International Elite Summer School Est. 2021 in Robotics & Entrepreneurship

#5_UR

MCP Server for Robot Tools (simulated + physical)

Context

Universal Robots is exploring how large language models (LLMs) can be connected to collaborative robots using the Model Context Protocol (MCP). MCP is an open standard that lets AI assistants interact with external tools and data sources through a unified interface. By wrapping robot capabilities like motion commands, sensor readings, and I/O control as MCP tools, a conversational AI agent can program and operate a robot using natural language alone.

In this case, students will build the bridge between LLMs and robots: an MCP server that exposes UR robot capabilities as tools callable by any MCP-compatible AI assistant. The vision is a future where robot programming requires no code, only conversation.

Challenge

Design and implement a Python MCP server that exposes UR10e robot capabilities

as LLM-callable tools. The server must communicate with the robot (or URSim) via the RTDE interface and allow a conversational AI agent to perform manipulation tasks through natural language prompting.

Minimum deliverable (low-level primitives):

- get_robot_state(): joint positions, TCP pose, velocities, status
- move_joint(q1..q6, speed, acceleration): joint space motion
- move_linear(x, y, z, rx, ry, rz, speed, acceleration): Cartesian motion
- stop(): halt robot motion
- (Optional) gripper_open() / gripper_close()

Stretch goal (compound tools):

- go_home: return robot to a predefined safe position
- teach_position(name): save the current pose by name for later reuse
- Conversational task execution: demonstrate the robot performing a multi-step task described purely in natural language

Keywords: MCP, LLM tool use, RTDE, agentic robotics, natural language programming

Tiers

Bronze: Get the MCP server running with the provided starter code. Connect it to an MCP client (Claude Code, OpenClaw, Cursor, or similar). Successfully call get_robot_state and move_joint through natural language.

Silver: Extend the server with 2-3 new tools beyond the starter set, for example move_circular, set_payload, or get_digital_inputs. Add input validation so the server rejects out-of-range values before they reach the robot. Document each tool with clear descriptions so the LLM understands when and how to use them.

Gold: Implement a trajectory tool that accepts a sequence of waypoints and executes them with configurable blending between points. Add real-time state streaming so the LLM can monitor progress during long moves. Demonstrate a multi-step task that uses blended trajectories.

Diamond: Build a safety layer into the MCP server that enforces workspace limits, monitors joint speeds, and performs forward kinematics checks before executing

any motion command. The server must reject commands that would move the TCP outside a defined safe zone or exceed speed thresholds. Demonstrate the safety layer catching and blocking an unsafe command issued by the LLM.

Tools, methods and materials

LLM backend: Students are responsible for sourcing their own LLM access. Free options include GLM-5.1 via NVIDIA NIM, Z.AI direct, Puter JS, and Modal. Students may also use any LLM they already have access to.

MCP client: Students choose their preferred MCP-compatible client. Options include OpenClaw, Claude Code CLI, Cursor, or any other tool that supports the MCP protocol.

Robot interface: A custom pure-Python RTDE client is provided for robot communication. The URScript and Dashboard interfaces are used for motion and control, and the FastMCP library is used for building the MCP server. All work with both URSim and the real UR10e.

Development environment: a URSim Docker container provided via docker-compose, started with a single command. All development and testing happens against URSim on the student's own laptop.

Validation: two UR10e robots are available for real-robot testing. Gripper behavior can be simulated using standard digital I/O or a timed wait.

From UR, the team will receive a docker-compose setup for URSim, a Python RTDE/URScript/Dashboard client library, starter MCP server code with example tools, and documentation on RTDE signals and the MCP protocol. Two UR10e robots are available for validation on real hardware. UR will be available to discuss details along the way.

View pdf

Back to industry cases