CD Mapping & Control Overview

Paper Machine Cross-Direction Control — PRBS Identification, Adaptive Shrinkage Modelling, MPC Supervisory Control, Wander Detection & Feed-Forward Compensation

Updated: 8 July 2026

By: Ed Chapman — Stec Corporation

Table of Contents

- 1. System Architecture & Design Philosophy
- 2. Core Concept — The Mapping Matrix
- 3. PRBS Identification — From Sign-Multiplier to Real Cross-Correlation
- 4. Response Window Geometry & Sizing
- 5. Shrinkage Model & Coordinate Frames
- 6. Transport Delay Auto-Estimation & Manual Override
- 7. Profile Normalisation & Edge Handling
- 8. MPC Supervisory Control
- 9. Wander Detection, Frequency Tracking & Feed-Forward
- 10. Edge-Ignore Margin Recommendation from Wander Trend
- 11. Stale Scan Protection
- 12. Setpoint Ownership & Bumpless Transfer
- 13. Manual Bump Interface
- 14. CSV Tuning Log, HTML Reports & PDF Generation
- 15. Persistence (VSettings)
- 16. ZMQ / STEC Communication Protocol
- 17. File Reference
- 18. Workflow Summary & Parameter Relationships

1. System Architecture & Design Philosophy

The automap4 project implements a complete cross-direction (CD) identification and control system for paper machines. The fundamental architectural decision is separation of concerns: the machine-side processes own actuator positions and scanner measurements, while the controller (automap4) owns setpoints and discovers the process model purely by observing input/output data. This mirrors a real mill installation where the QCS/ machine interface cannot reveal internal simulator constants.

1.1 Architecture Principle

modelstdcom.py (and the deprecated modeldriver.py) pretends to be a paper machine. It knows its own physical transport delay $\tau = L / v$, the true Gaussian footprint width, the shrinkage polynomial, and the actuator dynamics. It must not share these secrets. automap4 MUST infer everything: the mapping matrix G, transport delay D, process gain Kp, response width σ, process time constant τp, and controller time constant Tc, by moving actuators and observing the resulting CD profile.

This constraint drives every algorithmic choice. For example, the mapping matrix rows are area-normalised (sum-to-one) rather than amplitude-normalised, because a real machine gives no calibrated gain reference. Gain is recovered separately from the learnt shape using $\sqrt{(2\pi)}\cdot\sigma$, which is derived from the analytical integral of a Gaussian footprint.

1.2 Process Roles

The runtime now has three roles, connected by ZeroMQ REQ/REP or by STEC Multiverse:

Process	Role	Transport	Pattern
automap4.py	ZMQ REP controller — PRBS, mapper, MPC, charts, heatmap, wander FFT, dynamic transport-delay estimation, CSV logging, PDF reports	ZeroMQ	REP (bind)
modelstdcom.py	STEC-based paper-machine simulator — replaces modeldriver.py for lab testing; communicates via STEC Multiverse	STEC / stdcomQtPS	Publisher + Subscriber
modeldriver.py	DEPRECATED ZMQ	ZeroMQ	REQ (connect)

	REQ simulator driver — PaperMachineSimulator, scan generation, scanner standardisation		
machinedriver.py	ZMQ REQ live-machine bridge — STEC subscriber for positions/profile, ZMQ worker thread for automap4 I/O	STEC + ZeroMQ	REQ (connect)

For lab testing, modelstdcom.py publishes actuator positions, profile, left/right edges, and machine speed to STEC Multiverse and subscribes to actuator setpoints. machinedriver.py subscribes to the same Multiverse topics and forwards them to automap4 over ZeroMQ. At site, the real machine QCS replaces modelstdcom.py; machinedriver.py requires no logic changes.

1.3 Data Flow

1. modelstdcom runs the PaperMachineSimulator at a configurable tick rate. Each tick advances one databox. After scan_length_ticks a complete 500- or 512-databox profile is assembled. The live actuator positions update immediately from STEC setpoints, but the simulator computes the profile effect from an actuator snapshot delayed by transport_delay_ticks = τ / tick_period.
2. modelstdcom publishes the scan_data-equivalent fields (profile, left_edge, right_edge, actuator_positions, machine_speed) to STEC Multiverse at the end of each scan.
3. machinedriver receives the STEC data in the main Qt thread and forwards it to its ZmqWorker running in a dedicated QThread. The worker sends scan_data JSON via ZeroMQ REQ to automap4.
4. automap4 receives scan_data in the ZmqWorker thread, emits sig_scan_data to the main Qt thread, and processes it in _handle_scan_data.
5. automap4 normalizes the profile, updates the mapper (PRBS or control), computes next setpoints, and sends the reply.
6. machinedriver receives the setpoint reply and writes actuator_setpoints back to STEC Multiverse, which modelstdcom (or the real machine) applies.
7. Between scans, machinedriver polls at 1 Hz via status_poll to pick up mid-scan setpoint changes (e.g. manual bumps).

2. Core Concept — The Mapping Matrix

Cross-direction control is a spatially distributed MIMO problem. For N_act actuators and N_box databoxes, the static mapping is an N_act × N_box matrix G where row i describes the normalised spatial response of actuator i across all databoxes:

```
G[i, j] = contribution of actuator i at databox j
```

Each row is area-normalised so that Σ_j G[i, j] = 1.0. This representation encodes only spatial allocation; the physical amplitude of the response is recovered later through the process gain.

2.1 Why Area Normalisation?

A dilution actuator changes basis weight by a fixed amount per percent stroke. The raw CD footprint is approximately Gaussian with peak Kp and standard deviation σ. Its total area under the curve is $Kp \cdot \sigma \cdot \sqrt{(2\pi)}$. If we divide by that area, we obtain a dimensionless shape with unit area. Therefore, if the controller knows the shape G[i, :] and estimates σ, it can recover $Kp = \sqrt{(2\pi)} \cdot \sigma$. This factor is exact for a Gaussian and remains an excellent approximation for the learnt rows after PRBS convergence.

The sign of Kp is negative for dilution actuators: opening the valve reduces basis weight. The system hard-codes PROCESS_GAIN_SIGN = −1.0. The mapping matrix stores the positive footprint shape; the controller applies the sign when building the prediction matrix.

2.2 Initial Fallback Mapping

Before any data is learnt, each row is initialised as a Gaussian centred on the hard coded S-curve (fallback_center_box) with σ = 6.0 databoxes. This provides a stable prior so that the controller can attempt cautious moves even before PRBS identification begins. The fallback centres are derived from the simulator shrinkage polynomial; see Section 5.

3. PRBS Identification — From Sign-Multiplier to Real Cross-Correlation

PRBS identification excites many actuators simultaneously with statistically independent sequences. Because the sequences are mutually uncorrelated, the response of each actuator can be separated from the others by cross-correlation, dramatically reducing the time needed to learn the full mapping compared with sequential bump tests.

3.1 Sequence Generation

A 511-length sequence is generated independently for each actuator. Values are drawn from {−1, 0, +1} with P(−1)=0.25, P(0)=0.5, P(+1)=0.25, giving a density of approximately 50%. The pointer wraps modulo 511. At each scan the sequence value is multiplied by the configured PRBS amplitude A (default 3% of stroke, configurable 1–20% in Preferences) and added to the nominal actuator position:

```
u(t) = clamp(u_nominal + A · PRBS_i(t), 0, 100)
```

The {−1,0,+1} direction indicators are stored in prbs_history; actual positions are stored separately in act_history. The latter is used for transport-delay estimation because it reflects the real movement after amplitude scaling and clipping.

3.2 Cross-Correlation & Response Extraction

The original implementation used a single-snapshot sign-multiplier: the current profile delta was multiplied by the sign of the PRBS move and dropped into the response window. This was fast but noisy, because a single scan contains uncorrelated neighbour moves and MD variations. The current implementation uses full-history cross-correlation aligned by the transport delay D:

```
prbs_slice = prbs_history[0:T-D, :]
meas_slice  = meas_history[D:T, :]
For each actuator i and each databox k in response window W_i:
    response_i[k] = Σ_t prbs_i[t] · meas[t+D, k] / Σ_t prbs_i[t]²
```

Why does this work? The measurement at time t+D is the superposition of all actuators' responses delayed by D plus measurement noise. When we correlate with actuator i's PRBS sequence, the contributions from other actuators average to zero because their sequences are uncorrelated with prbs_i. The numerator is the covariance between prbs_i and the measurement; dividing by the PRBS energy normalizes by the input power.

The response window W_i is positioned at the fallback S-curve centre converted to the normalised profile index (Section 7). Negative values are clipped to zero, the shape is Gaussian-filtered (σ = 1.5 boxes) to suppress noise, and the entire mapping row outside the window is zeroed before normalisation. Zeroing the tails is important: the broad fallback Gaussian pedestal (σ = 6 over the full frame) otherwise remains in the row and inflates the estimated width measured by the MPC controller.

3.3 EWMA Update Rules

The new response shape is blended into the existing row with an EWMA to smooth observation noise:

```
G_i[W_i] = (1 − α) · G_i[W_i] + α · response_shape
```

- PRBS_LEARNING_RATE α = 0.05 during dedicated PRBS identification.
- LEARNING_RATE α = 0.20 during natural scan-to-scan moves and control refinement.

After blending, the row is re-normalised to sum to 1. During PRBS, update_model_from_scan and update_model_from_control are gated so that only PRBS-correlation updates modify the mapper. Once PRBS stops, control moves can refine the model via closed-loop identification.

3.4 Mapping Quality & Progress Metric

Each actuator accumulates a learned_count and a per-zone quality Q_i = min(1.0, count_i / 15). The threshold 15 was chosen because with α = 0.05 an EWMA reaches approximately 99.5% convergence after 15 observations (1 − (1−α)^15 ≈ 0.995). Effective progress combines mean quality with a shrinkage-coverage bonus:

```
P = (Σ Q_i + N_unlearned · 0.5) / N_act
```

The 0.5 bonus for unlearnt zones is justified once the cubic shrinkage polynomial is fitted: the polynomial interpolates the alignment of neighbouring learnt zones, so even unbumped zones have a plausible predicted centre. Progress is capped at 1.0 once it exceeds 0.995 to avoid a stuck progress bar when a few zones stop one observation short of the threshold.

4. Response Window Geometry & Sizing

The response window is the local databox neighbourhood in which an actuator's response is expected to lie. Its size is a compromise: too narrow and the true response is truncated; too wide and neighbouring actuators' PRBS moves leak into the correlation, biasing width and gain estimates. The original hardcoded half-width of 35 boxes was much too large for a typical 50-mm actuator on a 4800-mm headbox sampled into 500 databoxes, causing severe neighbour crosstalk and inflated widths.

4.1 Geometry-Based Half-Width

The half-width is now computed from physical machine geometry and a user-configurable multiplier (default 2.0, range 1.5–2.0, step 0.1):

```
databox_width_mm = headbox_width_mm / N_box
actuator_width_boxes = actuator_width_mm / databox_width_mm
half_width = round(actuator_width_boxes · multiplier)
half_width = clip(half_width, 3, 20)
```

For a 4800 mm headbox, 50 mm actuators, and 500 databoxes, databox_width = 9.6 mm and actuator_width_boxes ≈ 5.21. With multiplier 2.0 the half-width is 10 boxes (full window 21 boxes); with 1.5 it is 8 boxes (full window 17 boxes). This is a dramatic reduction from the original 70-box window and matches the true footprint width much better.

4.2 Why the Multiplier Range 1.5–2.0?

The multiplier must cover the true 3–4σ extent of the Gaussian footprint. The simulator uses σ = 4.5 boxes, so the 1% amplitude width is about 6.07σ ≈ 27 boxes. A multiplier of 2.0 times the 5.2-box actuator pitch gives a 21-box window, which comfortably captures the central shape while excluding adjacent actuator responses that are separated by the actuator pitch. Values

below 1.5 risk truncating the response and biasing the Gaussian-fit width low; values above 2.0 reintroduce crosstalk without improving the learnt shape.

4.3 Window Positioning

All three update paths (prbs_update_model, update_model_from_scan, refine_from_control_move) position the window using the hardcoded fallback S-curve, not the learnt shrinkage model. This prevents a feedback loop: an incorrect learnt centre would shift the window, which would capture wrong data, which would shift the centre further. The learnt shrinkage model is used for display, controller model estimation, and Gaussian placement for unlearnt zones, but never for window positioning.

5. Shrinkage Model & Coordinate Frames

Because the headbox jets are angled and the sheet shrinks as it travels, the geometric mapping from actuator index to databox position is non-linear, especially near the edges. The system models this with a 3rd-order S-curve polynomial.

5.1 Fallback S-Curve

The fallback model uses coefficients that match the simulator:

```
x = i / (N_act − 1) − 0.5          # normalised actuator index, [−0.5, 0.5]
s(x) = c3 · x³ + c1 · x            # initial c3 = −0.4, c1 = 0.95
c_i = HEADBOX_CENTER + s(x) · HEADBOX_WIDTH
```

HEADBOX_CENTER and HEADBOX_WIDTH are scaled with N_box to preserve the original 20-box physical margin. For N_box = 500, HEADBOX_CENTER = 250.0 and HEADBOX_WIDTH = 460.0.

5.2 Polynomial Fitting from Learnt Centres

Once at least 4 zones have been bumped at least 2 times each, the learnt response centres are used to fit a cubic polynomial in the normalised coordinate frame:

```
actuator_norm = valid_indices / (N_act − 1) − 0.5
centres_norm  = response_centers_norm[valid] / (N_box − 1) − 0.5
estimated_coeffs = polyfit(actuator_norm, centres_norm, 3)
```

The fitted coefficients are stored in mapper.estimated_coeffs and used by expected_normalized_center_box(), refresh_fallback_mapping() for unlearnt zones, and mpc_controller.update_model_from_mapper() for controller alignment. The learnt model is preserved for display and refinement; window positioning remains on the hardcoded fallback (Section 4.3).

5.3 Asymmetric Per-Zone Shrinkage

A single global cubic cannot capture local geometric distortions such as trapezoidal sheet shrinkage. An optional asymmetric mode (default off) maintains a per-zone residual offset:

```
residual_i = c_learned_i − c_predicted_i
offset_i = (1 − β) · offset_i + β · residual_i
```

with β = 0.3. The offset is added to the predicted centre when the mode is enabled.

6. Transport Delay Auto-Estimation & Manual Override

The transport delay D is the number of scans between an actuator move and the first observable effect in the scanner profile. It is determined physically by the paper path length L and machine speed v: τ = L / v seconds, converted to scans by dividing by the scan period. automap4 cannot measure L or v directly, so it estimates D from data.

6.1 PRBS Cross-Correlation Estimator

For each actuator with at least 3 detected bumps, the method extracts the profile time series at the learnt row centre, differences both actuator positions and measurements to remove slow drift, and computes the full cross-correlation. Only the positive-lag portion [0, max_delay] is retained. The average absolute correlation across qualifying actuators is computed, and the peak lag gives the transport delay:

```
act_diff[t] = act[t] − act[t−1]
meas_diff[t] = meas[t] − meas[t−1]
corr[lag] = Σ_t meas_diff[t+lag] · act_diff[t]
D = argmax_{lag>0} mean_i |corr_i[lag]|
```

To avoid noisy single-scan outliers, the controller only switches after the same estimate is seen for two consecutive calls (transport_delay_estimate_required = 2). The initial default is 2 scans.

6.2 Manual Override

A Preferences spinner allows the operator to force a fixed transport delay. Values ≤ 0 enable auto-estimation; any positive value overrides D for both PRBS correlation and MPC control. The override is persisted in VSettings, shown as [MANUAL OVERRIDE] in the mapping info dialogue, and logged as transport_delay_override_scans in CSV files.

7. Profile Normalisation & Edge Handling

Sheet wander shifts the raw databox coordinates of the sheet edges. To make the mapping invariant to wander, profiles are mapped into the sheet coordinate frame [left_edge, right_edge] → [0, N_box−1] by linear interpolation:

```
x_norm = linspace(left_edge, right_edge, N_box)
normalized_profile = interp(x_norm, arange(N_box), raw_profile)
```

This normalisation is critical: without it, the same actuator would appear to affect different raw databoxes as the sheet shifts and correlation would average the response to zero. The helper _raw_pos_to_norm_idx converts a raw databox position (e.g. the S-curve centre) to the corresponding normalised profile index:

```
norm_idx = (raw_pos − left_edge) / (right_edge − left_edge) · (N_box − 1)
```

Response windows are sliced in the normalised profile. Edge masking uses floor(left_edge) and ceil(right_edge) to handle fractional edge positions correctly: databoxes outside the sheet span are forced to zero, while partial edge boxes are preserved.

8. MPC Supervisory Control

The CDControllerMPC class implements a single-step-ahead receding-horizon controller using SciPy's SLSQP optimiser. It is deliberately lightweight: it optimises one future move vector δu rather than a full multi-step trajectory, because the learnt model is static and the control interval is already throttled by the transport delay.

8.1 Process Model from Mapping Matrix

Every scan, update_model_from_mapper() extracts a process model from the learnt matrix:

8. For each row, compute the weighted centre and a Gaussian-fit σ using a log-parabola fit to log(row). The Gaussian fit extrapolates the true sigma from the central shape, ignoring low-amplitude fallback tails that inflate raw std-dev estimates.
9. Average σ across all learnt rows to obtain estimated_response_width.
10. Recover gain: estimated_gain = √(2π) · σ. This converts the area-normalised row back to a raw exp() footprint with peak −1.0 per unit step.
11. Build the prediction matrix G = −1.0 · (mapping_matrix · estimated_gain).T. The negative sign implements the dilution gain.
12. If ≥4 centres are available, fit a 3rd-order polynomial for actuator-to-databox alignment.
13. Set τp = max(1, transport_delay_scans) and Tc = max(2, 2·τp).

8.2 Gaussian-Fit Width Derivation

For $y = A \cdot \exp(-0.5 \cdot ((x-\mu)/\sigma)^2)$, taking logs gives $\log(y) = \log(A) - 0.5 \cdot (x-\mu)^2/\sigma^2$, which is a parabola. Expanding and collecting terms shows that the quadratic coefficient a2 satisfies $a2 = -1/(2\sigma^2)$, so $\sigma = \sqrt{(-1/(2a2))}$. The fit uses only points above 5% of peak to avoid noise/floor corruption. This method is preferred over raw std-dev because the narrow response window truncates the tails; std-dev would be dominated by the remaining fallback pedestal, while the 1%-width method underestimates σ when the window is narrow.

8.3 SLSQP Objective & Constraints

The optimiser minimises:

$$J(\delta u) = Q \cdot \Sigma(w \cdot error)^2 + R \cdot \Sigma \delta u^2 + S \cdot \Sigma(\Delta^2 u)^2$$

where error = y_meas + G·δu − y_target, w is an optional edge-ignore weight, R penalizes large moves, and the bending term $\Delta^2 u$ is the second spatial difference of u_next = u_prev + δu. Bounds enforce actuator limits and a per-scan maximum move max_delta_u; an inequality constraint enforces |u_next[i+1] − u_next[i]| ≤ max_adj_diff.

8.4 Control Throttling & Auto-Tuning

Control moves are computed every control_interval = D + (1 if delay_throttle_plus_one else 0) scans. The +1 option adds one scan of delay for stability. When control is enabled, the controller auto-tunes:

```
delay_factor = 1 + D/5
Q = clip(1/estimated_gain, 0.2, 2.0)
R = clip((1/Q)·2.5·delay_factor, 5.0, 40.0)
S = clip(σ/2 · delay_factor/2, 1.0, 40.0)
max_delta_u = clip(2σ, 2.0, 15.0)
```

High-gain dilution processes with transport delay need conservative tuning; the lower bound on R (5.0) and the upper bound on Q (2.0) prevent aggressive moves that would ring.

8.5 Edge-Ignore Margin

The control target mean and the profile error are computed over a centre region excluding the outer control_edge_ignore_db databoxes at each edge. This prevents the controller from over-driving edge actuators where the response is truncated or noisy. The same margin is used for the centre 2-sigma calculation so that the green centre trend reflects only the controllable region.

8.6 Fallback Controller

If SLSQP fails to converge, a proportional fallback is applied: $\delta u = -K_{fallback} \cdot G^T \cdot error$, clipped to ±FALLBACK_MAX_MOVE = 2.0 with Kfallback = 0.02. This keeps the loop closed while the optimiser recovers.

9. Wander Detection, Frequency Tracking & Feed-Forward

Sheet wander is the lateral movement of the sheet relative to the scanner. It appears as a common-mode shift of both left and right edges. The system tracks wander, estimates its dominant frequency, and optionally applies feed-forward compensation.

9.1 Edge Tracking

Edge positions are smoothed with a slow EWMA (α = 0.02) to separate wander from measurement noise. Wander is computed according to edge availability configured in Preferences:

- Both edges available: wander = (left + right)/2 − N_box/2
- Left only: wander = left − nominal_left
- Right only: wander = right − nominal_right
- Neither: wander = 0

9.2 FFT Frequency Estimation

A Hann-windowed real FFT is applied to the last 200 wander samples. The peak magnitude (excluding DC) gives the dominant frequency in cycles/scan. Confidence is derived from the peak-to-background ratio:

```
confidence = clip((peak_mag/background − 1) / 9 · 100, 0, 100)
```

The Wander Trend dialogue colours the line green when confidence ≥ threshold (default 90%) and red otherwise.

9.3 Feed-Forward Compensation

When enabled and confidence is high, the FFT amplitude and phase are used to project the sheet position one scan ahead. The predicted wander offset shifts actuator positions in the mapper, compensating for the anticipated sheet movement before it appears in the profile.

10. Edge-Ignore Margin Recommendation from Wander Trend

Because wander directly determines how far the sheet edges move into and out of the scanner frame, it also determines how many databoxes should be excluded from the control objective to avoid driving edge actuators on noisy or truncated data. automap4 now computes a suggested edge-ignore margin from the recent wander history and logs it in every CSV row under suggested_edge_ignore_db.

10.1 Derivation of the Suggested Margin

The suggestion is computed as:

```
suggested = ceil(max|wander| + std(wander) + 3)
```

The max|wander| term covers the largest observed lateral excursion. The std(wander) term adds a buffer for normal jitter around the excursion. The +3 databoxes is a fixed safety margin for outliers and interpolation/rounding effects. The result is clipped to [0, 100] to match the valid range of the Control edge ignore preference.

10.2 Report Integration

Both field report generators (live-state and CSV-based) display the current edge-ignore setting and the suggested starting value in the Wander Analysis section. The Recommendations section flags whether the current margin already covers the suggestion or whether the operator should consider increasing it. This gives field engineers a data-driven starting point for the edge-ignore tuning that previously had to be guessed.

11. Stale Scan Protection

During scanner standardisation or communication interruptions, scan gaps can occur. The system parses the ISO timestamp in each scan_data message and computes the gap from the previous scan. If the gap exceeds max_scan_interval (default 40 s, configurable), mapping and control are skipped for that scan and the last setpoints are held. When fresh scans resume, the loop continues automatically.

12. Setpoint Ownership & Bumpless Transfer

automap4 owns the setpoints. At the first connection, it captures the machine's actual actuator positions via setpoint_poll with first_after_init=True and uses them as the initial setpoints. This guarantees bumpless transfer: the controller does not command an instantaneous move on connection.

Subsequent setpoint priority is:

14. Stale scan: hold last setpoints.
15. PRBS active: apply PRBS moves (unless zone is manual).
16. Control active: apply MPC move.
17. Idle: echo current actuator positions.
18. Manual bump override: selected zones use bump bar values.

Manual-bump overrides are skipped during PRBS so that PRBS can move all non-selected actuators.

13. Manual Bump Interface

The Manual Bump view provides an interactive bar chart for direct actuator manipulation. Zones can be drag-selected and adjusted with the scroll wheel. Right-click options include Flatten to average, Flatten all, and Unselect. Applying the bump stops PRBS and sends the bar values as setpoints. The bump bar synchronises with automap4's computed setpoints for non-manual zones so the operator always sees the current controller output.

14. CSV Tuning Log, HTML Reports & PDF Generation

The Record button starts a CSV log with one row per scan. Fields include scan metadata, mapping progress/quality, profile statistics (including centre 2-sigma), wander metrics, controller tuning and estimates, truth diagnostics in SIM mode, machine configuration, and the new suggested_edge_ignore_db.

Field Report (File > Generate Field Report) builds a professional PDF from a selected CSV. It renders mapping heatmap, quality distribution, 2-sigma trend and wander trend charts, plus machine configuration, process estimates, controller tuning, and recommendations. After the PDF is saved it is opened automatically with QDesktopServices.openUrl() so the operator can review it immediately.

15. Persistence (VSettings)

Settings are persisted via VSettings, a QSettings wrapper storing INI files. automap4 persists ZMQ bind host/port, STEC host/port, PRBS amplitude, max scan interval, edge availability, wander FFT confidence threshold, mapping file, control progress threshold, delay throttle, response window multiplier, control edge ignore, and transport delay override. Mapping data is saved/loaded as .npz files containing the full mapping matrix, learnt counts, quality, shrinkage coefficients, and controller tuning.

16. ZMQ / STEC Communication Protocol

automap4 communicates with modeldriver/machinedriver over ZeroMQ REQ/REP with JSON messages. The init handshake advertises what the driver can provide. scan_data carries profile, edges, actuator positions, machine speed, and (in SIM mode) truth diagnostics. The controller reply contains setpoints and mode flags. setpoint_poll / status_poll messages handle mid-scan updates.

For STEC-based lab testing, modelstdcom publishes and subscribes to named Multiverse topics; machinedriver bridges these to ZeroMQ. The subscription mapping table in machinedriver assigns each field a STEC subscription name, an owner flag, a ZMQ direction (to_zmq / from_zmq), and a description.

17. File Reference

File	Purpose
automap4.py	ZMQ REP controller server — PRBS, mapper, MPC, charts, heatmap, wander FFT, dynamic transport delay, CSV logging, PDF reports,

	manual bump
modelstdcom.py	STEC-based paper-machine simulator — replaces modeldriver.py for lab testing
modeldriver.py	DEPRECATED ZMQ REQ simulator driver
machinedriver.py	ZMQ REQ live-machine bridge — STEC subscriber, ZMQ worker thread for non-blocking I/O
stec_bridge.py	AutomapSubscriber — direction-aware bridge between STEC subscriptions and ZeroMQ
mapper.py	OnlineAdaptiveMapper — correlation-based PRBS learning, EWMA updates, shrinkage fitting, transport delay estimation, profile normalisation
mpc_controller.py	CDControllerMPC — SLSQP MPC, Gaussian/log-parabola width estimation, gain conversion √(2π)·σ, auto-tuning
mapping_heatmap_window.py	MappingHeatmapWindow — mapping matrix visualization
manual_bump_window.py	_BarWidget + ManualBumpWindow — interactive actuator bar chart
stdcomvsettingsPS.py	VSettings — QSettings INI persistence wrapper
stdcomQtPS.py	Qt-style TCP client for STEC Multiverse

18. Workflow Summary & Parameter Relationships

19. Start automap4.py, modelstdcom.py (or machinedriver.py + real machine).
20. Configure machine geometry in Preferences: headbox width, actuator width, N_act, N_box.
21. Initiate STEC / ZMQ connection. Bumpless transfer captures initial actuator positions.
22. Begin scanning. scan_data (or STEC profile updates) flow to automap4.
23. Start PRBS. Independent 511-length {−1,0,+1} sequences excite all zones.
24. Mapper learns rows via real cross-correlation aligned by transport delay D.
25. Transport delay is auto-estimated (or manually overridden).
26. Shrinkage polynomial is fitted from learnt response centres.
27. Progress bar advances; at ≥90% stop PRBS and enable CD Control.
28. MPC auto-tunes from learnt model and begins supervisory control.
29. Wander FFT tracks sheet motion; feed-forward compensates when confidence is high.
30. Edge-ignore suggestion is logged and reported; operator tunes control margin accordingly.
31. Stale-scan protection pauses/resumes automatically.
32. Save grade (.npz) and generate PDF field report for records.

Key Parameter Relationships

- Process gain: $K_p = \sqrt{(2\pi)} \cdot \sigma$
- Transport delay: D = cross-correlation peak lag (actuator vs. profile) or manual override
- Control interval: D + (0 or 1 scan)
- MPC horizon: max(3, D + 1)
- MPC move bound: max_delta_u = clip(2σ, 2.0, 15.0)
- Mapping progress: P = (Σ Q_i + N_unlearned · 0.5) / N_act, capped at 1.0
- Wander confidence: clip((peak/background − 1)/9 · 100, 0, 100)
- Shrinkage S-curve: $s(x) = c3 \cdot x^3 + c1 \cdot x$
- Response window half-width: clip(round((actuator_width_mm / (headbox_width_mm/N_box)) · multiplier), 3, 20)
- EWMA convergence: 1 − (1−α)^n; α=0.05 needs ~15 observations for 99.5%
- Edge EWMA: α = 0.02 for wander tracking
- Suggested edge ignore: ceil(max|wander| + std(wander) + 3), clipped [0, 100]

Simulator Secrets (for reference only — not available to automap4)

- N_act = 96 (modelstdcom default), N_box = 500 (modelstdcom default)
- FOOTPRINT_STD_DEV = 4.5 databoxes (true Gaussian response width)
- S_CURVE_COEFF_C3 = −0.4, S_CURVE_COEFF_C1 = 0.95
- WANDER_AMPLITUDE = 6.0 databoxes (sinusoidal)
- machine_speed and L determine τ = L/v
- Negative process gain: opening dilution valve decreases basis weight