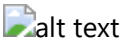
Sci-Agent-SPM



A **visual automation agent** for running **Scanning Probe Microscope (SPM)** instruments:

- **Pure GUI automation**: runs directly on the SPM controller software (e.g., Nanonis) as-is — **no instrument API** required, no NPI integration.
- **Fast setup**: calibrate a workspace once and start automating in minutes.
- **Long-horizon execution**: ReAct based operation that can carry multi-step experiments across different open-ended experimental scenarios and run stably over a long period of time. Translation: it helps you do experiments while you sleep.
- **Structured context + modular memory**: persistent sessions, structured run memory, and automatic memory compression.

What It Can Do

Sci-Agent-SPM is designed for UI-driven lab automation where "integration" is impossible or undesirable.

- **Type into fields** and **click buttons** using calibrated anchors.
- **Verify outcomes** by re-checking ROIs linked to the action (post-action observation is automatic when anchors have `linked_ROIs`).
- **Wait like an operator** with ROI-aware sleeps (`wait_until`) that can follow visible countdowns.
- **Run in different modes**:
  - `agent`: automation with tools (click/type/wait).
  - `chat`: model-only reasoning and planning (no UI side effects).
  - `auto`: the model classifies whether your message should be `agent` or `chat`.

How It Runs (Architecture)

At runtime, the agent is a tight "pixels in → actions out" loop:

- **Capture**: ROI screenshots via `mss` + `Pillow` (`src/capture.py`).
- **Act**: mouse/keyboard via `pyautogui` (`src/actions.py`) using your calibrated anchors.
- **Decide**: two-model design (`src/agent.py`):
  - `agent_model`: decides what to do next and updates structured memory.
  - `tool_call_model`: cheap "perception" model used for ROI reading / waiting decisions.
- **Tools**: exposed through an in-process **MCP server** so schemas are discoverable and the agent can "call tools" in a controlled way (`src/mcp_server.py`).
- **Memory**: structured session memory with optional "keep last N turns" and on-demand / threshold-based compression (`/compress_memory`).

Quickstart (Recommended)

From the repo root, run the bootstrap script:

```
.\Sci-Agent-SPM.ps1
```

It will:

- create `.venv` if needed
- install dependencies from `requirements.txt`
- create `workspace.json` from `workspace.example.json` (if missing)
- prompt for `OPENAI_API` (if missing) and start the TUI

Optional: install a user-level `Sci-Agent-SPM` command (copies a shim to `~\.local\bin` and adds it to your user `PATH`):

```
.\tools\install_sci_agent_spm.ps1
```

Setup (Manual)

Prerequisites

- Windows 10/11
- Python 3.11+ on PATH
- Your own SPM controlling software, (eg: Nanonis SPM Control Software) running on a stable monitor layout
- An OpenAI API key (`OPENAI_API` or `OPENAI_API_KEY`)

1) Create a venv + install deps

```
python -m venv .venv
.\.venv\Scripts\python -m pip install --upgrade pip
.\.venv\Scripts\python -m pip install -r requirements.txt
```

2) Calibrate your `workspace.json` (ROIs + Anchors)

This agent is deliberately “dumb” about UI structure: it only knows the rectangles and points you give it.

Option A (recommended): use the GUI calibrator:

```
.\.venv\Scripts\python -m src.calibrate_gui --workspace workspace.json
```

In the calibrator:

- Add/select an ROI or Anchor in the left list
- Click **Draw ROI box** (drag a rectangle) or **Pick anchor point** (single click)

- Describe the ROI as best as possible. For example:

```
"name": "set_current_readout",

"description": "ROI covering the displayed set current value (A) in Nanonis. It
does not always equal to the real current. Since this set current is only used
in the constant-current-scanning mode to set the current that STM tries to
maintain. But the actual real current depends on the tip's microscopical
geometry and the electronic status. "
```

- (Optional) For anchors, link ROIs in **Linked ROIs** so the agent auto-checks them after using that anchor
- Click **Save**

Option B: edit JSON directly:

1. Copy `workspace.example.json` → `workspace.json`
2. Edit:
    - `rois`: screenshot regions (readouts, status panels, countdowns)
    - `anchors`: click targets (input boxes, buttons)
    - `linked_ROIs` (anchor only): ROIs to observe after that action for verification

3) Set your OpenAI key

```
$env:OPENAI_API = "YOUR_KEY_HERE"
```

The bootstrap script also loads `OPENAI_API` / `OPENAI_API_KEY` from a local `.env` file if present.

Run

```
.\.venv\Scripts\python -m src.main --agent
```

How To Use It

1) Think in anchors + ROIs

The agent cannot "find the Bias field" unless you gave it:

- an anchor like `bias_input` (where to click/type)
- ROIs like `bias_readout`, `scan_status`, `scan_time_count_down` (what to verify / wait on)

Link the right ROIs to the right anchors to make verification automatic.

2) Give it a real operator command

In `agent` mode, you can ask for sequences like:

- "Set bias to 500 mV, start one topography scan, wait until status returns to `<idle>`, then set 400 mV and repeat."

In `chat` mode, ask for planning, SOP drafting, or sanity checks without touching the controlling software.

3) TUI controls (slash commands)

In the TUI, type `/help` (or `/menu`) to show commands. Settings persist in `sessions/.tui_settings.json`.

Core settings

- `/workspace [path]`: get/set the active workspace file.
- `/mode`: show current mode.
- `/mode agent|chat|auto`: set execution mode.
- `/agent_model [name]` (alias: `/model`): get/set the main "decision" model.
- `/tool_call_model [name]`: get/set the perception/tool helper model.
- `/max_agent_steps [int]`: limit steps per run (prevents runaway loops).
- `/action_delay [seconds]`: delay between UI actions (stability vs speed).
- `/abort_hotkey [on|off]`: enable/disable cooperative abort from the TUI (Ctrl+C).
- `/log_dir [path]`: set where runs write logs (default: `logs`).

Memory

- `/memory_turn`: show current `memory_turns`.
- `/set_memory_turn [-1|N]`: `-1` = full memory; `0` = none; `N` = keep last N entries.
- `/memory_compress_threshold [int tokens]`: set auto-compress threshold (`0` disables auto-compress).
- `/compress_memory`: manually compress memory now (moves details to archive and keeps summaries).

Sessions

- `/chat new`: start a new session (clears transcript + memory).
- `/chat save [name]`: save current transcript + agent state to `sessions/<name>.json`.
- `/chat list`: list saved sessions.
- `/chat resume <name>`: load a saved session.

Maintenance

- `/calibration_tool`: launch the calibrator for the current workspace.
- `/clear_cache`: delete log folders on disk (asks for confirmation).

4) TUI key bindings

- `Enter`: newline
- `Ctrl+S`: send input
- `Ctrl+I`: focus input

- `Ctrl+L`: focus transcript
- `Shift+Mouse`: select/copy transcript (Esc to close)
- `Ctrl+Q`: quit
- `PageUp` / `PageDown`: scroll transcript
- `Ctrl+C`: request abort (when `/abort_hotkey` is ON)

5) Logs (audit trail)

Each program run creates a timestamped folder under `logs/` (or your configured `/log_dir`).

Typical structure:

`logs/<YYYYMMDD_HHMMSS>/`

- `click_<anchor>/meta.json`
- `set_field_<anchor>/meta.json`
- `wait_until_<roi>_sleep/meta.json` + `before_<roi>.png`
- `observe_<attempt>_<roi>/meta.json` + `roi_<roi>.png`

These logs are designed to make automation reviewable: you can inspect exactly what the agent saw and did.

Safety

This project can control your mouse/keyboard.

- **PyAutoGUI failsafe**: move the mouse to the top-left corner to trigger `pyautogui.FAILSAFE`.
- Prefer running on a dedicated machine / dedicated desktop session so other apps don’t steal focus.
- Start with conservative `/action_delay` and small `/max_agent_steps` until your workspace is calibrated and stable.