ADCL Wind Tunnel — Operating & Contributor Manual

Hardware, control chain, code, maintenance

Kshitij Anand

Table of contents

1. ADCL Wind Tunnel — Operating & Contributor Manual

This manual is the single technical reference for the wind tunnel facility in the **Aeronautics, Dynamics and Controls Laboratory (ADCL)** at the University of California, Irvine. It covers the hardware chain, the control software, day-to-day operating procedures, and the maintenance practices needed to keep the facility working.

1.1 Audience

The manual is written for **project contributors**: anyone who will operate the tunnel beyond a one-off visit, modify the control software, recalibrate the system, or hand it off to the next student. If you only need to run an experiment safely, start with the **Standard Operating Procedure (SOP)** instead — it is a separate, operator-focused document with the safety checklist, emergency contacts, and the per-run sequence.

1.2 How to read this manual

The manual is task-oriented, not chronological. The milestone debug logs in `WT_MS_<n>/` capture how the facility got into its current state; this manual captures *what the current state is* and what you should do with it.

A reasonable first read:

1. **Safety** — non-negotiable. Read before touching anything.
2. **Hardware overview** — the airflow path, the electrical chain, and the network topology in one diagram.
3. **Software architecture** — when to use ADCL WinSoft, when to use `WindTunnelControl.ps1`, and why AeroWare is being retired.
4. **Pre-run checklist** + **Startup via ADCL WinSoft** — the golden path for an experiment.

If something breaks, jump straight to **Troubleshooting** and the **Reference** section.

1.3 What's *not* in this manual

- **Per-experiment data analysis** — that lives with each experiment, not in the facility manual.
- **CAD/model files for test articles** — those live under `models/` in the repository.
- **The milestone debug logs themselves** — see `WT_MS_<n>/debug_session.md`. The manual references them for historical context but is not a copy of them.

1.4 How this manual is built

The source lives at `docs/manual/content/` in the project repository. It is rendered as:

- a **static website** via MkDocs with the Material theme, searchable and navigable;
- a **single PDF** via `mkdocs-with-pdf`, suitable for printing or archival.

Both outputs come from the same Markdown files; there is no separate authoring step. See **Software → Building & deploying** for build instructions if you are picking this up after a `git pull`.

2. Safety

2.1 Safety overview

The ADCL wind tunnel is a low-speed open-circuit tunnel driven by a three-phase induction motor through an ABB ACH550 variable-frequency drive (VFD). It is not a high-energy facility, but every component listed below has injured someone in some lab somewhere; treat the safety guidance here as load-bearing, not decorative.

2.1.1 In an emergency

Situation	Action
Life-threatening injury, fire, electrical contact	**Dial 911** from a mobile phone, or **9-911** from a campus landline.
Hazardous materials / smoke / odor making people sick	Dial 911 if anyone is symptomatic; otherwise UCI EH&S **949-824-6200**.
Power must be killed immediately	Operate the **main switch** (see Test section & exhaust) to cut power to the VFD and motor.
Motor must be stopped immediately but power is fine	**Emergency Stop** in the control software (ADCL WinSoft red button, or `WindTunnelControl.ps1` E-Stop). See Emergency stop.

UCI EH&S full contact list and policy references are in the **SOP → Emergency contacts** doc. Verified 2026-05-14.

2.1.2 Hazard categories

Five hazard categories apply to this facility. Each has a dedicated page; the bullets below are the one-line summaries.

- **Electrical** — the VFD enclosure carries 480 V three-phase mains; the motor is mains-driven through it. Lockout/tagout discipline applies. Never open the VFD cabinet while the main switch is on.
- **Emergency stop** — three independent stop paths (software ramp-stop, software E-Stop, mains kill). Know all three before powering up.
- **Mechanical** — the fan moves significant air. Anything loose in the test section becomes a projectile; anything loose downstream becomes someone else's problem. The exhaust path is physically blocked by a band saw to deter people from walking into it (see Test section); keep that block in place.
- **Laser** — a 5 mW Class 2 laser is used for flow visualization. Class 2 is eye-safe via the blink reflex but follow the controls in the UCI Laser Safety Manual (alignment, signage, beam-path control).
- **Smoke generator** — glycol-based fog. Low toxicity but irritating; ensure ventilation, and stop the run if anyone reports symptoms.

2.1.3 The non-negotiables

1. **One person controls the tunnel at a time.** That person stays at the computer station for the entire run.
2. **Visual confirmation before pressing Start.** Test section is clear of people and tools; exhaust path is clear; band-saw block is in place.
3. **Both stop paths verified before the first powered run of the day.** Stop the motor from the software, confirm it stopped; then practice the mains kill location so muscle memory exists.
4. **Door placard up to date.** UCI EH&S requires a posted placard with PI name, hazard pictograms, and emergency instructions. Update it when hazards or contacts change.
5. **No running the tunnel alone after hours.** Have a second person reachable (in the building or by phone) any time the motor is energized outside normal working hours.

2.1.4 Training prerequisites

Before operating the tunnel solo, an operator must:

- complete UCI EH&S **Lab Safety Fundamentals**;
- complete UCI EH&S **Laser Safety** training if flow-visualization runs are planned;
- be briefed in person by an existing operator on the pre-run checklist and the emergency stop paths;
- demonstrate a startup → stop → mains-kill cycle while observed.

2.1.5 Documenting incidents

Any **near-miss or incident** — motor stalling under load, smoke alarm, unexpected fault code, a model coming loose, a person entering the exhaust path — is logged in a new `WT_MS_<n>/` milestone folder with a `debug_session.md`. The pattern is the same one used for the AeroWare diagnosis in `WT_MS_2/`: chronological notes, photos in `images/`, and a "Resolved / Open / Deferred" line at the bottom. See `docs/repository_layout.md` for the layout.

2.2 Emergency stop

There are three independent ways to stop the motor. They differ in reaction time, in what they leave the system in, and in when each is appropriate.

2.2.1 The three stop paths

Path	Action	Result	Use when
Software E-Stop	Press the red **Emergency Stop** button in ADCL WinSoft (VFD tab, right column) or in `WindTunnelControl.ps1`.	Writes Control Word `0x0000` and `REF1 = 0` — drive coasts to stop with no ramp.	A person is in danger or hardware integrity is at risk and seconds matter.
Software ramp-stop	Press **Stop Tunnel** in the same panel.	Writes Control Word `0x0476` — drive decelerates per its programmed ramp (parameter 22.02 `DECEL TIME`). Motor remains energized but at zero RPM until commanded to start again.	Normal end-of-run.
Mains kill	Operate the **main switch** on the wall to the building's three-phase feed for the tunnel.	Removes power from the VFD and motor. Coasts to stop; software loses Modbus connection.	Software is unresponsive, smoke or fire is present, or both stop buttons have failed.

2.2.2 Sequence in a real emergency

1. **Stop the motor first** — software E-Stop if you can reach it, mains kill otherwise. Stopping the motor stops the airflow, which stops the laser/smoke/projectile chain reaction.
2. **Then call for help** — 911 (or 9-911 from a campus landline) if anyone is hurt, EH&S 949-824-6200 otherwise.
3. **Then secure the area** — keep people out of the test section and the exhaust path until the motor is fully spun down.
4. **Then power off cleanly** — mains kill if not already done, then close the control software, then de-energize ancillary equipment (laser, smoke generator, lights).
5. **Then log it** — `WT_MS_<n>/debug_session.md` with what happened, what you did, and the system state at the time.

2.2.3 Why the three paths are independent

- The **software E-Stop** writes Modbus registers over the lab Ethernet to the RETA-01 fieldbus adapter, which writes the drive's control word. It depends on the PC, the Ethernet adapter, the RETA-01, and the drive being powered.
- The **software ramp-stop** uses the same path but a different control word and therefore relies on the drive's deceleration ramp.
- The **mains kill** is a hard electromechanical disconnect upstream of everything else. It is the path of last resort because it is also the slowest to recover from (need to power-cycle, re-establish Modbus, etc.) and it does not give the motor a graceful ramp.

2.2.4 Pre-run verification

The first run of the day, with no model in the test section and the tunnel at low RPM (e.g. 100 RPM setpoint):

Software ramp-stop → motor decelerates smoothly to zero.

Restart, software E-Stop → motor stops faster than the ramp.

Restart, mains kill → motor coasts down, drive loses Modbus (verify the control software reports the disconnect).

If any of the three paths fails, **do not load a model and do not run a real experiment.** Move to Troubleshooting or stop for the day.

2.3 Electrical hazards

The wind tunnel draws three-phase mains power to a VFD which drives a three-phase induction motor. Treat the VFD enclosure as a high-voltage cabinet at all times.

2.3.1 What is energized when

Component	Energized when	Notes
Main switch upstream of the VFD	Always, until physically opened	The mains feed is live at the switch terminals even when the switch is open. Treat as live.
VFD enclosure busbars	Main switch closed	Three-phase 480 V to the input rectifier.
VFD DC link	Main switch closed; remains charged briefly after opening	Bus capacitors retain charge. Vendor minimum wait is **5 minutes** after de-energizing before opening the cabinet.
Motor terminals	Main switch closed; only spinning when drive is commanded	The motor terminals can carry voltage while the drive is enabled even if the rotor is not turning.
RETA-01 fieldbus adapter, Ethernet 4 adapter	Lab PC powered (low voltage)	Communication-side only; safe to handle.

2.3.2 Rules

1. **Never open the VFD enclosure with the main switch closed.** No exceptions.
2. **After opening the main switch, wait at least 5 minutes** before opening the VFD cabinet, to allow the DC-link capacitors to discharge.
3. **Lockout/tagout** the main switch any time the cabinet must be opened: a physical lock through the switch handle plus a tag identifying who placed it. UCI EH&S policy reference: Hazardous Energies / LOTO program.
4. **Do not touch motor terminals while the drive is enabled**, even at zero commanded speed. The motor sees PWM voltage whenever the drive is enabled.
5. **Inspect cabling before each run.** Visible damage to any power cable means: stop, isolate, repair, then resume — never tape over and continue.
6. **No liquids near the VFD or the main switch.** Smoke-generator fluid spills are an electrical hazard; clean immediately with the run paused.

2.3.3 Arc-flash and high-voltage references

- UCI EH&S Electrical Safety Program (PDF): https://ehs.uci.edu/programs/_pdf/safety/electrical-safety-program.pdf
- UCI EH&S High-Voltage Electrical Safety Program (PDF): https://www.ehs.uci.edu/programs/_pdf/safety/high-voltage.pdf
- UCI EH&S Energized Electrical Work Reference Guide (PDF): https://www.ehs.uci.edu/safety/_pdf/Energized-Electrical-Work-Guidance.pdf

2.3.4 What's inside the VFD

The drive is an **ABB ACH550** configured for fieldbus control via a RETA-01 module in option slot 1. Critical parameter values that *must* be present for remote operation are documented in Reference → Parameter index. If any of them drift, Maintenance → Parameter recovery is the recovery path.

The drive is *configured*, not *programmed*, through its front keypad. Never modify parameters from the keypad without recording the change in a milestone log — the parameter map is the difference between "the tunnel works" and "the tunnel does not work" and there is no automated backup.

2.4 Laser and smoke generator

Flow visualization uses a small green laser to illuminate a sheet of glycol smoke. Both components are low-hazard for trained operators but can cause real injuries when mishandled — eye damage from beam mis-alignment, respiratory irritation from over-fogged enclosed spaces.

2.4.1 Laser

Current installation: a **5 mW Class 2 laser**, powered from a DC bench supply preset to **12 V**, with a **glass rod** placed in front of the beam to spread it into a sheet for slicing the smoke (see Hardware → Laser & smoke generator for photos).

Hazard class

Class 2 (visible, < 1 mW continuous wave per ANSI Z136.1) is considered **eye-safe via the blink reflex** at the cornea: looking directly into the beam for less than 0.25 s does not damage the retina. The 5 mW device in this lab is at the upper end of Class 2 (some references classify it as 3R); treat it conservatively.

Rules

1. **No staring into the beam, even briefly.** Blink reflex assumes an involuntary blink within 0.25 s; intentional staring overrides it.
2. **No reflective objects in the beam path.** Watches, jewellery, polished metal — remove or cover. The beam reflects off curved surfaces at angles that can find an eye.
3. **The beam must terminate on a diffuse, non-reflective surface.** Black-anodized aluminium or matte black painted card, not bare metal or glossy plastic.
4. **Door placard updated** when the laser is in use: "Laser in use — Class 2, do not enter without authorization."
5. **Eyewear** is not strictly required for Class 2 but is recommended during alignment. UCI EH&S can advise on appropriate optical density if alignment becomes a frequent activity.
6. **Power supply 12 V exactly** — do not raise the voltage. Higher voltage will raise the optical power output past Class 2.

Current known issue

The glass-rod sheet former produces a sheet with intensity concentrated at the centre rather than uniform across the sheet. This is the subject of the Project improvements thread. For now, position the test article within the centre of the sheet where intensity is highest, and do not expect quantitative streamline analysis from the current optics.

Reference

- UCI EH&S Radiation Safety: https://ehs.uci.edu/radiation-safety/index.php
- UCI EH&S Laser Safety Manual (PDF): https://www.ehs.uci.edu/radiation-safety/_pdf/laser-safety-manual.pdf
- Laser Safety Officer: Bridgette Neri (`nerib@uci.edu`); general inbox `radsafety@uci.edu`.

2.4.2 Smoke generator

Current installation: small commercial smoke generator with **wired remote** placed at the computer station; generator itself sits near the tunnel inlet. Current settings are recorded in the Hardware → Laser & smoke generator photo.

Hazard

Glycol smoke is **low-toxicity** but **respiratory-irritating** in enclosed spaces. Symptoms of overexposure are coughing, eye irritation, and shortness of breath. The smoke generator vents into the tunnel inlet and is carried through the test section to the exhaust path, so the operator is usually downstream of the smoke — but a stalled tunnel or a back-eddy can put smoke into the operator's breathing space.

Rules

1. **The tunnel must be running before the smoke generator is enabled.** Smoke is meant to be entrained by airflow; without airflow it pools in the room.

2. **Stop the smoke generator before stopping the tunnel.** Order of shutdown: smoke off → wait for the test section to clear → tunnel off. See Operations → Shutdown.
3. **Stop the run immediately** if anyone reports respiratory irritation or eye irritation. Open the lab door for ventilation. UCI EH&S "Who Do I Call?" odor / irritation contact: `safety@uci.edu`, 949-824-6200.
4. **No smoke generator while the door placard does not warn of fog.** Class 2 laser plus smoke can trip a fire alarm if a stray detector is in the airflow path; coordinate with building management if you're not sure.
5. **Refill the glycol reservoir in the off state with the unit unplugged.** Spills onto the heater element are an electrical hazard and a smoke-quality hazard (uneven dispersion next run).

Optimizing settings

Smoke density, particle size, and dispersion stability depend on the generator's settings (heater temperature, fluid flow rate, fan speed) and on the ambient tunnel airflow. The current settings (photographed) are a starting point, not a calibrated optimum. The Project improvements thread plans an empirical sweep.

2.4.3 Door placard policy

The lab door placard must reflect the *currently active hazards*. When the laser or smoke generator is in use, the placard must show:

- the laser pictogram and "Class 2 in use" annotation;
- the smoke / fog warning;
- the PI name and emergency contacts;
- the date of last chemical inventory.

UCI placard policy and the generator app are at https://www.ehs.uci.edu/research-safety/lab-safety-inspections/labels-signs.php. The full reference guide is the Laboratory Door Placards Reference Guide (PDF).

3. Hardware

3.1 Hardware overview

The wind tunnel is an open-circuit, low-speed facility. Air is drawn from the room through the intake, accelerated through the contraction, passed across the test section, and exhausted to the back of the lab. A three-phase induction motor drives the fan; an ABB ACH550 VFD regulates motor speed; a Modbus/TCP fieldbus connects the lab PC to the VFD; the lab PC runs the control software and the data acquisition.

3.1.1 Airflow path

```
flowchart LR
  IN[Intake<br/>+ smoke generator] --> CN[Contraction] --> TS[Test section<br/>+ laser sheet] --> DF[Diffuser] --> FN[Fan / motor] --> EX[Exhaust<br/>blocked off by band saw]
```

The smoke generator sits **near the intake** so its plume is entrained into the main airflow. The laser sheet illuminates the **test section** from the side. The fan and motor are downstream of the test section, so airflow blockage from a loose model immediately loads the motor; this is one reason why the **pre-run checklist** is non-negotiable.

3.1.2 Electrical & control chain

```
flowchart LR
  MAINS[Building 480 V three-phase] --> SW[Main switch]
  SW --> VFD[ABB ACH550 VFD<br/>option slot 1: RETA-01]
  VFD --> MOT[3-phase induction motor]
  MOT --> FAN[Tunnel fan]

  PC[Lab PC<br/>Win11 + WSL2] -->|USB| ETH[Ethernet 4<br/>Realtek 2.5GbE<br/>192.168.50.100]
  ETH -->|Cat 5e| RETA[RETA-01<br/>192.168.50.10]
  RETA -.->|internal bus| VFD

  PC -->|USB| CDAQ[NI cDAQ chassis]
  CDAQ --> S1[NI-9203 4–20 mA]
  CDAQ --> S2[NI-9215 ±10 V]
  CDAQ --> S3[NI-9237 bridge]
```

The main switch is the **only** physical disconnect upstream of the drive. Everything to the right of `SW` is energized once the switch is closed.

3.1.3 Network topology

The lab PC's onboard Ethernet talks to the campus network. A separate USB-to-Ethernet adapter (the operating system labels it **Ethernet 4**, MAC `6C:1F:F7:C3:96:59`) carries the private VFD network on a static `192.168.50.100/24`. The RETA-01 inside the drive holds `192.168.50.10/24`. Both are configured to use the **ABB Drives** Modbus profile on port 502, unit ID 1.

The reason for the separate adapter is that the campus network's DHCP would assign an address in a different subnet and the RETA-01 has no route there. Treating the VFD network as a separate physical link with a static address means there is no DHCP race, no campus-network firewall, and no risk of someone else's traffic confusing the drive.

3.1.4 Components at a glance

Component	Make / model	Purpose	Manual page
Main switch	Building install	Mains disconnect	Test section & exhaust
VFD	ABB ACH550	Speed regulation of the motor	Motor & VFD
Fieldbus adapter	ABB RETA-01	Ethernet → drive internal bus	Fieldbus
Motor	3-phase induction, nameplate ~880 RPM at 60 Hz	Drives the fan	Motor & VFD
Lab PC	Windows 11 with WSL2 sidecar	Control + DAQ host	Control PC
Ethernet 4	Realtek USB 2.5 GbE	VFD network adapter	Control PC
cDAQ chassis	NI compactDAQ	Frame for the DAQ modules	Data acquisition
NI-9203	8-channel, 4–20 mA current input	Pressure transducers	Data acquisition
NI-9215	4-channel ±10 V voltage input	Angle, miscellaneous voltages	Data acquisition
NI-9237	4-channel bridge input	PGB sting force balance	Data acquisition
Laser	5 mW Class 2 + glass-rod sheet former	Flow visualization	Laser & smoke
Smoke generator	Glycol-based, wired remote	Flow visualization	Laser & smoke

3.1.5 Physical layout in the lab

The lab is one rectangular room. The **computer station** is at one end (PC, smoke generator remote, ADCL WinSoft on screen); the **wind tunnel** runs the length of the room with the intake near the computer station and the exhaust at the far wall. The **VFD enclosure** is mounted on the wall next to the tunnel. The **main switch** is mounted within reach of the computer station so the operator can disconnect mains without moving away from the controls.

The exhaust opening at the far wall is blocked by a parked **band saw** to deter people from walking into the airflow path. The band saw is there as a physical barrier, not as a tool to be used during a tunnel run — it must stay in place while the tunnel is energized.

For high-resolution photos of each station, see the dedicated pages:

- Control PC
- Motor & VFD
- Laser & smoke generator
- Test section & exhaust

3.2 Motor and VFD

3.2.1 Motor

The fan is driven by a three-phase induction motor. The nameplate parameters that matter for control are stored in drive parameter group 99:

Parameter	Value	Meaning
`99.07 MOTOR NOM FREQ`	60 Hz	Nameplate frequency (read at the drive).
`99.08 MOTOR NOM SPEED`	880 RPM (approx.)	Nameplate synchronous speed used as the 100 % reference in fieldbus scaling. The `WindTunnelControl.ps1` startup probe reads this at launch; if the probe fails, the PowerShell tool falls back to a hardcoded 1800 RPM.

The 880 RPM nameplate is consistent with a 4-pole motor at 60 Hz with slip. The operating range used in software is **0–880 RPM** with a hard clamp.

The motor itself is a sealed industrial unit; there are no user-serviceable parts. The only routine task is checking the belt / coupling alignment as part of the maintenance schedule — see Maintenance → Schedule.

3.2.2 VFD: ABB ACH550

The drive runs the motor from the building's three-phase mains and accepts speed commands from the lab PC over Modbus/TCP via the RETA-01 fieldbus adapter in option slot 1.

Control profile

The drive is configured for the **ABB Drives** profile. This profile uses fixed quick-access registers for the control word, the reference, the status word, and the actual values; parameter reads/writes use a different mapping (see Reference → Register map).

Quick-access registers:

Modbus address	Name	R/W	Use
0	Control Word (CW)	R/W	Start / stop / emergency-stop bit pattern.
1	Reference 1 (REF1)	R/W	Speed reference, 0–20000 scaled.
2	Reference 2 (REF2)	R/W	Unused.
3	Status Word (SW)	R	Drive state bits (READY, RUN, FAULT, REMOTE, etc.).
4	Actual 1 (ACT1)	R	Speed feedback.
5	Actual 2 (ACT2)	R	Current feedback.

Control word patterns

Pattern	Meaning	When written
`0x0476`	Prepare to run / ramp stop	Before transitioning to run; also used to ramp down for normal stop.
`0x047F`	Run	After REF1 and the prepare-word are both set.
`0x0000`	Off (coast / E-Stop)	Emergency stop; drive coasts to zero, no ramp.

Start sequence

The drive requires the following sequence to spin the motor from idle:

1. Write **REF1** to the desired scaled reference.
2. Wait at least 200 ms.
3. Write **CW = 0x0476**.
4. Wait at least 500 ms (allows the drive to internally clear OFF2/OFF3).
5. Write **CW = 0x047F**.

For **mid-run setpoint changes**, write **REF1 only**. Writing `0x0476` again resets the run state and the drive will stop until `0x047F` is reissued. This footgun is the source of the original "set RPM live" bug in the early ADCL WinSoft prototype and is documented in `code/adcl_winsoft/vfd/controller.py`.

Critical drive parameters

These must be set for fieldbus control to work. If any drift, see Maintenance → Parameter recovery. The full list is in Reference → Parameter index.

Parameter	Value	Meaning
`99.02 APPLIC MACRO`	1	ABB Standard macro.
`98.02 COMM PROT SEL`	4	External fieldbus (RETA-01) is active.
`10.01 EXT1 COMMANDS`	**10**	Start/stop commands come from the fieldbus. **If this drifts, the motor cannot be started from software.**
`11.02 EXT1/EXT2 SEL`	0	EXT1 is the active command source.
`11.03 REF1 SELECT`	**8**	REF1 (speed setpoint) comes from the fieldbus. **If this drifts, software writes to REF1 are ignored.**
`22.02 DECEL TIME`	site-specific	Controls the ramp-stop time. Confirm before changing — too short causes overvoltage faults at high RPM.

Mode indicator on the drive keypad

The drive's keypad shows `REM` in the upper-left corner when control is delegated to the fieldbus. **The motor cannot be started from software unless the keypad shows `REM`.** If it shows `LOC`, press the `LOC/REM` button on the keypad to switch to remote mode.

Calibration

The mapping from `drive_RPM` to the REF1 register value was measured empirically in `WT_MS_2`:

$$ \mathrm{REF1} = \mathrm{drive\_RPM} \times 22.42 $$

with a residual error of about ±1 %. This is consistent with ABB's nominal scaling (20000 = 100 % = motor nameplate RPM ≈ 880 → 20000/880 ≈ 22.7) to within 1.4 %. The exact constant `22.42` is used in both control codebases (`code/adcl_winsoft/vfd/controller.py`, `code/wind_tunnel_control/WindTunnelControl.ps1`).

A second mapping, from `drive_RPM` to **wind speed**, is documented separately at Reference → Calibration constants.

3.3 Fieldbus: RETA-01

The **RETA-01** is an Ethernet fieldbus adapter from ABB, installed in **option slot 1** of the ACH550 drive. It bridges between the lab PC's Ethernet network and the drive's internal control bus. To the PC it looks like a Modbus/TCP slave on `192.168.50.10:502`, unit ID 1.

3.3.1 Identification

Item	Value
Firmware revision	`0x0130` (1.30)
MAC address	`00:30:11:16:3E:3A`
IP address	`192.168.50.10`
Subnet mask	`255.255.255.0`
Gateway	none configured (private link)
Modbus profile	ABB Drives (`51.16 PROTOCOL = 0`)
Unit ID	1

These values were recovered during the `WT_MS_1` bring-up session and are stored in drive parameter group 51 (see Reference → Parameter index).

3.3.2 DIP switches

There is a single block of DIP switches (`S1`) on the RETA-01 board. All of them are set to **OFF**. With all switches off, the adapter takes its IP configuration from the drive's group 51 parameters; setting any of them on overrides parameters with on-board defaults and is **not what we want**.

If the adapter ever powers up with the wrong IP, the first thing to check is that `S1` is still all off.

3.3.3 LED indicators

LED	Healthy state	Meaning
LINK / ACTIVITY	Green, blinking with traffic	Ethernet cable is connected and packets are flowing.
MODULE STATUS	Green, steady	Adapter is configured and talking to the drive.
NETWORK STATUS	Green, steady	At least one Modbus master has an active connection.

A **red MODULE STATUS** is the diagnostic to watch — it means the adapter cannot reach the drive over the internal bus, which is usually a seating or cabling issue inside the cabinet. If it appears, see Troubleshooting → Modbus connection.

3.3.4 Reaching the RETA-01 from the lab PC

The lab PC's `Ethernet 4` adapter holds the private side: static `192.168.50.100/24`. There is no DHCP server on this segment and no need for one. From PowerShell:

```
Test-NetConnection -ComputerName 192.168.50.10 -Port 502
```

should report `TcpTestSucceeded : True`. If it does not, the problem is one of:

1. Ethernet 4 has lost its static IP. Reassert with the connection script in `code/wind_tunnel_control/` or via Windows Settings.
2. The RETA-01 has lost power (drive is off, or the adapter has come unseated).
3. The cable is unplugged or damaged.

3.3.5 Why a persistent TCP connection matters

The RETA-01 is happy to accept one Modbus/TCP connection at a time. Software that opens a fresh TCP socket for every register read or write — which is how AeroWare was written — ends up retransmitting frequently and stalling on TCP handshake delays of ~9 s. This is the failure mode that ultimately motivated the move from AeroWare to ADCL WinSoft.

The replacement code (`code/adcl_winsoft/vfd/modbus_client.py`) maintains **one persistent TCP socket per `VfdClient` instance** with a `threading.Lock` around the wire-format writes. The PowerShell tool (`code/wind_tunnel_control/WindTunnelControl.ps1`) takes the simpler approach of opening a fresh socket per call — that is acceptable at 4 Hz polling but would not scale to a faster loop.

This is captured in the pymodbus quirks note and is the most important architectural decision in the VFD layer.

3.3.6 Parameter mapping

To read or write a drive parameter (not a quick-access register) over Modbus, the address is:

$$ \mathrm{address} = \mathrm{group} \times 100 + \mathrm{index} - 1 $$

For example, parameter `03.05 FB REF 1` is at Modbus address `3 \times 100 + 5 - 1 = 304` . This is in the register map reference.

3.4 Control PC

The lab's control PC is a Windows 11 desktop that runs the control software, the data acquisition, and a WSL2 sidecar for shell-side scripting. It is the same machine that hosts this repository's working copy.

3.4.1 Operating system layout

Layer	Role
Windows 11 (host)	Runs the control software, the NI-DAQmx driver, the USB-to-Ethernet adapter for the VFD network.
WSL2 (Ubuntu)	The shell of choice for repository work — `git`, editor, file management, MkDocs builds. WSL2 cannot directly drive the USB Ethernet adapter; PowerShell scripts that talk to the VFD must be invoked from the Windows side.
NI-DAQmx Runtime	National Instruments user-mode driver for the cDAQ chassis. Required for ADCL WinSoft's real-DAQ mode; without it the application falls back to a simulated DAQ.

3.4.2 Network adapters

The PC has at least two Ethernet interfaces:

Interface	Address	Purpose
Built-in Ethernet	DHCP from campus	Internet, campus services.
Ethernet 4 (USB)	static `192.168.50.100/24`	VFD private network.

The USB-to-Ethernet adapter is a **Realtek 2.5 GbE** part. Its MAC is `6C:1F:F7:C3:96:59`; identifying by MAC is more reliable than identifying by index when Windows renumbers the adapters after a reboot.

If `Ethernet 4` loses its static IP, the symptom is that the control software cannot reach the drive at `192.168.50.10`. The fix is to re-assert the static address either through Windows network settings or via the connection helper script in `code/wind_tunnel_control/`.

3.4.3 Software footprint

The PC carries three control-software stacks at the time of writing. Only one of them is intended to remain after the AeroWare retirement.

ADCL WinSoft (current, long-term)

- **Install path:** `E:\Wind_tunnel\ADCLWinSoft\ADCL WinSoft.exe`
- **Data path:** `E:\Wind_tunnel\ADCLWinSoft\Data\YYYY-MM-DD\`
- **Config path:** `%APPDATA%\AeroLab\ADCLWinSoft\settings.json`
- **Admin password hash:** `%APPDATA%\AeroLab\ADCLWinSoft\admin.hash` (Argon2id)
- **Build environment (dev):** Python 3.12 venv at `C:\Users\kshit\.venvs\adcl_winsoft\`
- **Source repo:** this repository, at `code/adcl_winsoft/`. Full overview in Software → ADCL WinSoft.

WindTunnelControl.ps1 (fallback)

- **Deployment path:** `E:\Wind_tunnel\AeroWare\WindTunnelControl.ps1` (alongside the legacy AeroWare install for convenience).
- **Source:** `code/wind_tunnel_control/WindTunnelControl.ps1`.
- **Use when:** ADCL WinSoft is unavailable, or for quick manual verification of the VFD path. Full overview in Software → WindTunnelControl.ps1.

AeroWare (legacy, being retired)

- **Install path:** `E:\Wind_tunnel\AeroWare\AeroWare.exe`
- **Config path:** `E:\Wind_tunnel\AeroWare\` (multiple `.ini` files, see AeroWare install layout note)
- **Status:** The VFD-control path is broken in place — AeroWare writes to a stale IP (`192.168.1.1`) and never falls back. The DAQ path still works. AeroWare can be retired once ADCL WinSoft has covered the DAQ + calibration workflows that AeroWare currently handles. See Operations → Start via AeroWare (legacy).

3.4.4 Repository placement on the PC

The repository is cloned into the WSL2 home (`~/wind_tunnel_maintenance/`) for editing, and the `code/adcl_winsoft/deploy/install_dev.cmd` helper syncs source into the Windows venv via `xcopy` when needed. The reason for the copy step rather than `pip install -e .` over the WSL UNC share is that the share is slow and `pip install -e` over it is unreliable — see the PyInstaller quirks note.

3.4.5 Data layout

What	Where	Created by
Live experiment CSVs	`E:\Wind_tunnel\ADCLWinSoft\Data\YYYY-MM-DD\DDMMYY HHMMSS.fff<tag>.csv`	ADCL WinSoft
Legacy AeroWare CSVs	`E:\Wind_tunnel\AeroWare\Data_and_others\`	AeroWare
PGB sting calibration matrices	`E:\Wind_tunnel\AeroWare\Configs\N_C1_inv*.csv`	Loaded by both AeroWare and ADCL WinSoft
Milestone captures (pcap, sweeps)	this repository under `WT_MS_<n>/captures/`	Manual

3.5 Data acquisition

Process variables (pressures, model angle, force-balance loads) are read through a National Instruments **compactDAQ** (cDAQ) chassis hosting three input modules. The cDAQ is connected to the lab PC by USB; both ADCL WinSoft and AeroWare use the NI-DAQmx driver to talk to it.

3.5.1 Chassis and modules

Slot	Module	Channels	Range	Used for
1	**NI-9203**	8	4–20 mA current input	Pressure transducers (static ring, model pressure).
2	**NI-9215**	4	±10 V differential	Pitch angle sensor, miscellaneous voltages.
3	**NI-9237**	4	Bridge ±25 mV/V	PGB sting force balance (normal, axial, moment).

Channels are addressed in NI-DAQmx as `cDAQ1Mod<N>/ai<M>`. ADCL WinSoft's default channel map is in `code/adcl_winsoft/src/adcl_winsoft/daq/channel_map.py`.

3.5.2 Sample rate and continuous mode

ADCL WinSoft reads at **5 Hz** by default. The NI-9237 requires a **continuous** sample clock — single-point reads are not supported by the module — so the DAQ worker (`code/adcl_winsoft/src/adcl_winsoft/daq/cdaq_reader.py`) opens a continuous task and reads one sample per polling tick.

Sample rate is configurable per channel in the settings file but kept low because the physical phenomena of interest (steady-state pressure, slowly varying angle, average force) do not need high temporal resolution.

3.5.3 Calibration

The cDAQ produces **raw electrical** values: mA from the NI-9203, V from the NI-9215, mV/V from the NI-9237. ADCL WinSoft converts these to **engineering units** in two steps:

1. **Per-channel linear calibration** (slope + offset) applied in `daq/calibration.py`. Slope and offset come from the channel map.
2. **Multi-channel matrix calibration** for the PGB sting (`N_C1_inv.csv` and optional `N_C1_inv_C2.csv` matrices) — converts the three bridge readings to forces and moment. These matrices are inherited from the AeroWare install and live in `E:\Wind_tunnel\AeroWare\Configs\`.

Velocity from the static ring uses **Bernoulli** with a Sutherland-viscosity correction; the model is in `daq/calibration.py`.

3.5.4 Simulated DAQ

If the `nidaqmx` package is not importable (e.g. on a Linux dev machine), or if the environment variable `ADCL_WINSOFT_SIMULATE=1` is set, the DAQ worker degrades gracefully to a synthetic signal generator. This is what makes it possible to develop and test the UI on this Mac without the lab hardware. The simulated mode is signalled in the application's footer bar so the operator cannot confuse it for real data.

3.5.5 Data outputs

Each acquisition run produces a single CSV file in:

```
E:\Wind_tunnel\ADCLWinSoft\Data\YYYY-MM-DD\DDMMYY HHMMSS.fff<tag>.csv
```

The filename format is inherited from AeroWare for backward compatibility with existing post-processing scripts. The columns are: timestamp, raw channels (mA / V / mV/V), engineering-unit channels (Pa, deg, N, N·m), and the derived signals (velocity, Reynolds number).

The default data root is `E:\Wind_tunnel\ADCLWinSoft\Data\`; it is configurable in `%APPDATA%\AeroLab\ADCLWinSoft\settings.json`. Migrations from AeroWare's old paths (`E:\Wind_tunnel\Data`, `E:\Wind_tunnel\AeroWare`) are handled silently in `config/settings.py`.

3.6 Laser and smoke generator

This page documents the physical installation. Hazard handling and rules are on the safety page: see Safety → Laser & smoke. The future of this subsystem is in the Project improvements → laser sheet uniformization and Project improvements → smoke rack threads.

3.6.1 Laser system

Item	Spec
Laser	5 mW, Class 2, visible (green)
Power supply	DC bench supply, **preset to 12 V**
Beam shaping	Glass rod placed across the beam to spread it into a sheet
Mounted on	Optical bench / aluminium extrusion to the side of the test section

Known issue: non-uniform sheet

The glass-rod sheet former produces a sheet with intensity **concentrated in the middle**. This is acceptable for qualitative flow visualization on the centreline of the test section, but it is not suitable for quantitative streamline extraction from video — the intensity falloff at the edges of the sheet makes a constant-threshold particle-tracking algorithm unreliable.

Replacing the glass rod with a **Powell lens** or a small cylindrical-lens train is the cleanest fix; this is what the laser-sheet improvement thread plans to evaluate.

3.6.2 Smoke generator


CHAUVET DJ
HURRICANE
1200


FOG
REMOTE
MAX
MIN
FUSE
HURRICANE™
1200
CHAUVET WORLDWIDE HEADQUARTERS
SUNRISE, FLORIDA USA
CHAUVET DJ
MODEL: H1200
TYPE: VERSION 1
ITEM CODE: 09071018
FUSE: F13A,250V
POWER LINKING: N/A
ORIGIN: MADE IN P.R.C.
USA/CANADA:
Power: AC 120 V 60 Hz
10 A, 1,180 W
THIS APPLIANCE MUST BE GROUNDED
CETTE UNITÉ DOIT ÊTRE RELIÉE À LA TERRE
USE ONLY FUSE OF SAME TYPE & RATING
N'UTILISER QUE DES FUSIBLES DU TYPE ET
DE L'AMPÉRAGE INDIQUÉ
NOT FOR HOUSEHOLD USE
IMPROPRE À L'USAGE DOMESTIQUE
DRY LOCATIONS
EMPLACEMENTS SECS
DISCONNECT POWER BEFORE SERVICING
DÉBRANCHER AVANT TOUT ENTRETIEN
FOR INDOOR USE ONLY
USAGE INTÉRIEUR UNIQUEMENT

Item	Detail
Type	Glycol-based commercial smoke generator
Placement	Near the **tunnel inlet** so smoke is entrained by the airflow
Remote	Wired, placed at the **computer station** so the operator can fog without leaving the controls
Current operating settings	See the **Smoke_Generator_Back_View** photo above (heater temperature, fluid flow rate, fan setting)

The current settings are an inheritance, not a calibrated optimum. Empirical sweeps of these settings against smoke density, smoke stability, and dispersion uniformity are planned in Project improvements → smoke rack. The operator should record the dial positions in the run log so we can correlate visualization quality with settings over time.

3.6.3 Test-section lighting

A separate bulb illuminates the test section for mounting models and inspections. It is **off** during smoke-visualization runs (the goal is to see the laser sheet against a dark background). The dark-room thread (Objective 2 in the top-level README) plans a fabric or rigid enclosure around the test section to reduce ambient light leakage; until that is built, switching this lamp off and dimming the room is the standard arrangement.

3.6.4 Run-order interaction

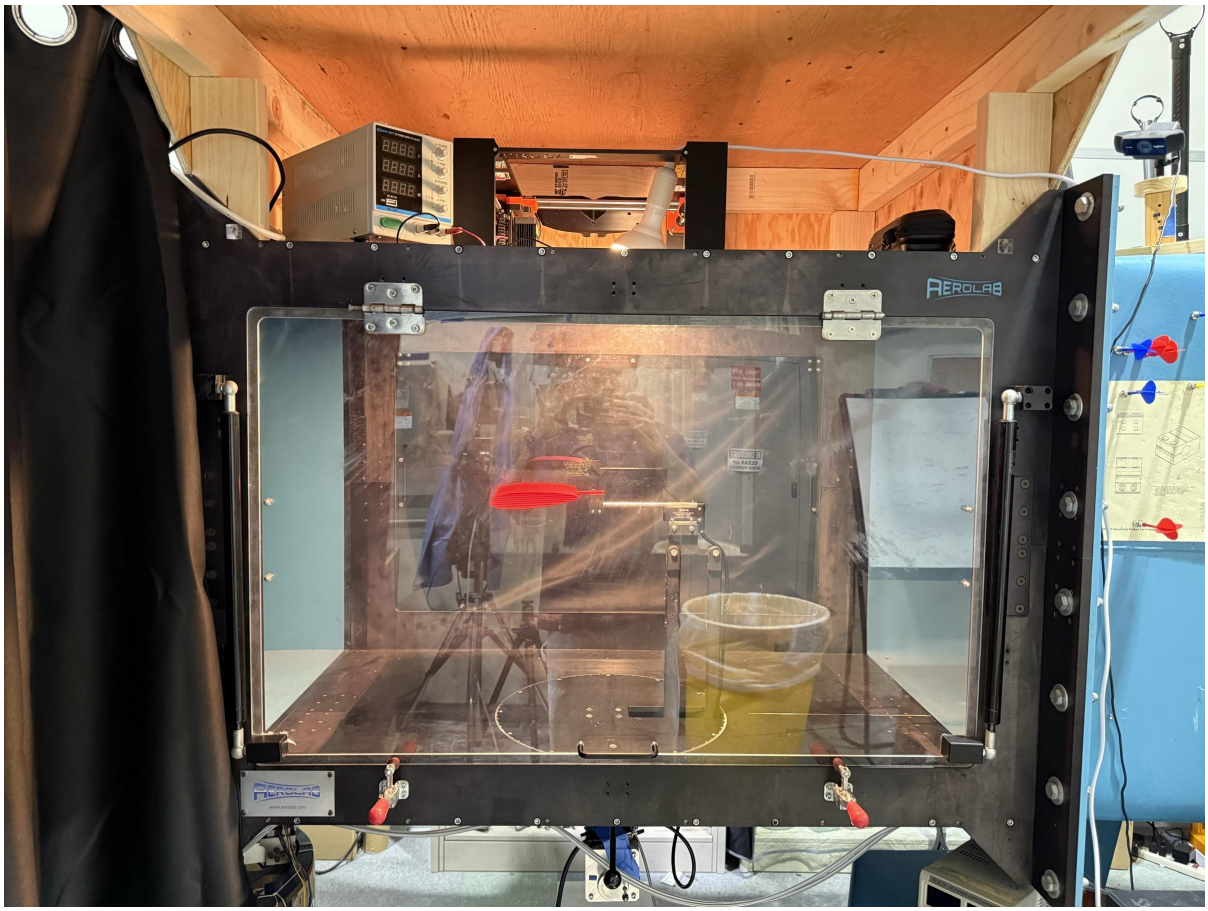
The laser, the smoke generator, and the tunnel interact strongly. The shutdown order matters: **stop the smoke first, let the test section clear, then stop the tunnel, then turn off the laser**. See Operations → Shutdown for the full sequence.

The startup order is the inverse: **laser first** (it warms up and is easy to position with the tunnel quiet), **tunnel next**, **smoke last** (once airflow is established). The dependency is that smoke without airflow pools in the room and irritates the operator.

3.7 Test section and exhaust

The test section is the working volume of the tunnel: the place where the model sits, the laser sheet cuts, and the smoke is illuminated. The exhaust path is everything downstream of it.

3.7.1 Test section



Key features:

- **Optical access** through clear side walls for the laser sheet and for camera placement.
- **A removable top** for model mounting.
- **A mounting plate / hardpoint** on the floor of the section for sting mounts and custom fixtures.
- **The pressure-ring tap** for the static-ring transducer that feeds the cDAQ NI-9203 channel and supplies the Bernoulli wind-speed calculation.

Anything placed in the test section must be **mechanically secure** before the tunnel is started. Loose hardware becomes a projectile at the fan and a maintenance event downstream. The pre-run checklist includes a visual check of model security and a tug test on any quick-mounted fixtures — see Operations → Pre-run checklist.

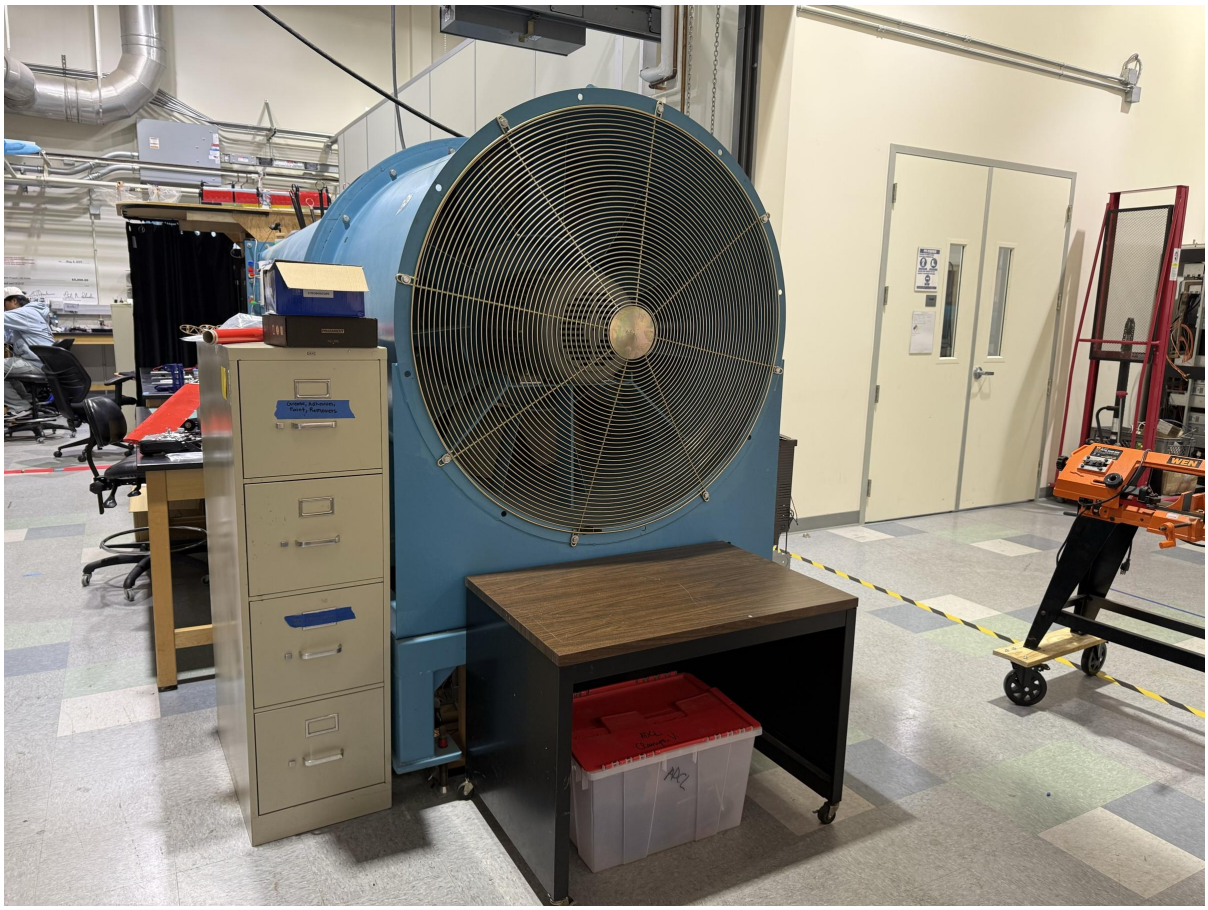
3.7.2 Main switch

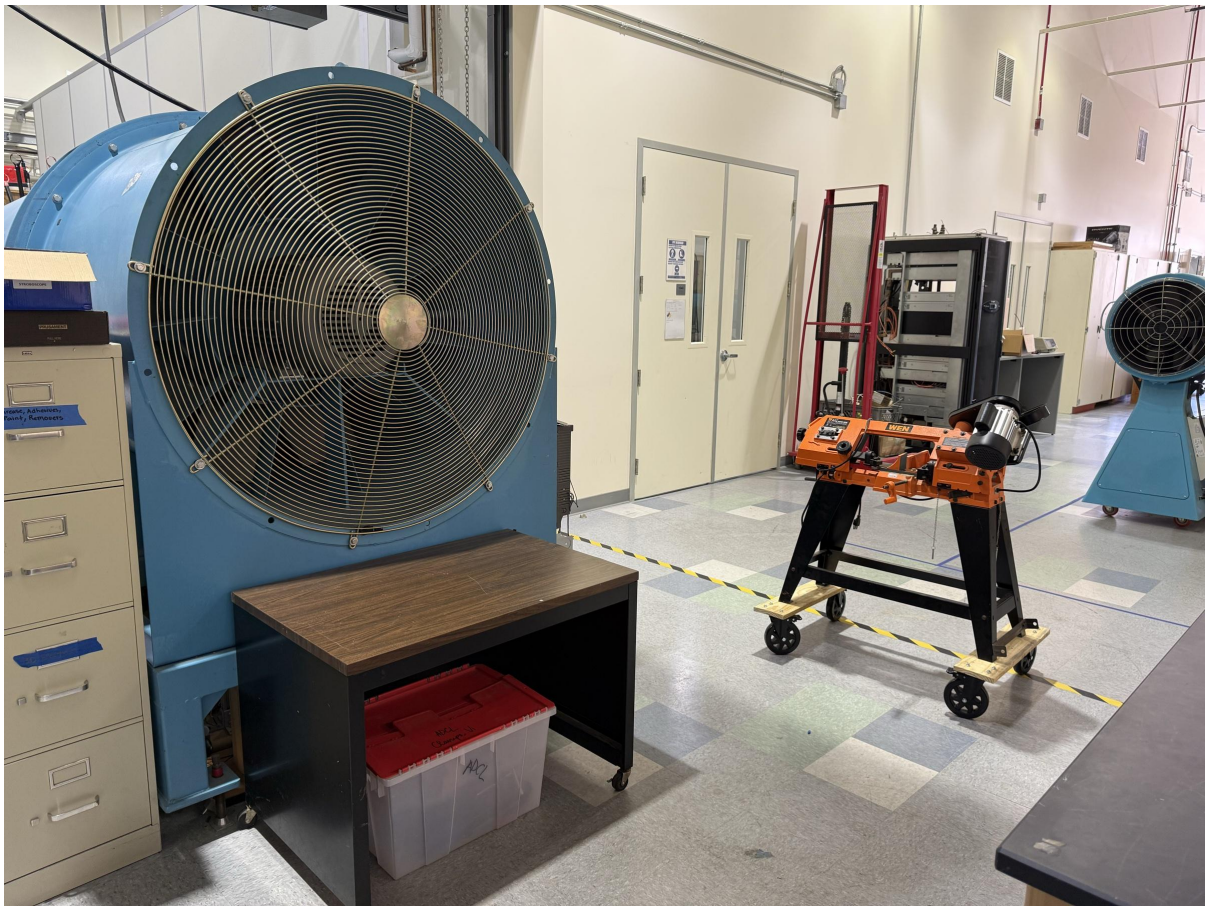
The main switch is the mains disconnect for the VFD and motor. It is mounted on the wall **within reach of the computer station** so the operator can kill power without moving away from the controls — this is by design and the layout must not be changed casually.

The switch operates on a **closed = energized** convention. The operator's view: opening the switch downward de-energizes the drive.

For lockout/tagout discipline (when the VFD enclosure must be opened), see Safety → Electrical hazards.

3.7.3 Exhaust path





The exhaust opens onto the back of the lab. The space immediately downstream of the exhaust is kept **clear of people and obstructions**. To deter people from walking into the airflow path, a parked **band saw** is positioned across the exit. The band saw is there as a **physical barrier**, not as a tool to be used; it must remain in place whenever the tunnel is energized.

The pre-run checklist confirms both: the exhaust area is clear of loose items downstream of the band saw, and the band saw is in place. If either condition is not met, the run does not start.

Why a band saw

The block exists because the lab is shared and people occasionally walk through. A rope or chain is too easy to step over; a band saw makes the path obviously impassable while still being movable when the tunnel is off and the area needs to be used for something else.

Symptoms of exhaust blockage during a run

If something restricts the exhaust during a run, the motor sees increased load and the drive current rises. The drive may trip on an overload fault (`F0001` family); if so see Troubleshooting → Faults and alarms. Stop the tunnel before clearing the obstruction. Never reach into the airflow path with the motor energized.

4. Operations

4.1 Pre-run checklist (engineering rationale)

This page is the engineering-rationale version of the checklist. The same checklist in operator form, intended for printing, is in the **SOP**. The two should be kept in sync; if you change something here, change it there too.

4.1.1 Physical state of the lab

- ☐ **Exhaust path clear.** Nothing inside the band-saw block; the band saw itself is in place across the exhaust. *Rationale: exhaust airflow becomes a wind on objects past the saw; a person past the saw becomes the wind's target.*
- ☐ **Test section interior clean.** No tools, fasteners, cables, or rags. Visual sweep with the test-section lamp on. *Rationale: anything loose accelerates to fan speed.*
- ☐ **Model is mechanically secure.** Visual inspection of all mounts and fasteners; tug test on the sting / model mounting. *Rationale: a model coming loose is the highest-consequence failure mode for the facility.*
- ☐ **Test-section access panels closed and latched.** *Rationale: open panels at speed are noisy at best and structural at worst.*
- ☐ **Camera and laser bench mechanically secure.** Tripods stable, cables out of the airflow, no items leaning against the test-section walls.

4.1.2 Electrical state

- ☐ **Main switch off** when initially walking up. *Rationale: an unsupervised energized drive is a defect; we treat the de-energized state as the resting state.*
- ☐ **Visual inspection of cabling** between the main switch, the VFD enclosure, and the motor. Any visible damage halts the run.
- ☐ **VFD enclosure closed.** No exceptions; never run with the enclosure open.
- ☐ **Lockout/tagout removed** if it was applied for prior maintenance.

4.1.3 Network and PC state

- ☐ **Lab PC powered on, no pending updates that will force a reboot mid-run.**
- ☐ **Ethernet 4 holds `192.168.50.100`.** Verify in PowerShell with `Get-NetIPAddress -InterfaceAlias 'Ethernet 4'` or whatever the adapter friendly name is on the PC.
- ☐ **ACTA-01 reachable.** `Test-NetConnection -ComputerName 192.168.50.10 -Port 502` returns `TcpTestSucceeded : True`. *Rationale: this catches lost-IP and unseated-adapter problems before they confuse the control software.*

4.1.4 Drive state

- ☐ **Keypad shows `REM`** in the upper-left corner. If it shows `LOC`, press `LOC/REM` on the keypad. *Rationale: the motor cannot be started from software in `LOC` mode.*
- ☐ **No fault codes on the keypad.** If a fault is latched, clear it from the keypad (`RESET`) and identify the cause before proceeding — see Troubleshooting → Faults and alarms.
- ☐ **Critical parameters intact:** `10.01 = 10`, `11.03 = 8`, `99.02 = 1`, `98.02 = 4`, group 51 IPs as documented. *Rationale: these define remote-controllability of the drive. If they drift, see Maintenance → Parameter recovery.*

4.1.5 Smoke generator and laser

Only required if flow visualization is part of the run.

- **Smoke generator off and unplugged from fluid refill in the last 5 minutes.**
- **Glycol reservoir has fluid.** Refilling mid-run is disruptive and a spill hazard.
- **Laser power supply at 12 V exactly.** Do not raise the voltage.
- **Laser beam terminates on a diffuse non-reflective surface.** Inspect the path; remove any new reflective object that has appeared since the last run.
- **Door placard updated** to reflect laser-in-use.

4.1.6 Software state

- **ADCL WinSoft launches and connects to the VFD.** Footer bar shows a green "Connected" indicator on the VFD side.
- **DAQ status: real, not simulated.** Footer bar shows "DAQ: cDAQ1" rather than "DAQ: simulated". If it shows simulated and you expect real data, see Troubleshooting → Modbus connection — the same diagnostic style applies to DAQ.
- **Data output folder for today exists** and the destination drive has at least a few GB of free space.
- **Operator-entered model length** is the right value for the model in the section (used for the live Reynolds-number display).

4.1.7 The first powered run of the day

Before loading the test article, do a **stop-path verification** as documented in Safety → Emergency stop. All three stop paths (ramp-stop, E-Stop, mains kill) must work. If any one fails, the day's runs do not proceed.

4.2 Start via ADCL WinSoft

ADCL WinSoft is the long-term replacement for AeroWare. It runs the VFD and the DAQ from one application. This page is the per-run startup sequence; the application's internals are in Software → ADCL WinSoft.

4.2.1 Prerequisites

- Pre-run checklist complete.
- Main switch **closed** (mains live to the VFD), drive keypad shows `REM`.
- Lab PC powered, Ethernet 4 holding `192.168.50.100`.

4.2.2 Launch

Double-click `E:\Wind_tunnel\ADCLWinSoft\ADCL WinSoft.exe`. The application takes a few seconds to start (the PyInstaller bundle is ~65 MB and unpacks to a temporary folder).

On **first launch only** the application asks for:

- an admin password (used to access the calibration / channel-map / password-change tabs); minimum 6 characters. The hash is written to `%APPDATA%\AeroLab\ADCLWinSoft\admin.hash`.
- a model characteristic length (used for the live Reynolds-number display in the side panel).

Subsequent launches go straight to the main window.

4.2.3 Connect to the drive

1. Open the **VFD** tab.
2. Verify the **Modbus connection indicator** in the footer goes green. If it does not, see Troubleshooting → Modbus connection.
3. The status panel should show the drive's quick-access registers updating at 4 Hz: CW, REF1, SW, ACT1 (speed), ACT2 (current).

4.2.4 Start the DAQ (optional but recommended)

If you want data recorded:

1. Confirm the footer says **DAQ: cDAQ1**, not "DAQ: simulated".
2. In the right-column **DAQ Gate**, press **Start DAQ**.
3. The **Signals** tab now shows live engineering-unit readings; the **Graphs** tab shows the rolling buffers.

4.2.5 Set the speed reference

1. In the right-column **Quick Tunnel** panel, drag the slider to your initial RPM setpoint (range 0–880 RPM).
2. Press **Apply Setpoint**. This writes REF1 only — it does not start the motor.
3. The drive's keypad shows the new setpoint as a frequency reference.

4.2.6 Start the motor

1. Visually confirm the test section is clear of people and tools (one more time).
2. Press **Start Tunnel**. ADCL WinSoft writes the start sequence (REF1 → CW=0x0476 → CW=0x047F) with the timing the drive requires.
3. The Status panel shows the drive transition to RUN. The **Speed RPM** readout begins to climb.
4. The motor reaches the commanded RPM in a few seconds (per the drive's `ACCEL TIME` parameter).

4.2.7 Change setpoint mid-run

Move the **Quick Tunnel** slider and press **Apply Setpoint**. ADCL WinSoft writes only REF1, never re-issues the start word; the drive ramps smoothly to the new RPM.

> **Never press Start Tunnel mid-run**
>
> Pressing **Start Tunnel** again writes `0x0476` followed by `0x047F`. Writing `0x0476` first stops the drive. This is a quirk of the ABB Drives profile; ADCL WinSoft's command worker exists in part to make this distinction explicit (`SetRpmCmd` vs `StartCmd`). For the user it means: use **Apply Setpoint** for any mid-run change, and **Start Tunnel** only from idle.

4.2.8 During the run

The right-column **Quick Tunnel** panel shows the requested vs actual RPM. The **VFD** tab shows the raw register values. The **Signals** tab shows the engineering-unit DAQ readings. The **Graphs** tab shows the rolling buffers (last 4096 samples per channel, FIFO).

Watch the footer for:

- the **Connected** indicator on the Modbus side;
- the **DAQ** indicator;
- any **fault** message from the drive (which appears as a red status banner with the fault code).

If a fault appears, Troubleshooting → Faults and alarms is the first stop.

4.2.9 Stop the motor

Two options:

- **Stop Tunnel** (preferred for normal end of run): ramp-stop per the drive's `DECEL TIME` .
- **Emergency Stop** (red, larger button): coast-stop with `CW=0x0000` and `REF1=0` .

Both write the appropriate Modbus registers. The Status panel reflects the transition.

For the rest of the shutdown sequence (smoke, laser, lights, mains), see Shutdown.

4.3 Start via PowerShell (`WindTunnelControl.ps1`)

`WindTunnelControl.ps1` is the **fallback** VFD controller. It does what ADCL WinSoft does for the motor but with none of the DAQ, no calibration, no data recording, and no graphs. Use it when:

- ADCL WinSoft is unavailable (e.g. mid-rebuild, mid-upgrade);
- you are verifying that the Modbus path itself works, independent of the application;
- you need to spin the motor without committing to a full data-recording session.

For everything else, prefer ADCL WinSoft.

The PowerShell tool's source documentation is in Software → WindTunnelControl.ps1.

4.3.1 Launching

The tool is deployed alongside the legacy AeroWare install at `E:\Wind_tunnel\AeroWare\WindTunnelControl.ps1`. Launch from the Windows side (not WSL) because the USB Ethernet adapter is host-resident:

```
powershell.exe -STA -NoProfile -ExecutionPolicy RemoteSigned `
               -File E:\Wind_tunnel\AeroWare\WindTunnelControl.ps1
```

The flags are not optional:

- `-STA` is required by Windows Forms (the GUI framework).
- `-NoProfile` skips any user profile that might inject modules or `Set-PSReadlineOption` chatter into a non-interactive run.
- `-ExecutionPolicy RemoteSigned` matches the lab PC's policy and lets the script run without prompts.

Optional parameters (defaults in parentheses):

Parameter	Default	When to override
`-VfdIp`	`192.168.50.10`	Almost never.
`-VfdPort`	`502`	Almost never.
`-UnitId`	`1`	Almost never.
`-Ref1FullScale`	`20000`	Only if a parameter rescale on the drive has changed the 100 %-reference value.

4.3.2 The window

The GUI shows:

- a **status banner** at the top with the connection target;
- a **live readout** that updates at 4 Hz: speed RPM, frequency Hz, current A, plus raw CW/REF1/SW bytes;
- an **RPM input** with an **Apply Setpoint** button;
- three action buttons: **Start Tunnel**, **Stop**, **Emergency Stop**;
- a **large status text** (`Motor: stopped` / `Motor: running at N RPM`).

4.3.3 Start sequence

1. Type the desired RPM into the input box.
2. Press **Apply Setpoint** — this writes REF1 but does not start the motor.
3. Visually confirm the test section is clear.

4. Press **Start Tunnel**. A confirmation dialog appears asking you to confirm; click **Yes** to proceed. The tool then issues REF1 → CW=0x0476 → CW=0x047F with the required delays.
5. The motor spins up.

4.3.4 Mid-run setpoint changes

Same as ADCL WinSoft: change the RPM input, press **Apply Setpoint**. The tool writes only REF1, never re-issues the start word.

4.3.5 Stop

- **Stop**: ramp-stop via `CW=0x0476`.
- **Emergency Stop**: `CW=0x0000` plus `REF1=0`, coast.

On window close, the tool does a best-effort ramp-stop followed by `CW=0x0000`. Do **not** rely on this — close the window only after the motor has been explicitly stopped and verified to be at zero RPM.

4.3.6 What this tool does not do

- It does not record data.
- It does not read the cDAQ.
- It does not apply per-channel calibration.
- It does not display wind speed or Reynolds number.
- It does not authenticate; anyone running the script can spin the motor.

For experiments that need data, use ADCL WinSoft.

4.4 Start via AeroWare (legacy, deprecated)

> **AeroWare's VFD control path is broken**
>
> As of `WT_MS_2` (2026-05-13), AeroWare attempts Modbus writes to `192.168.1.1` (a stale IP from the old Win7 install) and never reaches the drive at the actual address `192.168.50.10`. The connection fails silently and the GUI's RPM control has **no effect** on the motor.
>
> The DAQ side of AeroWare still works because it is independent of the VFD path.
>
> Do not use AeroWare to spin the motor. Use ADCL WinSoft or WindTunnelControl.ps1 instead.

This page exists because AeroWare's DAQ side is still being used as a reference while ADCL WinSoft's DAQ paths are being completed, and because the diagnosis history is useful for new operators encountering "but AeroWare is installed, can I just use it?" — the answer is **no, not for the motor**.

4.4.1 Why AeroWare's VFD path is broken

The short version, with sources:

1. AeroWare is a LabVIEW-built application; its VFD network configuration lives in `cDAQ Config.ini` files inside `E:\Wind_tunnel\AeroWare\`. There are three copies of this file, and they disagree.
2. The top-level `cDAQ Config.ini` was edited (mtime 2026-02-24) to set `VFD IP Address = "192.168.0.253"` — also wrong, just a different wrong.
3. The `Configs/` and `Executable/` copies still hold the original `192.168.1.1` from the Win7 install.
4. AeroWare reads one of these at start and uses it for the run; it does not detect failure and does not retry against alternate addresses.
5. From `WT_MS_1` and `WT_MS_2` we know the drive is reachable at `192.168.50.10` and the Modbus path works under direct PowerShell. AeroWare never gets there.

Full diagnosis trail is in `WT_MS_2/debug_session.md`.

4.4.2 What still works

The DAQ side. AeroWare's NI-DAQmx integration reads the cDAQ channels at the configured rate, applies the calibration matrices in `Configs/N_C1_inv*.csv`, and writes CSVs to `E:\Wind_tunnel\AeroWare\Data_and_others\`. This is independent of the VFD network.

If you need AeroWare's DAQ-side workflow for a specific experiment (e.g. you are validating a calibration against AeroWare's output before retiring it), it is acceptable to run **ADCL WinSoft or WindTunnelControl.ps1 for the motor** alongside **AeroWare for the DAQ**, on the same PC. They do not share state.

4.4.3 Retirement plan

AeroWare will be retired once:

- ADCL WinSoft's PGB sting calibration produces the same engineering-unit outputs as AeroWare against a held-out validation file;
- ADCL WinSoft's CSV format matches the one consumed by downstream post-processing scripts;
- ADCL WinSoft has been exercised on at least three full experimental sessions without operator-visible defects.

After retirement, AeroWare's install at `E:\Wind_tunnel\AeroWare\` stays in place for historical reference but is renamed or marked deprecated. Calibration matrices at `Configs/N_C1_inv*.csv` are read by ADCL WinSoft from the same location, so the directory is not removed.

4.4.4 Reference

- Project notes → AeroWare install location
- WT_MS_2 debug session

4.5 Shutdown

Shutdown order matters. Stopping things in the wrong sequence is rarely dangerous but is annoyingly easy to do — smoke without airflow pools in the room, killing the mains while the drive is running causes a coast-down that confuses the software, and so on. The recommended sequence below is the operator-tested default.

4.5.1 Normal shutdown (end of session)

1. **Stop the smoke generator.** Press OFF on the wired remote at the computer station. Wait until the visible plume in the test section clears (typically 10–20 s of continued airflow does it).
2. **Ramp-stop the motor.** Press **Stop Tunnel** in ADCL WinSoft (or `WindTunnelControl.ps1`). The motor decelerates per the drive's `DECEL TIME` parameter.
3. **Verify motor at zero RPM.** Look at the speed readout and at the drive keypad. Both should read 0.
4. **Stop the DAQ recording.** In ADCL WinSoft's right column, press **Stop DAQ**. The session's CSV is finalized to `E:\Wind_tunnel\ADCLWinSoft\Data\YYYY-MM-DD\…`.
5. **Close the control software.** ADCL WinSoft — close the window cleanly; it will write back its settings. `WindTunnelControl.ps1` — close the window (the tool does a best-effort coast-stop on close, but the motor should already be stopped).
6. **Turn off the laser.** Open the DC power supply's output enable, then power the supply down.
7. **Turn off the test-section light** if it was used.
8. **Open the main switch.** This de-energizes the VFD and motor. The drive keypad goes dark.
9. **Close out the lab.** Logbook entry, dispose of any smoke-generator fluid spills, restore the band saw and any other moved obstructions.

> The order is: **everything inside the airflow path first** (so smoke clears), **then the airflow itself**, **then the things outside the airflow path** (laser, lights), **then the mains disconnect**.

4.5.2 Pause shutdown (between runs in the same session)

If you are between runs and want to swap models or adjust the laser:

1. **Smoke generator OFF.**
2. **Motor ramp-stop.**
3. Leave the DAQ running or pause it — your choice; pausing is cleaner if you'll be touching transducer cables.
4. Leave the laser **off** while reaching into the test section.
5. Leave the main switch **closed**. The drive should keep its Modbus connection live.
6. When ready to resume, follow the startup sequence from "Set the speed reference".

4.5.3 Emergency shutdown

See Safety → Emergency stop for the full procedure. The short version: **stop the motor first** (software E-Stop, or mains kill if software is unresponsive); **then call for help**; **then secure the area**.

4.5.4 After-action logging

End every session with:

- A note in the day's log: who operated, what was tested, RPM range covered, any oddities.
- If anything unexpected happened — fault codes, software hiccups, hardware noise — open a new `WT_MS_<n>/` milestone with `debug_session.md` and attach the relevant files (CSVs, photos, parameter dumps). See `docs/repository_layout.md` for the layout.

4.6 Data acquisition workflow

This page covers what ADCL WinSoft does when you press **Start DAQ** during a run, where the data ends up, and how to feed it into downstream post-processing.

4.6.1 What happens when you press Start DAQ

1. The application starts a `DaqWorker` background thread (`code/adcl_winsoft/src/adcl_winsoft/daq/worker.py`).
2. The worker opens a continuous NI-DAQmx task across all configured channels at the configured sample rate (default 5 Hz).
3. Each tick, the worker reads one sample per channel, applies the per-channel slope/offset, applies the PGB matrix where applicable, derives the secondary signals (velocity, Reynolds), and writes:

- one row to the open CSV;
- one snapshot into `AppState.last_daq` for the UI;
- one entry into each channel's rolling buffer (capped at 4096 samples, FIFO).

4. The UI's 4 Hz refresh timer picks up `AppState.last_daq` and updates the visible panels.

There are two paths from the worker into the UI — Qt signals and direct shared-state writes. The direct path is the backup that keeps the application responsive if signal delivery stalls. This is the "belt and suspenders" pattern in `app.py` .

4.6.2 CSV layout

```
E:\Wind_tunnel\ADCLWinSoft\Data\YYYY-MM-DD\DDMMYY HHMMSS.fff<tag>.csv
```

Columns (typical):

Column	Unit	Source
`t`	seconds since session start	DAQ worker timestamp
`i_pressure_static`	mA	NI-9203, raw
`i_pressure_model`	mA	NI-9203, raw
`v_pitch_angle`	V	NI-9215, raw
`mv_pgb_normal`	mV/V	NI-9237, raw
`mv_pgb_axial`	mV/V	NI-9237, raw
`mv_pgb_moment`	mV/V	NI-9237, raw
`p_static_pa`	Pa	calibrated
`p_model_pa`	Pa	calibrated
`pitch_deg`	degrees	calibrated
`pgb_n_N` , `pgb_a_N` , `pgb_m_Nm`	N, N·m	matrix-calibrated
`velocity_mps`	m/s	Bernoulli on static-ring ΔP
`re`	dimensionless	from velocity, model length, viscosity

Exact column set depends on the channel map (`daq/channel_map.py`).

The **filename format** `DDMMYY HHMMSS.fff<tag>.csv` is inherited from AeroWare so existing post-processing scripts work unchanged.

4.6.3 Conventions for tagging runs

The optional `<tag>` in the filename is taken from the **Acquisition** panel's tag field. Use it to mark the run in a way the filename system can index — examples: `_baseline`, `_naca0012_alpha_05`, `_smoke_only`. Avoid spaces; use underscores. Avoid characters that the Windows filesystem rejects (`:` `*` `?` `<` `>` `|` `/` `\`).

4.6.4 How much data per session

At 5 Hz with ~14 columns of doubles plus a timestamp, a CSV row is ~150 bytes. A one-hour session is roughly 2.6 MB. A full day of measurements is well under a gigabyte. Free space on `E:\` is not a routine concern but check before starting if you are using a fresh drive.

4.6.5 Downstream post-processing

Post-processing is **not** in this repository — it lives with the individual experiments. The contract from ADCL WinSoft to downstream is the CSV format above and the matrix calibration on disk at `E:\Wind_tunnel\AeroWare\Configs\N_C1_inv*.csv`. If a downstream script reads the CSV and produces, say, force-vs-AoA plots, it can be invoked independently from MATLAB, Python, or a Jupyter notebook.

4.6.6 Validating against AeroWare

While ADCL WinSoft is still being validated, the workflow for a fresh calibration session is:

1. Run AeroWare's DAQ pipeline (motor stopped) for a known reference condition.
2. Save the AeroWare CSV.
3. Run ADCL WinSoft's DAQ pipeline against the same physical state.
4. Compare the engineering-unit columns. They should match to within transducer noise.

   If they do not, the discrepancy is either in the channel map, the per-channel calibration, or the matrix application. Walk through `daq/channel_map.py`, `daq/calibration.py`, and the matrix file in that order.

5. Software

5.1 Software architecture

The control software comes in three layers in this lab:

1. **AeroWare** — legacy LabVIEW application. VFD path broken (see Operations → Start via AeroWare); DAQ path still works. Being retired.
2. `WindTunnelControl.ps1` — short-term PowerShell + Windows Forms GUI in `code/wind_tunnel_control/`. Does VFD control only, no DAQ. Used as a fallback and for direct hardware verification.
3. **ADCL WinSoft** — long-term Python + PySide6 application in `code/adcl_winsoft/`. Does VFD control **and** DAQ in one process. Packaged as a single-file `.exe`. This is the supported tool going forward.

5.1.1 When to use which tool

Tool	Use when
ADCL WinSoft	The default. Any run that needs data, calibration, or a UI with feedback.
`WindTunnelControl.ps1`	ADCL WinSoft is unavailable; you need to test the Modbus path itself; you need to spin the motor without committing to a recording session.
AeroWare	Only its DAQ side, only while validating ADCL WinSoft against it. Never its VFD side.

The two new tools share the **same Modbus protocol behaviour** (start sequence, calibration constants, drive parameters required). They differ in language, threading model, polish, and feature scope. Both target the same RETA-01 at `192.168.50.10:502`.

5.1.2 Shared concepts

Modbus profile

Both tools use the **ABB Drives** profile with the quick-access registers:

Address	Name	Notes
0	Control Word	`0x0476` (prep / ramp-stop), `0x047F` (run), `0x0000` (off / E-Stop)
1	REF1	Speed reference, scaled `REF1 = drive_RPM × 22.42`
3	Status Word	RDY, RUN, REM, FAULT bits
4	ACT1	Speed actual
5	ACT2	Current actual

Both tools read parameter values through the formula `address = group × 100 + index − 1`.

Start sequence

Both tools issue the same sequence from idle:

```
write REF1 = setpoint × 22.42
wait 0.2 s
write CW = 0x0476
wait 0.5 s
write CW = 0x047F
```

For mid-run changes both tools write **only REF1**. This is the most important behavioural rule in either codebase.

Calibration constants

Both tools share the empirical constants from `WT_MS_2`:

- `REF1 = drive_RPM × 22.42`
- `wind_speed_MPH ≈ 0.0822 × drive_RPM − 13.14` (for `drive_RPM > 300`, where the static-ring transducer escapes noise)

See Reference → Calibration constants.

5.1.3 Differences worth knowing

Aspect	ADCL WinSoft	`WindTunnelControl.ps1`
Language	Python 3.12 + PySide6 (Qt6)	PowerShell 5.1 + Windows Forms
Modbus socket	**One persistent** socket per `VfdClient`, lock-guarded	One **fresh socket per call**
Polling rate	250 ms (4 Hz) reads; commands enqueued async	4 Hz polling loop
DAQ	Yes — NI-DAQmx via `nidaqmx`, simulated fallback	No
Calibration	Yes — per-channel + matrix	No
Data recording	Yes — CSV in AeroWare-compatible format	No
Admin gate	Yes — Argon2id password, hidden tabs	No
Packaging	Single-file `.exe` via PyInstaller	Run script directly
Source	`code/adcl_winsoft/`	`code/wind_tunnel_control/`

Why persistent vs fresh sockets

The RETA-01 is happy with one connection at a time. Opening a fresh TCP socket per Modbus call works — that is what `WindTunnelControl.ps1` does, at 4 Hz that is ~8 sockets per second and the adapter does not complain — but at higher rates the TCP handshake overhead becomes a problem. ADCL WinSoft polls at the same 4 Hz but its DAQ side adds load, and its command worker can fire writes in bursts; the persistent-socket design is what makes that stable.

This is the architectural lesson out of `WT_MS_2`'s AeroWare diagnosis: AeroWare appears to have opened a fresh socket per Modbus call, which combined with its incorrect target IP produced a ~9 s TCP retransmit storm before failure. The persistent-socket plus correct IP design avoids both classes of bug.

5.1.4 Where to read next

- ADCL WinSoft — modules, threading model, footguns.
- WindTunnelControl.ps1 — Modbus client, start sequence, GUI layout.
- Building & deploying — how to ship updates to either tool.
- Testing — pytest smoke tests in `code/adcl_winsoft/tests/`.
- Extending the code — how to add a tab, a new register, or refresh calibration.

5.2 ADCL WinSoft (`code/adcl_winsoft/`)

The long-term AeroWare replacement: a PySide6 desktop application that controls the VFD and reads the cDAQ in one process. Built primarily on 2026-05-13; phases F.1–F.6 complete.

Companion notes that are mirrored from the assistant's development memory:

- Project notes → ADCL WinSoft overview
- Project notes → pymodbus quirks
- Project notes → PyInstaller quirks

5.2.1 Tech stack

Component	Purpose
Python 3.12	Language runtime. Pinned for the bundled `.exe` build.
PySide6 (Qt 6)	GUI framework.
`nidaqmx`	NI-DAQmx Python bindings for the cDAQ.
`pymodbus` 3.13	Modbus/TCP client for the RETA-01.
`pyqtgraph`	Live plotting.
`argon2-cffi`	Admin password hash.
`numpy`	Calibration math.
PyInstaller	Packaging — produces a single-file `~65 MB` `.exe` , `console=False` .

5.2.2 Directory layout (in this repo)

```
code/adcl_winsoft/
├── README.md
├── DESIGN.md
├── CHANGELOG.md
├── pyproject.toml          ← dev install (pip install -e .[dev])
├── adcl_winsoft.spec       ← PyInstaller single-file spec
├── deploy/
│   ├── install_dev.cmd     ← xcopy source into the Windows venv
│   ├── run_dev.cmd
│   ├── build_exe.cmd
│   ├── build_exe_nopause.cmd
│   ├── init_admin.cmd
│   └── build_exe.ps1
├── release/
│   ├── ADCL WinSoft.exe    ← shipped binary
│   └── README.md
├── src/adcl_winsoft/
│   ├── __init__.py
│   ├── __main__.py         ← entry point; first-launch dialogs; admin/model prompts
│   ├── app.py              ← AppState central hub; owns workers; signal wiring
│   ├── units.py            ← SI internal; user-facing display conversions
│   ├── auth/
│   │   └── admin.py        ← Argon2id hash + verify with incremental backoff
│   ├── config/
│   │   └── settings.py     ← JSON settings + AeroWare-path migration
│   ├── daq/
│   │   ├── cdaq_reader.py  ← NI-DAQmx wrapper; simulated fallback
│   │   ├── worker.py       ← QThread polling at 5 Hz
│   │   ├── calibration.py  ← Bernoulli velocity, PGB matrix, Sutherland viscosity
│   │   └── channel_map.py  ← default channel definitions
│   ├── data/
│   │   ├── format.py       ← AeroWare-compatible CSV layout
│   │   └── recorder.py     ← background CSV writer
│   ├── vfd/
│   │   ├── modbus_client.py    ← persistent TCP socket, lock-guarded
│   │   ├── controller.py       ← start/stop/E-Stop sequences, calibration constant
│   │   ├── command_worker.py   ← async write queue, precondition gating
│   │   ├── worker.py           ← async poller
│   │   └── __init__.py
│   ├── resources/fonts/        ← bundled Roboto + RobotoMono
```

```
│   └── ui/
│       ├── main_window.py      ← QSplitter [sidebar | tabs | right column]
│       ├── theme.py            ← 6 palettes via parameterized QSS
│       ├── admin/
│       │   ├── login_dialog.py
│       │   ├── setup_dialog.py
│       │   └── model_setup_dialog.py
│       ├── tabs/
│       │   ├── signals.py
│       │   ├── graphs.py
│       │   ├── vfd.py
│       │   ├── settings.py
│       │   ├── pgb_sting.py
│       │   └── placeholder.py
│       └── widgets/
│           ├── sidebar.py
│           ├── footer_bar.py
│           ├── quick_tunnel.py
│           ├── daq_gate.py
│           ├── control_panel.py
│           ├── collapsible.py
│           └── signal_list.py
├── tests/
│   └── test_smoke.py
└── tools/
    └── init_admin.py
```

5.2.3 Architecture

`AppState` central hub (`app.py`)

`AppState` is the Qt object that owns the three worker threads, the rolling buffers, and the last-known snapshots from each hardware layer. UI tabs read from `AppState` via two paths:

1. **Qt signals** — workers emit `daq_snapshot`, `vfd_snapshot`, `unit_system_changed`, etc.; tabs connect their slots and update.
2. **Direct shared state** — workers also write into `AppState.last_daq` / `AppState.last_vfd` / `AppState.buffers` directly. A 4 Hz `QTimer` in the UI reads from those fields. This is GIL-safe — `deque.append` and pointer assignment are atomic in CPython — and it keeps the UI responsive even if signal delivery is delayed.

Either path on its own would be enough most of the time. Having both is the "belt and suspenders" pattern; it survived the F.1–F.6 development without producing a stuck-UI bug.

Workers

Worker	Thread	Rate	What it does
`DaqWorker` (`daq/worker.py`)	QThread	configurable, default 5 Hz	Reads one sample per channel, applies calibration, writes a `DaqSnapshot` into AppState and emits a signal. Also writes the row to the open CSV.
`VfdPoller` (`vfd/worker.py`)	QThread	4 Hz (250 ms)	Reads CW, REF1, SW, ACT1, ACT2 from the drive; emits a `VfdSnapshot`.
`VfdCommandWorker` (`vfd/command_worker.py`)	QThread	event-driven	Receives `StartCmd`, `StopCmd`, `EStopCmd`, `SetRpmCmd`. Checks preconditions (REM mode, no FAULT). Executes the multi-step write sequence with the required delays. Emits step-by-step progress.

The reason the **command worker** is its own thread is that the start sequence is `write REF1 → sleep 0.2 s → write CW=0x0476 → sleep 0.5 s → write CW=0x047F`. Done synchronously on the GUI thread, that is 0.7 s of stalled UI per Start press. Done on a worker, the GUI stays responsive and the operator sees per-step feedback.

VFD client (`vfd/modbus_client.py`)

`VfdClient` wraps a `pymodbus.client.ModbusTcpClient` with:

- **one persistent connection** maintained for the life of the client;
- a `threading.Lock` around every wire-format write/read;
- a thin layer that translates parameter group/index pairs into Modbus addresses.

The persistent connection is the key architectural choice — see Architecture → Why persistent vs fresh sockets.

VFD controller (`vfd/controller.py`)

Owns the calibration constant (`REF1_PER_RPM = 22.42`) and the start/stop sequences. Three methods: `start(rpm)`, `stop()`, `estop()`, and a `set_rpm(rpm)` for mid-run setpoint changes that writes **only** REF1.

DAQ reader (`daq/cdaq_reader.py`)

Wraps NI-DAQmx and degrades gracefully:

- If `nidaqmx` is not importable (e.g. this Mac), the reader returns synthetic data.
- If `ADCL_WINSOFT_SIMULATE=1` is set, simulation mode is forced even with real hardware available.
- The NI-9237 requires a continuous sample clock; the reader opens a continuous task and reads one sample per polling tick.

UI shell (`ui/main_window.py`)

`QSplitter` with three columns:

1. **Sidebar** — list of tabs and theme selector.
2. **Tabs** — Signals · Graphs · VFD · Settings · PGB Sting · Traverse (placeholder) · CTA (placeholder) · XY Balance (placeholder).
3. **Right column** — `QuickTunnel` (slider + Apply/Start/Stop/E-Stop) on top, `DaqGate` (Start/Stop DAQ) middle, `ControlPanel` (collapsible Acquisition + File panels) bottom.

Below the splitter is a `FooterBar` showing connection states, simulation mode, and active theme.

Theme system (`ui/theme.py`)

Six named palettes (Dark, Light, Monokai, Topaz, Obsidian, Nord) driven from one parameterized QSS template. Only color tokens change between themes. Bundled Roboto fonts load at startup so the look is identical to the bundled `.exe` regardless of system fonts.

5.2.4 Footguns baked into the code

These are the lessons from the project notes, summarized so they appear here for new contributors:

- **`pymodbus` 3.13 renamed `slave=` to `device_id=`.** The old kwarg silently throws `TypeError`. Treat any "type error from pymodbus" as a version-skew tell. See `vfd/modbus_client.py` and the pymodbus quirks note.
- **PyInstaller does not bundle package metadata by default.** `nidaqmx`'s dependency `nitypes` calls `importlib.metadata.version("nitypes")` and dies on a fresh bundle. The spec at `adcl_winsoft.spec` uses `copy_metadata()` for `nidaqmx`, `nitypes`, `hightime`, `deprecation`, `pymodbus`, `argon2_cffi`, `PySide6`, and `pyqtgraph`. Adding a new dependency means adding it to that list. See PyInstaller quirks note.
- **`__main__.py` cannot use relative imports** when run by PyInstaller (no `__package__`). Absolute imports only in the entry point. Submodules may use relative imports freely.
- **Drive start sequence is `REF1 → CW=0x0476 (stops!) → CW=0x047F (starts)`.** Re-issuing `0x0476` mid-run will stop the drive. For setpoint changes, write REF1 only — `SetRpmCmd` does this; `StartCmd` is for the initial start only.
- **VFD UI buttons must respond to CW bit 3 (RUN).** When the drive is running, the Start button disables, Stop becomes the primary visual action, Apply stays enabled so the operator can change RPM live. This is wired in `ui/widgets/quick_tunnel.py` and `ui/tabs/vfd.py`.
- **Worker → UI signal delivery alone is not enough.** Workers must also write into `AppState.last_*` so the UI's 4 Hz refresh timer can pull state directly. This is the belt-and-suspenders pattern.

5.2.5 Where data is stored at runtime

Item	Path
Application binary	`E:\Wind_tunnel\ADCLWinSoft\ADCL WinSoft.exe`
Settings JSON	`%APPDATA%\AeroLab\ADCLWinSoft\settings.json`
Admin hash	`%APPDATA%\AeroLab\ADCLWinSoft\admin.hash`
Calibration matrices (read)	`E:\Wind_tunnel\AeroWare\Configs\N_C1_inv*.csv`
Experiment CSVs (written)	`E:\Wind_tunnel\ADCLWinSoft\Data\YYYY-MM-DD\`
Dev venv	`C:\Users\kshit\.venvs\adcl_winsoft\`
Build staging	`C:\Users\kshit\.adcl_winsoft_build\`

5.2.6 Phase status

F.1–F.6 complete as of 2026-05-13. Open work:

- **F.5 admin editors** — calibration matrix UI, channel-map UI, password-change UI in admin mode.
- **Closed-loop wind-speed control** — read static-ring ΔP via cDAQ, run a PID against target velocity, write REF1.
- **CTA, XY Balance, Traverse tabs** — currently placeholders.

See `code/adcl_winsoft/CHANGELOG.md` for the per-phase notes.

5.3 `WindTunnelControl.ps1` (`code/wind_tunnel_control/`)

A PowerShell + Windows Forms GUI that controls the VFD directly over Modbus/TCP. ~19 KB of script, one file, no dependencies beyond what ships with Windows 11.

This is the **short-term replacement** for AeroWare's broken VFD path. It came out of `WT_MS_2` (2026-05-13) once AeroWare's Modbus path was confirmed unrepairable in place.

For the long-term replacement see ADCL WinSoft; for usage see Operations → Start via PowerShell.

5.3.1 Directory layout

```
code/wind_tunnel_control/
├── README.md
├── WindTunnelControl.ps1     ← the GUI itself
├── deploy.ps1                ← copy WindTunnelControl.ps1 + .cmd launcher to E:\Wind_tunnel\AeroWare\
├── ref1_sweep.ps1            ← calibration: write REF1 values, log resulting drive frequency
└── rpm_velocity_sweep.ps1    ← calibration: sweep RPM, log static-ring-derived wind speed
```

5.3.2 Modbus client (in-script)

Hand-rolled because Windows PowerShell does not ship with a Modbus library. ~50 lines.

- **One TCP socket opened per call** — `New-Object System.Net.Sockets.TcpClient`, write the Modbus frame, read the reply, dispose. Simple, no connection pooling.
- Supports FC `0x03` (read holding registers) and FC `0x06` (write single register).
- Parses the MBAP header; checks exception codes; throws on `0x83` / `0x86`.
- Helper `Read-Param -Group N -Index M` computes the address as `N × 100 + M − 1`.

The fresh-socket-per-call pattern is acceptable at the tool's 4 Hz polling rate — that is ~8 sockets/second to the RETA-01 and the adapter does not complain. It would not be acceptable at the higher rates ADCL WinSoft can run.

5.3.3 Startup probe

On launch the tool reads parameter `99.08 MOTOR NOM SPEED` to know the motor nameplate RPM. If the read fails, the tool falls back to a hardcoded **1800 RPM** and surfaces the failure on the status banner. The fallback exists because we never want the tool to refuse to launch on a single Modbus read failure — but it is also a footgun: a wrong nameplate value means the displayed speed is wrong. The status banner is the user's signal that the probe failed.

5.3.4 GUI

Windows Forms, not WPF. Single window with:

- **Status banner** at the top with `VFD: 192.168.50.10:502 (Unit ID 1)`.
- **Live readout** polled at 4 Hz:
- Speed RPM (decoded from ACT1)
- Frequency Hz (decoded from drive parameters)
- Current A (decoded from ACT2)
- Raw CW / REF1 / SW hex values
- Decoded status word: RDY_ON · RDY_RUN · RDY_REF · TRIPPED · OFF2 · OFF3 · SWC_INH · ALARM · AT_SP · REMOTE · ABOVE
- **RPM input** with **Apply Setpoint** button.
- **Raw REF1 escape hatch** for calibration overrides.
- **Two radio buttons**: RPM control (enabled), Wind Speed control (disabled, phase 2).
- **Three action buttons**: Start Tunnel (with a confirmation dialog), Stop, Emergency Stop.
- **Big status text**: `Motor: stopped` or `Motor: running at N RPM`.

5.3.5 Start sequence

Identical to ADCL WinSoft (the calibration constant `22.42` is the same):

```
write REF1 = setpoint × 22.42
sleep 0.5
write CW = 0x0476
sleep 0.5
write CW = 0x047F
```

The Start button shows a confirmation dialog first, asking the operator to confirm the test section is clear, before issuing any writes.

5.3.6 Stop and E-Stop

- **Stop** writes `CW = 0x0476`. The drive ramps per parameter `22.02 DECEL TIME`.
- **Emergency Stop** writes `CW = 0x0000` plus `REF1 = 0`. The drive coasts (no ramp).
- **Window close** issues a best-effort `Stop` followed by `CW = 0x0000`. Do not rely on this; the operator must stop the motor explicitly and verify zero RPM before closing.

5.3.7 Calibration scripts

`ref1_sweep.ps1` and `rpm_velocity_sweep.ps1` are the two scripts that produced the calibration constants used by both control tools:

- `ref1_sweep.ps1` walks REF1 from 0 to a target maximum, polling the drive's frequency feedback, writing pairs to a CSV. This produced the `REF1 = drive_RPM × 22.42` fit.
- `rpm_velocity_sweep.ps1` walks RPM through a coarse set of setpoints, recording the static-ring-derived wind speed. This produced the `MPH ≈ 0.0822 × drive_RPM − 13.14` fit valid above ~300 RPM.

The CSV outputs from the 2026-05-13 sweeps live in `WT_MS_2/captures/`:

- `ref1_sweep_20260513_143847.csv`
- `rpm_velocity_20260513_144742.csv`

When the calibration needs to be refreshed (after hardware changes, after a motor swap, after a long idle period), re-run these scripts and update the constants in both:

- `code/adcl_winsoft/src/adcl_winsoft/vfd/controller.py` (`REF1_PER_RPM`)
- `code/wind_tunnel_control/WindTunnelControl.ps1` (the equivalent constant)

See Maintenance → Calibration refresh for the procedure.

5.3.8 Deployment

`deploy.ps1` copies `WindTunnelControl.ps1` and a `.cmd` launcher into `E:\Wind_tunnel\AeroWare\`. The destination directory is the legacy AeroWare install path; placing the new tool there gives the operator a single folder to look in.

5.3.9 What this tool intentionally lacks

- DAQ. The cDAQ is not touched.
- Calibration application. Raw drive register values are displayed; no engineering-unit conversions.
- Data recording. There is no CSV output.
- Authentication. Anyone who can run the script can spin the motor.
- Wind-speed setpoint. The radio button is in the UI but the write logic is not implemented — phase 2 needs closed-loop PID against cDAQ pressure feedback, which means cDAQ access, which is what ADCL WinSoft has and this tool does not.

5.4 Building and deploying

Both control tools live in this repository. Their build and deploy paths are different and described separately below.

5.4.1 ADCL WinSoft

The Python application is packaged as a single-file Windows `.exe` via PyInstaller. Build is a Windows operation — the `.exe` is for Windows, the NI-DAQmx bindings only exist on Windows, and the build tooling has been validated only on the lab PC.

Dev install (after `git pull`)

From a Windows command prompt or PowerShell:

```
cd code\adcl_winsoft
deploy\install_dev.cmd
```

This `xcopy`s the package source into `C:\Users\kshit\.venvs\adcl_winsoft\Lib\site-packages\adcl_winsoft\`. The reason we do not `pip install -e .` over the WSL UNC share is that the share is slow and the editable install is unreliable across the boundary. See the PyInstaller quirks note.

Dev run

```
deploy\run_dev.cmd
```

This activates the venv and runs `python -m adcl_winsoft`. Use this for any change-iterate-test loop; do not rebuild the `.exe` for every edit.

Building the `.exe`

```
deploy\build_exe.cmd
```

What this does:

1. Stages source from the WSL repo to `C:\Users\kshit\.adcl_winsoft_build\` via `xcopy` (much faster than reading via the WSL share).
2. Activates the venv.
3. Runs `pyinstaller --noconfirm --clean adcl_winsoft.spec`.
4. Copies the resulting `dist\ADCL WinSoft.exe` into `code\adcl_winsoft\release\` and to `E:\Wind_tunnel\ADCLWinSoft\` on the lab PC.

Total build time on the lab PC is about **2 minutes**. The resulting binary is ~65 MB, single-file, `console=False`.

What the spec file does

`adcl_winsoft.spec` is a hand-written PyInstaller spec, not an auto-generated one. The notable parts:

- `datas=` carries the bundled Roboto fonts from `src/adcl_winsoft/resources/fonts/`.
- `copy_metadata()` calls bundle package metadata for `nidaqmx`, `nitypes`, `hightime`, `deprecation`, `pymodbus`, `argon2_cffi`, `PySide6`, `pyqtgraph`. Without these, the `.exe` crashes at first import on `importlib.metadata.version(...)` calls. See PyInstaller quirks note.
- `hiddenimports=` lists `pymodbus` submodules that PyInstaller's static analyzer does not catch.
- `excludes=` removes large modules that are imported transitively but not used (tkinter, scipy submodules, etc.) to keep the binary small.

If you add a new third-party dependency to `pyproject.toml`, ask yourself:

1. Does it call `importlib.metadata.version(...)` on itself or anything it imports? If yes, add it to `copy_metadata()`.
2. Does it have submodules that are imported via string lookup or `importlib`? If yes, add them to `hiddenimports`.
3. Test the bundled `.exe`, not just the dev run. The dev run will not catch metadata bugs.

Admin password

`tools/init_admin.py` was the original CLI for setting the admin password. It is now legacy — first-run dialogs in `__main__.py` handle it interactively. The CLI is kept around because it is useful for unattended bring-up.

5.4.2 `WindTunnelControl.ps1`

The PowerShell tool needs no build step. Deployment is a copy.

Deploy after editing

From the WSL side (in the repository working copy):

```
cd code/wind_tunnel_control
powershell.exe -ExecutionPolicy RemoteSigned -File ./deploy.ps1
```

`deploy.ps1` copies `WindTunnelControl.ps1` and a `.cmd` launcher to `E:\Wind_tunnel\AeroWare\`. From there, the operator runs the launcher.

Why deploy alongside AeroWare

`E:\Wind_tunnel\AeroWare\` is the directory the existing operators look in. Putting the new tool there avoids a "where is the tunnel software?" moment for anyone returning to the lab after a few months. It is not a technical requirement; the script will run from anywhere.

5.4.3 Versioning

Neither tool has a formal version-tagging discipline yet. The source in this repository is the truth; the deployed copies are derivatives. To know what version is on the lab PC, look at the file mtimes and compare to the git log of the source.

If we end up shipping ADCL WinSoft to anyone outside this lab, a semver tag inside `pyproject.toml` plus a `CHANGELOG.md` entry per release should be added.

5.4.4 Where the release binary lives

`code/adcl_winsoft/release/ADCL WinSoft.exe` is the most recent built binary, committed to the repository so a fresh clone of the repo on the lab PC can be deployed without rebuilding. The release directory has its own `README.md` describing what is there.

5.5 Testing

The control code has two test layers: an automated **pytest** smoke suite that runs without hardware, and a **manual hardware bring-up** routine that runs against the real drive and DAQ.

5.5.1 Automated: `code/adcl_winsoft/tests/`

Five non-Qt smoke tests in `tests/test_smoke.py`. They exercise the parts of the code that do not need a display server, a Modbus slave, or NI-DAQmx.

Run

From a Windows venv or a Linux/macOS one (as long as you have not asked it to import Qt):

```
cd code/adcl_winsoft
pytest -q
```

All five tests should pass.

What they cover

Test	What it checks
Modbus address calculation	`group × 100 + index − 1` produces the expected register addresses for known parameters.
Calibration round-trip	`REF1 = drive_RPM × 22.42` and the inverse agree to within float precision.
Settings persistence	A round-trip through `config/settings.py` of a non-trivial settings dict reproduces the original.
Admin hash verify	Setting and verifying an Argon2id hash works for known good and known bad inputs.
Calibration matrix application	Applying a known PGB matrix to a known input vector produces the expected output.

When to add a test

Any time you add a calibration constant, a settings key, a Modbus address, or a hash routine. The test should be cheap (sub-second), hardware-free, and assert one specific behaviour. If a bug ever reaches the lab, the easiest mitigation is to land a regression test alongside the fix.

What is *not* covered

- The Qt UI. We do not run Qt under pytest in this project — too much setup, too little signal.
- The actual NI-DAQmx integration. The reader's simulated fallback is what the tests exercise.
- The actual Modbus integration. The tests use mocks where they touch `pymodbus`.

5.5.2 Manual: hardware bring-up

The hardware-side validation that the automated tests cannot do. Run this:

- after a software change that touches the VFD or DAQ path;
- after any drive-parameter change;
- after a long facility idle period;
- before a first run of the day if any of the above applies.

Procedure

1. **Pre-run checklist complete** (Operations → Pre-run checklist).

2. **Modbus reachability:** `Test-NetConnection -ComputerName 192.168.50.10 -Port 502` → `TcpTestSucceeded : True`.
3. **PowerShell control sanity:**
   - Launch `WindTunnelControl.ps1`.
   - Confirm the live readout is updating (CW, REF1, SW, ACT1).
   - Apply a low setpoint (e.g. 100 RPM), Start, observe motor spin to 100 RPM.
   - Apply 200 RPM mid-run, observe smooth transition.
   - Stop, verify motor reaches 0 RPM.
   - E-Stop test: re-start at 100 RPM, press E-Stop, verify coast-down.
4. **ADCL WinSoft control sanity:**
   - Launch `ADCL WinSoft.exe`.
   - Confirm footer shows **Connected** (Modbus) and **DAQ: cDAQ1** (or `simulated` if the DAQ is intentionally bypassed).
   - Repeat the 100 → 200 → 0 → 100 → E-Stop sequence in the UI.
5. **DAQ sanity:**
   - With the tunnel stopped, press **Start DAQ** in ADCL WinSoft.
   - Confirm Signals tab shows reasonable engineering-unit readings — pressures near zero, angle near sensor zero, PGB sting forces near zero.
   - Spin the tunnel to 400 RPM, confirm the static-ring pressure increases and `velocity_mps` reports a sensible value.
   - Stop, press **Stop DAQ**, confirm the CSV closes.
6. **CSV format spot check:**
   - Open the just-written CSV in a text editor.
   - Confirm the header line matches the expected column set from Operations → Data acquisition workflow.
   - Confirm timestamps increase monotonically and at the expected rate.

Logging

Anything unexpected — fault codes, missing channels, calibration drift — gets a new `WT_MS_<n>/debug_session.md`. This is the only way the team builds up a track record of what is normal for this facility.

5.6 Extending the code

Three common modification recipes. Each is small enough that the right way is documented; deviating is fine when there's a real reason.

5.6.1 Add a new tab to ADCL WinSoft

`code/adcl_winsoft/src/adcl_winsoft/ui/tabs/` is where tabs live. The pattern is:

1. Create `ui/tabs/<your_tab>.py` with a `QWidget` subclass. Inject `AppState` in the constructor.
2. In `__init__`, build the layout, connect to whichever `AppState` signals are relevant, and start a `QTimer` if you need a refresh loop.
3. In `app.py`'s `MainWindow.__init__`, instantiate the tab and add it to the `QTabWidget`.
4. Add the tab name to the sidebar list in `ui/widgets/sidebar.py`.
5. If the tab needs admin gating, wrap the add step behind `state.is_admin` and re-add on `state.admin_changed`.

Use `ui/tabs/placeholder.py` as the smallest possible template.

5.6.2 Add a new Modbus register to read

If the new register is a quick-access (CW, REF1, SW, ACTn) register:

1. Add a field to `VfdSnapshot` (`vfd/__init__.py` or wherever the dataclass lives).
2. In `VfdPoller.run()` (`vfd/worker.py`), read the new address each tick and populate the field.
3. Update any UI tabs that should display it.

If the new register is a parameter (group/index):

1. Use `VfdClient.read_param(group, index)` which already does the `group × 100 + index − 1` calculation.
2. If you are reading the parameter on every poll, fold it into `VfdPoller` so you do not multiply the Modbus traffic.
3. If you are reading the parameter once at startup (e.g. nameplate values), do it on the main thread before the workers spin up, or inside `AppState.initialize()`.

Update Reference → Parameter index and Reference → Register map when adding either kind.

5.6.3 Refresh the REF1 ↔ RPM calibration

When the calibration drifts (after a hardware change, a long idle period, or a motor swap), re-run the sweep scripts and update the constants.

1. Pre-run checklist complete; test section empty.
2. Launch `code/wind_tunnel_control/ref1_sweep.ps1`:

```
powershell.exe -ExecutionPolicy RemoteSigned -File ref1_sweep.ps1 `
    -OutputCsv "C:\Users\kshit\Desktop\ref1_sweep_$(Get-Date -Format yyyyMMdd_HHmmss).csv"
```

The script walks REF1 from 0 to a maximum, polling the drive's frequency feedback, writing a CSV row per setpoint.

3. Plot the CSV: REF1 on the x-axis, drive frequency on the y-axis. A linear fit gives the new constant: REF1 ≈ frequency × (`REF1_max` / `nominal_freq`).
4. Convert the linear fit into a `REF1 = drive_RPM × C` constant. The relationship between frequency and RPM is the motor's pole-pair count and slip; for our motor `60 Hz = 880 RPM`, so the constant comes out near 22.42 when the drive is in spec.
5. Run `rpm_velocity_sweep.ps1` next to capture the RPM → wind speed mapping under the new drive calibration:

```
powershell.exe -ExecutionPolicy RemoteSigned -File rpm_velocity_sweep.ps1
```

6. Update the two source-of-truth constants:

- `code/adcl_winsoft/src/adcl_winsoft/vfd/controller.py`: `REF1_PER_RPM = <new value>`
- `code/wind_tunnel_control/WindTunnelControl.ps1`: the equivalent constant (search for `22.42`)
- `docs/manual/content/reference/calibration_constants.md`: update both numbers and the date.

7. Commit the change with a message like `vfd: recalibrate REF1 = RPM × 22.51 (2026-05-20 sweep)`. Reference the sweep CSV in the commit body.
8. File the sweep CSVs in a new `WT_MS_<n>/captures/` folder with a one-line `debug_session.md` summarizing the date, the operator, and the resulting constants.

5.6.4 A note on changing the threading model

Don't, unless there is a concrete reason. The two-worker + command-worker arrangement in `app.py` survived F.1–F.6 and the explicit "belt and suspenders" pattern (signals **and** shared state) makes the UI robust to a class of bugs that the simpler single-path designs would expose. If you are tempted to consolidate into one worker or to remove the direct shared-state path, run a long-soak test first.

6. Maintenance

6.1 Maintenance schedule

The wind tunnel is a low-maintenance facility, but it has parts that drift, parts that loosen, and parts whose state needs to be re-verified periodically. The cadences below are starting points; adjust based on actual usage.

6.1.1 Per-run

- **Pre-run checklist** (Operations → Pre-run checklist) — every time the tunnel is energized.
- **Stop-path verification** — first run of the day; all three paths (ramp-stop, E-Stop, mains kill) must work.
- **Test-section sweep** — before any model is loaded.

6.1.2 Per-week (during active research)

- **Visual inspection of cabling** between the main switch, the VFD enclosure, the motor, and the cDAQ chassis. Any visible damage halts further runs until repaired.
- **Verify static IPs on the lab PC** — `Ethernet 4 = 192.168.50.100/24`. Windows occasionally resets adapter settings during updates.
- **Verify drive parameters are intact** — `10.01 = 10`, `11.03 = 8`, group 51 IPs as documented in Reference → Parameter index. A two-minute check from the keypad or via a parameter dump script.

6.1.3 Per-month

- **Smoke generator fluid level** — check before fluid runs out mid-experiment.
- **Glycol fluid age** — replace fluid older than 12 months; old fluid changes density and dispersion behaviour.
- **Laser power supply 12 V calibration** — verify the supply still reads 12.0 V; a drifting supply means a drifting beam power and possibly a Class-2 boundary crossing.
- **Run a `ref1_sweep.ps1` sanity check** at the existing calibration point (e.g. 400 RPM) to confirm the drive is still tracking the calibrated mapping. If the wind speed has drifted by more than 5 %, schedule a calibration refresh.
- **Update door placard** if hazards, contacts, or chemical inventory date have changed.

6.1.4 Per-quarter

- **Full calibration refresh** — re-run `ref1_sweep.ps1` and `rpm_velocity_sweep.ps1`, update constants if any have drifted. See Calibration refresh.
- **PGB sting matrix sanity check** — apply a known load (calibration weight) to each axis and verify the matrix produces the expected force/moment.
- **NI-DAQmx driver / hardware re-seat** — verify the cDAQ chassis is recognized cleanly in Measurement & Automation Explorer; re-seat modules if any show as faulted.
- **Belt / coupling visual inspection** on the motor-fan drive train. Replace fluids/grease only if the maintenance docs for the specific unit require it; the motor itself is sealed.

6.1.5 Per-year

- **Drive parameter dump** committed to the repo so we have an authoritative snapshot. Use `code/wind_tunnel_control/` style scripting to read groups 10, 11, 22, 51, 99, save to CSV, file in `WT_MS_<n>/captures/`.
- **Full hazard audit** — re-walk the safety pages, update for any new equipment, re-train operators, refresh the placard.
- **Document review** — re-read this manual, the SOP, and the project notes. Anything stale or wrong gets edited or removed.

6.1.6 After every incident

Independently of the schedule, any of these triggers an immediate inspection:

- A drive fault that took more than one reset to clear.
- A model that came loose or moved in the test section.
- A person entering the exhaust path or the airflow path while the motor was energized.
- Any electrical event: tripped breaker, smell of burning, visible arcing.
- Smoke / glycol overexposure symptoms in an operator or bystander.

Each gets a new `WT_MS_<n>/` milestone with the chronological log, root cause, and what changed.

6.2 Network recovery

The most common drift in this facility is the lab PC losing its static IP on the `Ethernet 4` adapter. Symptom: ADCL WinSoft cannot reach the drive; the footer shows a red "Disconnected" indicator; `Test-NetConnection -ComputerName 192.168.50.10 -Port 502` times out.

6.2.1 Diagnosis tree

```
flowchart TD
  A[Modbus connection fails] --> B{Test-NetConnection<br/>to 192.168.50.10:502}
  B -->|TcpTestSucceeded: True| C[Application-level issue<br/>see Troubleshooting]
  B -->|TcpTestSucceeded: False| D{Ethernet 4 has<br/>192.168.50.100?}
  D -->|No| E[Re-assert static IP]
  D -->|Yes| F{Cable plugged in<br/>both ends?}
  F -->|No| G[Plug it back in]
  F -->|Yes| H{RETA-01 LEDs<br/>all green?}
  H -->|No| I[Power-cycle drive;<br/>check group 51 params]
  H -->|Yes| J[Cable is bad;<br/>swap and retest]
```

6.2.2 Re-asserting the static IP on Ethernet 4

From PowerShell **as Administrator** on the lab PC:

```
$alias = 'Ethernet 4'

# Remove DHCP-assigned address if present
Remove-NetIPAddress -InterfaceAlias $alias -Confirm:$false -ErrorAction SilentlyContinue
Set-DnsClient -InterfaceAlias $alias -ResetServerAddresses
Set-NetIPInterface -InterfaceAlias $alias -Dhcp Disabled

# Set the static address
New-NetIPAddress -InterfaceAlias $alias `
                 -IPAddress 192.168.50.100 `
                 -PrefixLength 24
```

Verify:

```
Get-NetIPAddress -InterfaceAlias 'Ethernet 4'
Test-NetConnection -ComputerName 192.168.50.10 -Port 502
```

Expect to see `192.168.50.100` and `TcpTestSucceeded : True`.

If the adapter alias is different on your install (e.g. `Ethernet`, `Ethernet 2`, etc.), identify the right one by MAC address `6C:1F:F7:C3:96:59`:

```
Get-NetAdapter | Where-Object MacAddress -eq '6C-1F-F7-C3-96-59'
```

6.2.3 Power-cycling the drive

If the RETA-01's `MODULE STATUS` LED is anything other than steady green, the recovery is:

1. **Stop any control software** that might be writing to the drive.
2. **Open the main switch.** Wait at least 30 s (the RETA-01 is mostly drained by then; the full 5-minute discharge wait is only required for opening the enclosure).
3. **Close the main switch.** The drive boots, the RETA-01 boots; LEDs should turn green within ~15 s.
4. **Verify Modbus reachability** (`Test-NetConnection ...`).
5. If the LED is still not green, Parameter recovery may be needed — `51.16` might have been switched off the ABB Drives profile, or the IP in `51.04` – `51.07` might be wrong.

6.2.4 When the cable itself is at fault

The Cat 5e cable between the USB Ethernet adapter and the RETA-01 is plugged and unplugged occasionally for desk reorganization. Bad cables produce flaky Modbus connections (occasional timeouts, slow reads). Swap to a known-good cable as the second-to-last resort before declaring an adapter dead.

6.2.5 When the adapter is dead

The Realtek 2.5 GbE adapter has been reliable, but if it dies the replacement plan is:

1. Buy any USB-to-Ethernet adapter that supports static IP configuration on Windows 11.
2. Plug it in; let Windows install the driver.
3. Identify the new adapter via the MAC address shown in `Get-NetAdapter`. Update this manual and the network recovery script with the new MAC.
4. Re-assert the static IP `192.168.50.100/24`.
5. Verify Modbus reachability.

There is nothing magical about the existing adapter — any equivalent will do.

6.3 Parameter recovery

Drive parameters can drift, get reset by a firmware update, or get clobbered by an inexperienced operator at the keypad. The set below is what the facility requires to be remote-controllable. If any of them are wrong, restore them from this page.

The full parameter set is at Reference → Parameter index. This page covers just the **critical** ones for remote operation.

6.3.1 Verify before restoring

Read the parameters from the keypad or via Modbus. Identify which (if any) are wrong. Restore only those. Avoid a "reset everything to factory defaults" approach — it changes parameters we have not documented (acceleration ramps, voltage limits, motor tuning) and can put the facility into a worse state.

Reading via Modbus

Parameter `n.m` is at Modbus address `n × 100 + m − 1`. Example: `10.01` is at address `1000`.

From `WindTunnelControl.ps1` source (or any Modbus tool), read all the entries below.

Reading via keypad

Press `MENU` → `PARAMETERS` → navigate to the parameter group. The keypad will show `<group>.<index>` and the current value.

6.3.2 Critical parameter table

Parameter	Required value	What goes wrong if it's wrong
`99.02 APPLIC MACRO`	1 (ABB Standard)	Macro change overrides everything else. Verify first.
`98.02 COMM PROT SEL`	4 (EXT FBA)	Fieldbus (RETA-01) is disabled. No Modbus communication at all.
`10.01 EXT1 COMMANDS`	**10 (COMM)**	Start/Stop commands from Modbus are ignored. Drive sits in REM but never starts.
`11.02 EXT1/EXT2 SEL`	0 (EXT1)	EXT2 is active; our config is in EXT1.
`11.03 REF1 SELECT`	**8 (COMM)**	REF1 writes from Modbus are ignored. Drive accepts the start command but spins at zero RPM.
`51.03 DHCP`	0 (disabled)	RETA-01 hunts for a DHCP server and ends up on an unpredictable IP.
`51.04`	192	IP octet 1 of the RETA-01.
`51.05`	168	IP octet 2.
`51.06`	50	IP octet 3.
`51.07`	10	IP octet 4.
`51.08`	255	Netmask octet 1.
`51.09`	255	Netmask octet 2.
`51.10`	255	Netmask octet 3.
`51.11`	0	Netmask octet 4.
`51.16 PROTOCOL`	0 (Modbus/TCP, ABB Drives profile)	Wrong profile = wrong register map = our control words become noise.

6.3.3 Restoring via the keypad

For a parameter group at the keypad:

1. `MENU` → `PARAMETERS` → navigate to `<group>` → `<index>`.
2. Press `EDIT`.
3. Use the up/down keys to set the value.
4. Press `SAVE`.

For the IP and netmask entries in group 51, the drive needs to be **power-cycled** for the change to take effect. Open the main switch, wait 30 s, close.

6.3.4 Restoring via Modbus

Some parameters are not writable via Modbus in the ABB Drives profile (the drive returns Modbus exception code 2 = ILLEGAL DATA ADDRESS). Group 51 is one of them — the RETA-01 reads its own IP configuration directly from the drive's parameter store at boot, and changing it over the running Modbus connection would cut you off mid-write. Use the keypad for group 51.

For `10.01`, `11.02`, `11.03`, `98.02`, `99.02`: these may be writable via Modbus depending on the drive's parameter lock state. The keypad is the reliable path.

6.3.5 Backup approach

A future improvement (tracked separately) is a script that reads the full parameter set into a CSV and stores it in the repo. Until that exists, the Reference → Parameter index page is the authoritative copy of the values; this page restores from it.

6.4 Calibration refresh

There are two calibrations to refresh: `REF1 ↔ drive_RPM` (the speed reference scaling) and `drive_RPM ↔ wind_speed` (the operating-velocity mapping). Both come from the sweep scripts in `code/wind_tunnel_control/`.

The procedure is described from the engineering / contributor perspective; the Software → Extending the code page has the per-step recipe (the "Refresh the REF1 ↔ RPM calibration" section).

6.4.1 When to refresh

- After any drive parameter change in groups 20 or 22 (limits and ramps).
- After a motor or coupling replacement.
- After a long idle (months without runs).
- When a sanity-check measurement (e.g. spinning at the previously calibrated 400 RPM) drifts by more than ~5 % from the previously recorded wind speed.

6.4.2 Refresh `REF1 ↔ drive_RPM`

1. Pre-run checklist complete; test section empty (no model). Tunnel is allowed to spin freely.
2. Run `code/wind_tunnel_control/ref1_sweep.ps1` from PowerShell. The script writes a sequence of REF1 values and records the drive's frequency feedback (parameter `01.02` or similar — see the script's source for the exact register).
3. The output CSV has columns: REF1, drive frequency (Hz), drive speed (RPM).
4. Fit a line `REF1 = k × drive_RPM`. The fit should be linear through the origin with a slope near 22.42.
5. Compare to the previous calibration. If the slope has changed by more than 1 %, replace the constants.

6.4.3 Refresh `drive_RPM ↔ wind_speed`

1. Without resetting between sweeps, run `code/wind_tunnel_control/rpm_velocity_sweep.ps1`. The script sets a sequence of RPM setpoints and reads the static-ring pressure differential — converted to MPH via Bernoulli using the static-ring transducer's calibrated slope/offset.
2. Output CSV has columns: setpoint RPM, drive RPM, static pressure (Pa), wind speed (MPH or m/s).
3. Fit a line `MPH ≈ a × drive_RPM + b` for drive_RPM > 300 (below that the transducer is in its noise floor).
4. Compare to the previous fit. If `a` or `b` have changed by more than 5 %, replace the constants.

6.4.4 Update the source of truth

The constants live in three places that must be kept in sync:

1. `code/adcl_winsoft/src/adcl_winsoft/vfd/controller.py` — `REF1_PER_RPM` constant.
2. `code/wind_tunnel_control/WindTunnelControl.ps1` — search for `22.42`.
3. `docs/manual/content/reference/calibration_constants.md` — both constants and the date of the sweep.

Commit the source changes with a message like `vfd: recalibrate REF1=RPM*22.51, wind=0.0815*RPM-12.98 (2026-08-12 sweep)`. Reference the sweep CSV by path in the commit body.

6.4.5 File the sweep results

Open a new `WT_MS_<n>/` milestone folder (see `docs/repository_layout.md`). Inside:

- `README.md` — what was calibrated and the result.
- `debug_session.md` — chronological notes if anything surprised you.
- `captures/ref1_sweep_<date>.csv` and `captures/rpm_velocity_<date>.csv`.

This keeps a historical record of calibration drift over time, which is useful when diagnosing whether a future discrepancy is "the tunnel changed" or "the model changed".

7. Troubleshooting

7.1 Modbus connection issues

Symptom umbrella: the control software cannot read or write the drive. The footer indicator is red, the live readout is frozen, writes return errors.

7.1.1 First three checks

1. **Network reachability:** `Test-NetConnection -ComputerName 192.168.50.10 -Port 502` from PowerShell.
   - `TcpTestSucceeded : True` → the network is fine; the problem is at the application or profile layer. Skip to Application-level issues.
   - `TcpTestSucceeded : False` → the network path is broken. Go to Maintenance → Network recovery.
2. **RETA-01 LEDs:** check the LEDs on the adapter inside the drive enclosure (visible without opening the cabinet).
   - All green → adapter is healthy.
   - `MODULE STATUS` red → the adapter cannot reach the drive's internal bus. Power-cycle the drive (open and close the main switch with a 30 s wait between).
   - `LINK / ACTIVITY` off → the Ethernet cable is unplugged or dead.
3. **Drive keypad:** check for fault codes.
   - Fault visible → Faults and alarms.
   - `LOC` in the upper-left → press `LOC/REM` to switch to remote.
   - `REM` in the upper-left and no fault → drive is fine; the problem is upstream.

7.1.2 Application-level issues

If the network reaches the drive but the application still cannot operate it, walk through:

Wrong Modbus profile

`51.16 PROTOCOL` must be `0` (Modbus/TCP, ABB Drives profile). If it has been changed to a different protocol, the register map is wrong and every read/write looks like nonsense.

Fix: keypad → `PARAMETERS` → `51` → `16` → set to `0` → save → power-cycle drive.

Wrong critical parameters

The four parameters that must be set for remote control:

- `99.02 = 1` (ABB Standard macro)
- `98.02 = 4` (EXT FBA, RETA-01)
- `10.01 = 10` (COMM start/stop)
- `11.03 = 8` (COMM REF1)

If `10.01 = 10` is wrong, the drive ignores all start/stop writes. If `11.03 = 8` is wrong, the drive accepts the start but spins at zero. Restore from Parameter recovery.

Wrong calibration constant in the software

If the drive accepts the start and spins, but at the wrong speed compared to the requested setpoint, the `REF1 = drive_RPM × 22.42` constant in the software has drifted from what the drive currently does. Re-run `ref1_sweep.ps1` and update.

Connection conflict

The RETA-01 is happy with one Modbus master at a time. If two clients (e.g. `WindTunnelControl.ps1` and ADCL WinSoft, or AeroWare and one of them) try to talk to it concurrently, they will both see flaky reads and missed writes. Close all but one.

`pymodbus` version skew (ADCL WinSoft only)

`pymodbus` 3.13 renamed `slave=` to `device_id=`. If the dev venv has been upgraded out from under the project, register reads silently raise `TypeError`. The `.exe` builds with the pinned version from `pyproject.toml`; if the bundled `.exe` works but the dev install does not, recreate the venv from `pyproject.toml`.

7.1.3 What "flaky" usually means

If the connection works but reads time out occasionally:

- **Bad Ethernet cable** is the most likely culprit. Swap with a known-good cable.
- **Two masters connected at once** (see above).
- **High-rate polling against a fresh-socket-per-call client.** Not an issue at our current rates but if you increase the polling rate above 10 Hz in `WindTunnelControl.ps1` you will hit this.

7.1.4 When all else fails

If none of the above is the answer, open a `WT_MS_<n>/debug_session.md` and start a methodical bring-up: PowerShell + `Test-NetConnection`, then `WindTunnelControl.ps1` against the drive, then ADCL WinSoft. Each step that works narrows the failure scope; each that does not is documented with the exact symptom. This is the same pattern that produced the `WT_MS_2` diagnosis of AeroWare's broken Modbus path.

7.2 Motor not spinning

Symptom: the control software issues a Start command, the drive shows no fault, but the motor does not turn (or turns visibly slower than expected).

7.2.1 Quick triage

1. **Status word check.** Read the drive's Status Word (Modbus address 3) — or look at the decoded status in `WindTunnelControl.ps1`. The interesting bits:
   - `RDY_ON` set → drive is ready, supply OK.
   - `RDY_RUN` set → drive is enabled.
   - `RDY_REF` set → drive is following its speed reference.
   - `REMOTE` set → drive is in remote mode (keypad shows `REM`).
   - `TRIPPED` set → fault is latched; see Faults and alarms.
   - `OFF2`, `OFF3` set → emergency-stop bits are active; the drive will not start.
2. **REF1 actually written?** Read REF1 back. If it is `0`, the software's write did not land (or was overwritten by a CW write). If it is non-zero, the drive saw the setpoint.
3. **CW actually `0x047F`?** Read CW back. If it is `0x0476`, the second step of the start sequence did not happen — the drive is prepared but not running.

7.2.2 Causes by symptom

REF1 stays at zero after writing

- `11.03 REF1 SELECT` is not `8 (COMM)`. The drive is ignoring fieldbus writes to REF1; restore the parameter (Parameter recovery).
- Two clients are connected; one is overwriting REF1 with zero. Close all but one.

Drive accepts Start but coasts back to zero immediately

- `10.01 EXT1 COMMANDS` is not `10 (COMM)`. The drive is ignoring fieldbus start commands; restore.
- Drive is in `LOC` mode. Keypad shows `LOC`; press `LOC/REM`.
- A latched fault is being repeatedly auto-cleared. Read the fault history at the keypad.

Drive runs but at wrong speed

- Calibration drift: the software's `REF1 = RPM × 22.42` constant is no longer correct. Refresh with Calibration refresh.
- Acceleration ramp set very long (`22.02`). Drive is on its way to the setpoint, just slowly.
- Frequency limits in group 20 are clipping the request. Read `20.01 MAXIMUM FREQ` and `20.02 MINIMUM FREQ`.

Motor mechanical resistance

- Test section blockage from a poorly mounted model. Stop, inspect.
- Belt or coupling damaged. Inspect motor and fan coupling.
- Bearing seized. Drive will trip on overcurrent quickly — see Faults and alarms.

`OFF2` or `OFF3` bit is set in SW

These are the drive's "off path 2" and "off path 3" emergency-stop bits. They have to be cleared in the Control Word before the drive will accept a start. The start sequence writes `CW = 0x0476` which clears them; if the sequence was interrupted, write `0x0476` and wait 500 ms before retrying.

7.2.3 Sanity-check loop

If you cannot tell what is going on, drop down to `WindTunnelControl.ps1` for raw register access. The PowerShell tool's status panel decodes the Status Word and the Control Word into named bits; that view is the cleanest one for figuring out what state the drive is in.

If even the PowerShell tool cannot operate the motor, the problem is in the drive itself (parameters, hardware, or firmware) — not in the software. Open `WT_MS_<n>/debug_session.md` and start parameter dumps.

7.3 Faults and alarms

The ACH550 surfaces problems as **faults** (the drive trips and refuses to run) and **alarms** (the drive continues running but warns of an abnormal condition). Codes appear on the keypad and as bits in the Status Word.

7.3.1 Reading a fault

When the drive is tripped:

- The keypad shows the fault code (e.g. `F0001 OVERCURRENT`) and the description.
- The Status Word bit `TRIPPED` (bit 3) is set.
- The drive will not accept a start command until the fault is cleared.

7.3.2 Clearing a fault

1. Identify and address the underlying cause. **Do not** repeatedly reset a fault without understanding it; that masks the warning until it becomes a hardware failure.
2. From the keypad, press `RESET`. The fault clears if the underlying condition is no longer present.
3. From software, the Control Word reset bit (bit 7) clears the fault. ADCL WinSoft does not currently expose this as a UI button; clear from the keypad.

7.3.3 Common faults

Code	Meaning	Likely cause
`F0001 OVERCURRENT`	Drive output current exceeded its trip limit	Test section blockage, sudden mechanical resistance, very short acceleration ramp at high load.
`F0002 DC OVERVOLTAGE`	DC link voltage too high	Regenerative load on a fast deceleration; lengthen `22.02 DECEL TIME` or add a braking chopper.
`F0003 DEV TEMP`	Drive over temperature	Cabinet ventilation blocked or cooling fan failed. Open the cabinet (after the 5-min wait) and inspect.
`F0006 EARTH FAULT`	Imbalanced output currents	Motor or cabling insulation issue. Stop using the drive and inspect with a megger before resuming.
`F0010 PANEL LOSS`	Keypad disconnected while in `LOC` mode	Reseat the keypad or switch to `REM`.
`F0028 SERIAL 1 ERR` / similar	Fieldbus communication lost while the drive was being controlled remotely	Modbus connection dropped; see Modbus connection.

The full fault list is in the ABB ACH550 User's Manual, which lives under `docs/reference/abb/` (file not yet committed; pull it from the ABB website if you need the exhaustive list).

7.3.4 Common alarms

Indicator	Meaning
`ALARM` bit (bit 7) set in SW	A non-fatal warning is active; the drive continues to run.
Keypad shows `Auut` / `Auxx` text	Auto-tuning or auxiliary mode hints from the drive; usually informational.
`ABOVE` bit (bit 10) set in SW	The drive is operating above its commanded speed (overshoot during rapid acceleration).

Alarms do not require operator action but should be noted in the run log if they appear unexpectedly.

7.3.5 Repeating faults

If a fault repeats within minutes of being cleared, **stop running**. The pattern is either a hardware problem (overheating, blocked airflow, mechanical resistance) or a software bug producing bad commands. Investigate before re-energizing.

For repeating faults, the diagnostic flow is:

1. Document the fault: code, time, drive parameters at the time of the trip (some are stored in fault history).
2. Reproduce the conditions if possible at a low-load setpoint.
3. Open a `WT_MS_<n>/debug_session.md` and capture everything.
4. Only resume routine operation once root cause is identified.

7.3.6 Drive history at the keypad

The keypad has a fault history menu — `MENU` → `DRIVE HISTORY`. The last several faults are stored with timestamps. This is useful when an operator reports "the drive tripped during my last run but I cleared it and went home" — the history is the only record.

8. Reference

8.1 Parameter index

Every drive parameter that this project touches, with the value it must hold and the reason. Restored from `WT_MS_1`'s bring-up notes and `WT_MS_2`'s diagnosis.

8.1.1 Application setup

Parameter	Name	Value	Why
`99.02`	APPLIC MACRO	1	ABB Standard macro. Other macros change the register map and the parameter visibility.
`99.07`	MOTOR NOM FREQ	60 Hz	Motor nameplate. Used by the drive for internal scaling.
`99.08`	MOTOR NOM SPEED	~880 RPM	Motor nameplate. Used as 100 %-reference in fieldbus scaling. `WindTunnelControl.ps1` reads this at launch.

8.1.2 Fieldbus / communication

Parameter	Name	Value	Why
`98.02`	COMM PROT SEL	4 (EXT FBA)	RETA-01 is active.
`51.03`	DHCP	0 (disabled)	We use a static IP.
`51.04` – `51.07`	IP octets 1–4	192, 168, 50, 10	RETA-01 address.
`51.08` – `51.11`	Netmask octets 1–4	255, 255, 255, 0	Standard /24.
`51.16`	PROTOCOL	0	Modbus/TCP, ABB Drives profile. Other settings = different register map.

8.1.3 Command sources

Parameter	Name	Value	Why
`10.01`	EXT1 COMMANDS	**10 (COMM)**	Start/Stop from Modbus. Without this, the drive ignores fieldbus start commands.
`11.02`	EXT1/EXT2 SEL	0 (EXT1)	EXT1 is the active command source.
`11.03`	REF1 SELECT	**8 (COMM)**	REF1 (speed setpoint) from Modbus. Without this, REF1 writes are silently ignored.

8.1.4 Limits and ramps

Parameter	Name	Value	Why
`20.01`	MAXIMUM FREQ	(site-specific)	Hard upper limit on output frequency. Verify before changing.
`20.02`	MINIMUM FREQ	(site-specific)	Hard lower limit on output frequency.
`22.02`	DECEL TIME	(site-specific)	Ramp-stop time. Too short causes overvoltage faults at high speed.
`22.01`	ACCEL TIME	(site-specific)	Ramp-up time.

The exact values for the limits and ramps depend on the motor and load; they are not documented as fixed constants here. The right values were set during the drive's initial commissioning and should be left alone unless the motor or fan is changed.

8.1.5 Reading parameters over Modbus

Address = `group × 100 + index − 1`.

Examples:

- `10.01` → address `1000`
- `11.03` → address `1102`
- `51.04` → address `5103`
- `99.08` → address `9807`

8.1.6 Where the dump lives

When we run a full parameter dump (planned per Maintenance → Schedule), the output CSV goes into `WT_MS_<n>/captures/`. The most recent dump is the authoritative snapshot; this page is the human-readable summary.

8.2 Modbus register map

The ACH550 with the RETA-01 fieldbus adapter, configured for the **ABB Drives** profile, exposes two kinds of registers over Modbus/TCP:

1. **Quick-access I/O** at low addresses (0–11). Pre-defined slots for control word, references, status word, actuals.
2. **Parameter registers** addressed by `group × 100 + index − 1`.

8.2.1 Quick-access I/O

Address	Name	R/W	Used for
0	Control Word (CW)	R/W	Drive command bits. `0x0476` = prepare / ramp-stop, `0x047F` = run, `0x0000` = off / coast.
1	Reference 1 (REF1)	R/W	Speed reference. Scaled `REF1 = drive_RPM × 22.42`.
2	Reference 2 (REF2)	R/W	Unused in our config.
3	Status Word (SW)	R	Drive state bits (READY, RUN, REM, FAULT, …).
4	Actual 1 (ACT1)	R	Speed feedback.
5	Actual 2 (ACT2)	R	Current feedback.

These are addressed directly — no group/index translation needed.

Control Word bits (relevant subset)

Bit	Name	Meaning
0	OFF1	Inverse — 1 = OFF1 inactive (the drive can run).
1	OFF2	Inverse — 1 = OFF2 inactive.
2	OFF3	Inverse — 1 = OFF3 inactive.
3	RUN	1 = run, 0 = stop.
4	RAMP_OUT_ZERO	1 = ramp output to zero.
5	RAMP_HOLD	1 = hold the current ramp output.
6	RAMP_IN_ZERO	1 = force ramp input to zero.
7	RESET	Rising edge clears a latched fault.
10	REMOTE_CMD	1 = control from fieldbus active.

The three named patterns we use:

- `0x0476 = 0000 0100 0111 0110` — OFF1/OFF2/OFF3 inactive, ramp not held, RUN=0. Drive is prepared and ramps to zero if running.
- `0x047F = 0000 0100 0111 1111` — same as above but with RUN=1 and the lower three bits set. Drive runs.
- `0x0000` — all bits zero. Drive coasts.

Status Word bits (relevant subset)

Bit	Name	Meaning
0	RDY_ON	Drive is ready, supply OK.
1	RDY_RUN	Drive is enabled.
2	RDY_REF	Drive is following its speed reference.
3	TRIPPED	A fault is latched.
4	OFF2_STA	OFF2 emergency-stop active.
5	OFF3_STA	OFF3 emergency-stop active.
6	SWC_INH	Switch-on inhibited.
7	ALARM	A non-fatal alarm is active.
8	AT_SP	At setpoint.
9	REMOTE	Drive in remote mode.
10	ABOVE	Operating above commanded speed.

8.2.2 Parameter addressing

For any drive parameter `<group>.<index>`:

$$ \mathrm{Modbus\ address} = \mathrm{group} \times 100 + \mathrm{index} - 1 $$

Examples we use:

Parameter	Modbus address
`01.01 SPEED`	100
`01.02 FREQUENCY`	101
`01.04 CURRENT`	103
`03.05 FB REF 1`	304
`10.01 EXT1 COMMANDS`	1000
`11.03 REF1 SELECT`	1102
`22.02 DECEL TIME`	2201
`51.04 IP octet 1`	5103
`51.16 PROTOCOL`	5115
`98.02 COMM PROT SEL`	9801
`99.02 APPLIC MACRO`	9901
`99.08 MOTOR NOM SPEED`	9807

Some parameters (notably group 51) are **not writable** over Modbus while the connection is live — writing them would cut you off mid-write. Use the drive's keypad for those, then power-cycle. See Parameter recovery.

8.2.3 Function codes

FC	Name	Used for
0x03	Read Holding Registers	Reading any of the above.
0x06	Write Single Register	Writing CW, REF1, individual parameters.
0x10	Write Multiple Registers	Not currently used; would be useful for atomic multi-register writes.

Exception codes returned in error: `0x82` (FC 0x03 error response), `0x86` (FC 0x06 error response) with sub-codes:

Sub-code	Meaning
0x01	Illegal function.
0x02	Illegal data address. (Common: writing a read-only parameter.)
0x03	Illegal data value.
0x04	Slave device failure.

The PowerShell client and the Python client both parse and surface these.

8.3 Calibration constants

The empirical constants that map drive register values to physical quantities. Both control codebases (`code/adcl_winsoft/` , `code/wind_tunnel_control/`) use these; keep them in sync when refreshing.

8.3.1 REF1 ↔ drive RPM

$$ \mathrm{REF1} = \mathrm{drive\_RPM} \times 22.42 $$

- **Source:** `WT_MS_2/captures/ref1_sweep_20260513_143847.csv` from `code/wind_tunnel_control/ref1_sweep.ps1` .
- **Date measured:** 2026-05-13.
- **Residual error:** ±1 % across the range tested (0–800 RPM).
- **Operating range:** 0–880 RPM (motor nameplate).
- **In code:**
- `code/adcl_winsoft/src/adcl_winsoft/vfd/controller.py` — `REF1_PER_RPM = 22.42`
- `code/wind_tunnel_control/WindTunnelControl.ps1` — the equivalent constant
- **Theoretical reference:** ABB nominal scaling is `REF1_max = 20000` at `drive_RPM = nominal_speed` . With `nominal_speed = 880 RPM` we expect `20000 / 880 ≈ 22.73` . Our empirical 22.42 is within 1.4 % — consistent with the motor running slightly above its nameplate slip estimate.

8.3.2 drive RPM ↔ wind speed

For `drive_RPM > 300` (below this, the static-ring pressure transducer is in noise):

$$ \mathrm{wind\_speed\_{MPH}} \approx 0.0822 \times \mathrm{drive\_RPM} - 13.14 $$

- **Source:** `WT_MS_2/captures/rpm_velocity_20260513_144742.csv` from `code/wind_tunnel_control/rpm_velocity_sweep.ps1` .
- **Date measured:** 2026-05-13.
- **Residual error:** ±1.1 MPH across the fitted range (400–800 RPM).
- **Behaviour below 300 RPM:** wind speed reads 0 because the differential pressure across the static ring falls below the transducer's sensitivity.

Conversion to m/s: `m/s = MPH × 0.4470` .

8.3.3 Per-channel DAQ calibration (slopes and offsets)

These live in `code/adcl_winsoft/src/adcl_winsoft/daq/channel_map.py` . Each cDAQ channel has a slope and offset that convert raw electrical units (mA, V, mV/V) to engineering units. They are *not* documented in this page because they change per-channel and per-transducer; the channel map is the source of truth.

8.3.4 PGB sting matrix calibration

The PGB sting force balance uses a 3×3 inverse matrix (`N_C1_inv.csv`) plus an optional secondary correction (`N_C1_inv_C2.csv`) to convert three bridge readings (mV/V) to three forces and a moment.

- **Files (read-only, inherited from AeroWare):** `E:\Wind_tunnel\AeroWare\Configs\N_C1_inv.csv` , `E:\Wind_tunnel\AeroWare\Configs\N_C1_inv_C2.csv` .
- **Applied by:** `code/adcl_winsoft/src/adcl_winsoft/daq/calibration.py` .

These matrices are calibrated against known applied loads. Sanity-check by applying a calibration weight to each axis and verifying the calculated force matches.

8.3.5 Sutherland viscosity (for Reynolds number)

For air at lab temperatures (~20 °C), the dynamic viscosity is approximately `1.82 × 10⁻⁵ Pa·s` and density is approximately `1.20 kg/m³` . The code uses Sutherland's law for slightly better accuracy across the room temperature range; the implementation is in `daq/calibration.py` .

Reynolds number computation uses the operator-entered model characteristic length (asked for at first launch).

8.3.6 Refresh procedure

See Maintenance → Calibration refresh for the per-step procedure. The summary:

1. Re-run `ref1_sweep.ps1` and `rpm_velocity_sweep.ps1`.
2. Re-fit. Compare to existing constants.
3. If different by more than thresholds (1 % for REF1 scaling, 5 % for velocity), update the constants in both code locations and on this page.
4. Commit the change with a reference to the sweep CSVs.
5. File the new sweep CSVs in a `WT_MS_<n>/captures/` folder.

8.3.7 Change history

Date	What	Value
2026-05-13	Initial calibration after Win11 migration	`REF1 = RPM × 22.42` · `MPH ≈ 0.0822·RPM − 13.14`

8.4 Glossary

Term	Meaning
ACH550	The ABB variable-frequency drive that runs the tunnel motor.
ADCL	Aeronautics, Dynamics and Controls Laboratory, the UCI lab that owns this facility.
ADCL WinSoft	The Python + PySide6 control application in `code/adcl_winsoft/`. The long-term AeroWare replacement.
AeroWare	The legacy LabVIEW control application from AeroLab Inc. Being retired; VFD path broken, DAQ path still works.
CW	Control Word. The drive's Modbus quick-access register for start/stop/E-Stop commands. Address 0.
cDAQ	National Instruments compactDAQ chassis hosting the input modules for pressure, voltage, and bridge signals.
EH&S	UCI Environment, Health & Safety. Phone 949-824-6200.
EXT1 / EXT2	The drive's two command sources. EXT1 is what we use; the source choice is in parameter `11.02`.
Fieldbus	The Ethernet-to-drive-internal-bus bridge. Implemented by the RETA-01 module.
LOC / REM	Local (keypad) vs Remote (fieldbus) command mode on the drive. Press `LOC/REM` on the keypad to switch.
Modbus/TCP	The protocol the control software uses to talk to the drive via the RETA-01.
PGB sting	A pressure-gauge-bridge force-balance sting mount used to measure forces and moments on a test article.
REF1	The drive's Modbus quick-access register for the speed reference. Address 1. Scaled `REF1 = drive_RPM × 22.42`.
RETA-01	The ABB Ethernet fieldbus adapter installed in option slot 1 of the ACH550.
REM mode	The drive accepts commands from the fieldbus. Required for software control.
SOP	Standard Operating Procedure. Operator-facing document at `docs/sop/`.
SW	Status Word. The drive's Modbus quick-access register reporting state. Address 3, read-only.
UCI	University of California, Irvine.
VFD	Variable-Frequency Drive. The motor controller. We use the ABB ACH550.
WSL2	Windows Subsystem for Linux v2. The lab PC's shell environment for git and editor work.
`WindTunnelControl.ps1`	The short-term PowerShell + Windows Forms VFD control GUI in `code/wind_tunnel_control/`.