Drop-on-Demand Sample Delivery: Automation and Control of the DoD Robot

Scan Me!

Anas Jebali (1,2)- jebali@slac.stanford.edu, Sebastian Dehe(1)- dehe@slac.stanford.edu

1. SLAC National Accelerator Laboratory, USA. 2. Spartan Spark/San Jose State University

Introduction

The DoD robot could only be operated by calling its control API directly, there was no interface to monitor it, run its routines, or prevent an unsafe move. This internship built that missing layer: control software in Python and a graphical interface that let an operator monitor the robot and run its routines from a button, with safety built in at every step. The work went from a full offline model of the robot, to a control layer proven on the live hardware, to a GUI now used to operate it all, being built against the robot's real behavior, with every value read from the hardware rather than assumed.

WHAT'S THE DOD ROBOT ?

The DoD robot is a precision three-axis positioning system with a piezo-driven nozzle. On command, it ejects picoliter-scale droplets of sample directly into the X-ray interaction point.

Unlike a continuous liquid jet which flows constantly and wastes sample between pulses, the DoD robot dispenses only when a droplet is needed, conserving scarce material while still delivering fresh sample for each shot.

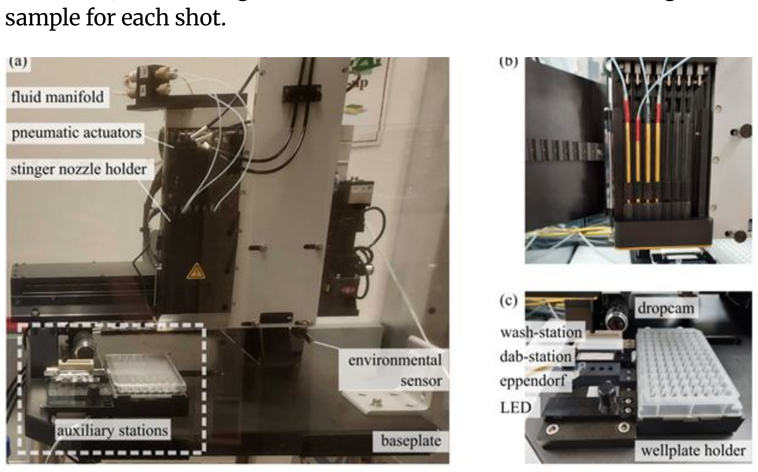

figure 1: The DoD robot at the MFX instrument.

HOW IT'S CONTROLLED

- The DoD robot is controlled through a stack of layers, from the interface down to the hardware
- The **GUI** is where the operator works, it calls the **Python DoD class**, which holds the control methods
- The DoD class runs inside the **LCLS Python middleware** (hutch-python), which connects out to the **robot's API**
- The **API** is the network gateway into the robot, it hands commands to the **vendor software** that drives the **hardware**
- Because everything above the API stays the same, a **dummy** can take the robot's place at the API level: running routines virtually, with no hardware

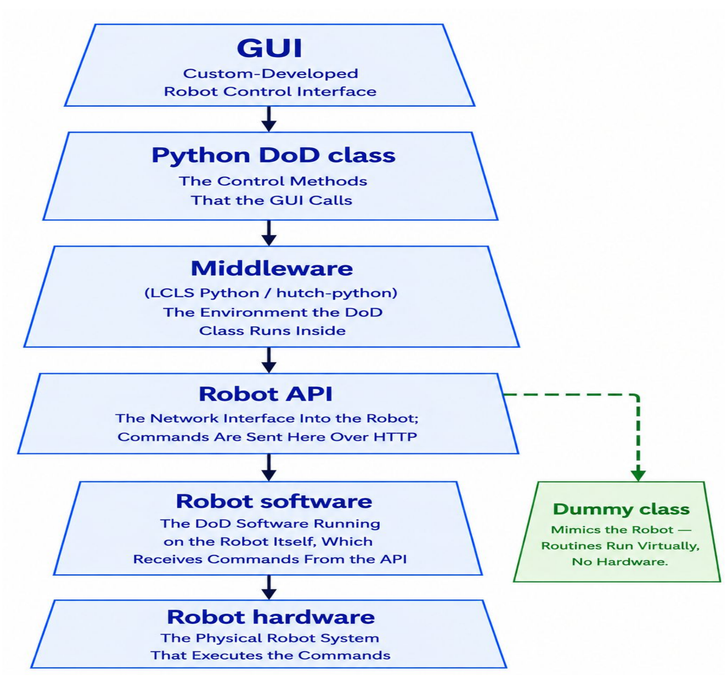

Figure 2. DoD system architecture.

OFFLINE DEVELOPMENT

- Built an offline version of the robot before working with the live hardware, to understand how it works and how its control interface is structured.
- Built a dummy robot that stands in for the real one, so routines could be built and tried out safely offline
- Built 2D and 3D visualizations of the workspace: the build plate, the named stations, and the robot's motion between them.
- Gave a clear picture of the robot's layout, named positions, and how a routine moves through the workspace.
- Made working with the real robot much more approachable

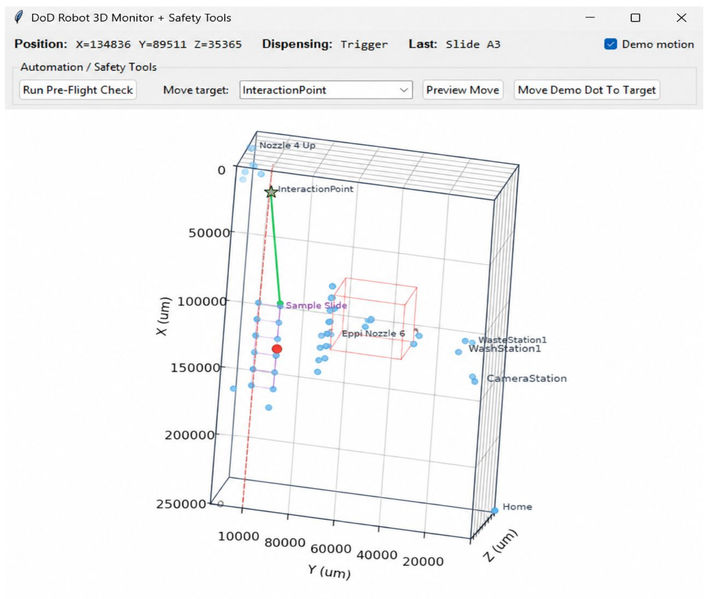

Figure 3: 3D visualization of the robot workspace and stations (offline model).

OBSTACLE-AVOIDING MOTION

- Integrated the team's safe-motion package into the interface to plan and preview collision-free paths
- The planner routes around exclusion zones: no-go regions the robot must avoid
- **Direct path** (dashed) shows the straight route; **safe path** (blue) shows the planned route that avoids every obstacle
- The path can be previewed before any motion, so the operator sees the intended route before committing
- Built on the robot's real coordinate frame and named positions, so previews reflect the actual workspace

Figure 4. Direct vs. safe path around the robot's exclusion zones.

THE CONTROL GUI

With hardware access, I connected to the live robot and built a graphical interface that replaces command-line calls with a monitored, interactive tool.

The design prioritizes safe, predictable operation: a dry-run mode previews any action before it executes, every motion requires explicit confirmation, and all readouts report measured values directly from the hardware. Commands execute on a background thread, keeping the interface responsive during motion.

Crucially, the interface is extensible: unlike the vendor's fixed software, new tasks and capabilities can be added as experimental needs evolve, since it is built on our own control layer.

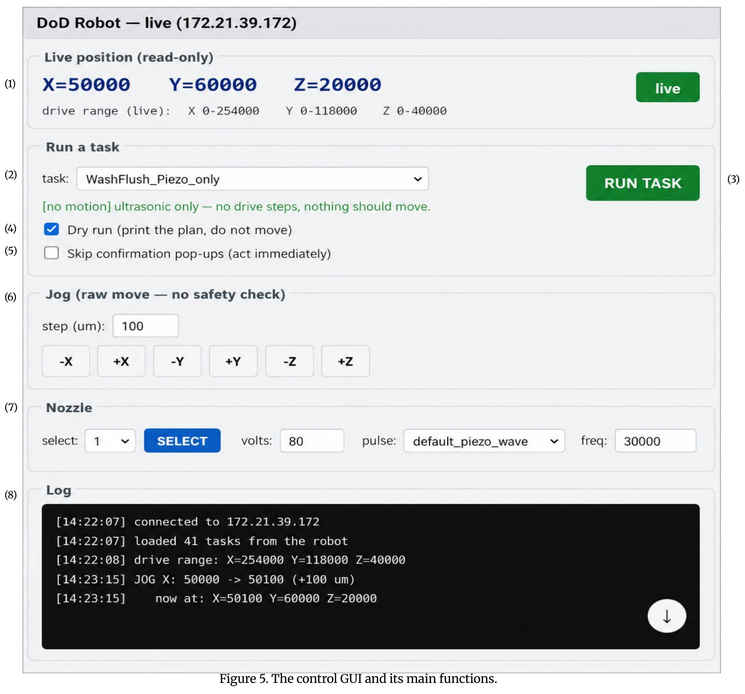

Figure 5. The control GUI and its main functions.

1) Live position: real-time X, Y, Z with drive-range limits; green "live" badge confirms it's current

(2) Run a task: dropdown of the robot's real routines, each risk-tagged

(3) RUN TASK: runs the selected task as one complete unit

(4) Dry run: prints the plan without moving; ON by default

(5) Skip confirmations: bypasses pop-ups for faster operation once trusted

(6) Jog: manual single-axis steps, clamped to the drive range

(7) Nozzle: selects the nozzle and sets voltage, pulse, and frequency

(8) Log: running record of every action and the robot's replies

Conclusion

- The interface makes the DoD robot easier and safer to operate. An operator can now run it without knowing the underlying control code, lowering the barrier for the whole team to use it
- Because it's built on our own control layer, it can keep growing with the experiment: new tasks and capabilities can be added as needs change, rather than being locked to fixed vendor software
- Bringing the software up against the real robot surfaced how the deployed system actually behaves, and that understanding is now captured in the tooling for whoever works on it next

Future Work

- Move from previewing safe paths to executing them, so the robot can navigate its workspace automatically
- Build full automated sequences that chain multiple tasks end-to-end
- Strengthen live status reporting and connection recovery for unattended runs

ACKNOWLEDGEMENTS

I would like to thank my mentor, Sebastian Dehe, for his guidance, dedication, and support throughout this internship, and for developing the safe-motion collision-avoidance package that this work integrates. I would also like to thank Josiah Kligmann for the discussions, ideas, and close collaboration throughout the day-to-day research, and the MFX instrument team at LCLS for their support. I am grateful to the Spark program at San Jose State University for the opportunity to take part in this internship.

REFERENCES

[1] D. DePonte, "Serial crystallography using automated drop dispensing," *J. Synchrotron Rad.* **28**, 1386–1392 (2021). DOI: 10.1107/S1600577521006160.
[2] S. Dehe, safe_motion: obstacle-avoiding motion package for the MFX DoD robot, SLAC (internal).
[3] DoD robot control API documentation, MFX instrument, LCLS.