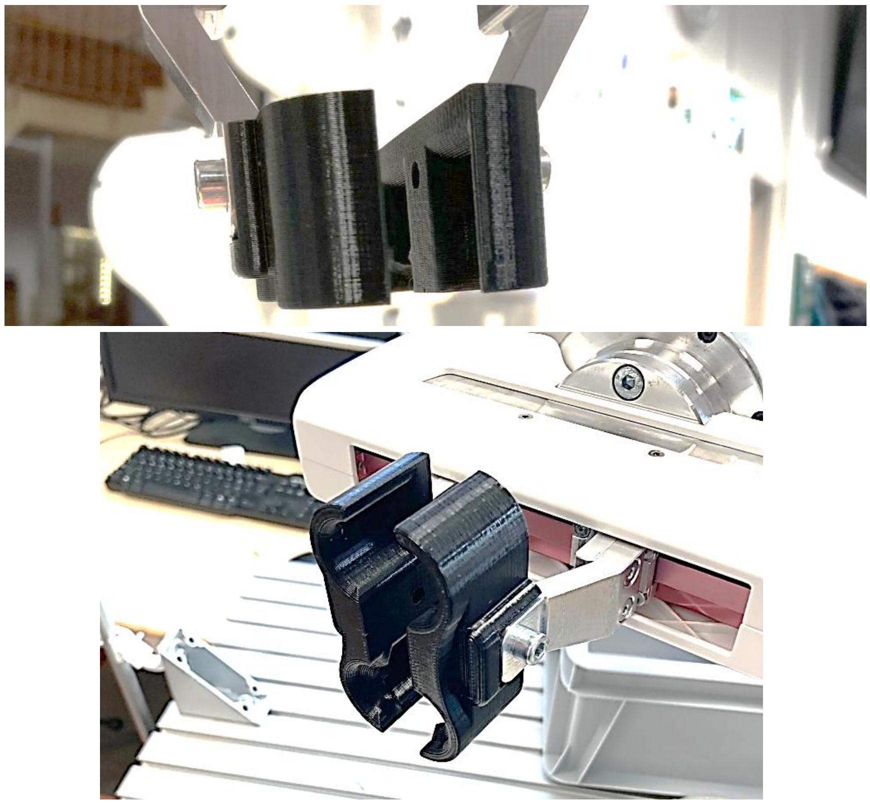

Figure 4: 3D-printed gripper

3.2 Buffer system

To store the parts, a buffer system was designed in CAD and subsequently 3D-printed. As shown in Figure 5, its slightly inclined slope ensures that parts can be picked from a consistent position at the bottom while the remaining parts slide down automatically. This design was used for all three colors that are to be assembled.

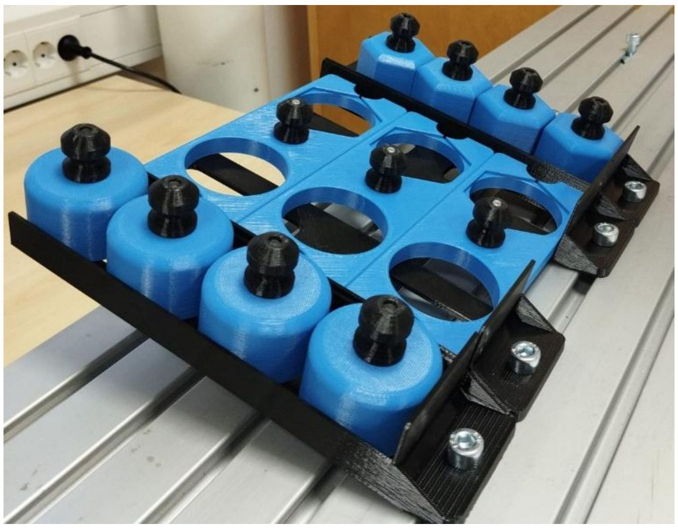

Figure 5: 3D-printed buffer system