MULTI-AXIS PSO: THREE METHODS AND HOW TO CHOOSE

1. Scope

This note covers the three methods of defining where trigger pulses fire in the multi-axis Position Synchronized Output (PSO) feature of the XPS-D Universal High-Performance 1-8 Axis Motion Controller and Driver, and how to select among them for a given application. It assumes the reader is familiar with the XPS-D controller and has access to the XPS-D Multi-Axis PSO User Guide, which covers configuration mechanics, API calls and product specifications that are intentionally not reproduced here.

This note does not cover the legacy single-axis Position Compare Output (PCO) mechanism present in earlier XPS-D firmware. Separate Tech Notes on the capture-radius parameter and runtime detection of missed pulses methods treat each of these subjects. This note refers to those topics but does not develop them.

2. Three ways to define where pulses fire

Trigger-point locations in multi-axis PSO can be defined three ways:

- by distance along the actual path
- by deriving them from the trajectory
- by an explicit point file

Each condition maps to a different controller configuration and suits a different class of application. In distance-based triggering, the controller generates pulses on the fly at a fixed incremental distance along the actual traveled path. Pulse locations are not pre-defined; the controller determines each pulse position from encoder feedback as motion proceeds. The user provides the pulse pitch and the trajectory. Distance-based triggering works with any trajectory type the controller supports, including PtoP (which covers absolute and relative moves), PT, PVT, and LineArc.

Pitch is set when the mode is enabled and applies uniformly for the duration of that trajectory segment; changing pitch requires ending the current trajectory and starting a new one with the updated setting.
In trajectory-derived triggering, the controller derives an absolute pulse-location array from a LineArc trajectory file before motion begins. Pitch can be specified in two ways: as a single global value at the time the trajectory is loaded, or as values embedded segment-by-segment within the LineArc file itself. The embedded form supports both segment-varying pitch and gating — a segment's embedded pitch value can be set to disable triggering for that segment entirely. This means a single continuous trajectory can have regions of fine pitch, regions of coarse pitch, and regions with no pulsing, all defined in the LineArc file without requiring a separate point file or a break in motion. The controller builds the corresponding array of target coordinates and fires pulses as the stage approaches each target during execution. This method is currently limited to LineArc-compatible trajectories; paths that require sharp direction changes, such as hard 90-degree corners, cannot be expressed in LineArc and would require either breaking the path into segments or using a different method.
In explicit point file triggering, the user supplies the pulse-location array directly as an external file. The controller loads the array and fires pulses as the stage approaches each listed coordinate. The user is also responsible for providing a motion trajectory that passes through every listed point within the controller's capture-radius tolerance. This method supports any trajectory type, including PT and PVT, which can express nearly any motion profile including sharp corners and complex curves that LineArc cannot.
The cost of that flexibility is that the developer must build and maintain two coordinated artifacts — the

trajectory and the point file — and must ensure they remain consistent whenever either changes. Anything that trajectory-derived triggering provides through the LineArc file structure can also be built in a point file, but the work that the LineArc format does automatically must be done explicitly.

At the API level, the controller exposes two PSO modes: relative and absolute. Distance-based triggering uses relative mode. Both trajectory-derived and explicit point file triggering use absolute mode, but differ in how the pulse-location array is loaded — trajectory-derived uses a LineArc-loading call, while explicit point file uses a file-loading call. The plain-language method names are used throughout the rest of this note.

3. How to choose

The right method is the simplest one that still provides the required pulse-placement behavior for the application. The following sections describe what each method does well and the conditions under which a different method may serve better. Figure A1 summarizes this comparison.

	Distance-Based	Trajectory-Derived	Explicit Point File
User provides	• Pitch • Trajectory	• Pitch (global or embedded) • LineArc trajectory • Capture radius	• Pitch (global or embedded) • LineArc trajectory • Capture radius
Best for	• Uniform spacing along the actual path • Simplest setup • Any trajectory type	• Repeatable absolute placement • Overlay and stitching • Variable pitch and gating • LineArc path	• Complex or external pulse patterns • Trajectories that do not suit LineArc • Maximum flexibility
Main tradeoff	• Less absolute placement certainty across passes	• Requires a LineArc trajectory	• Highest user setup burden • Trajectory and point file must be coordinated
Capture radius required	No	Yes	Yes
API	Relative	Absolute (LineArc-Loaded)	Absolute (file-loaded)

Distance-Based Triggering

Distance-based triggering is the easiest method to set up and the most forgiving in operation. Because pulses are triggered from incremental distance along the actual traveled path, more aggressive trajectories with larger following errors still maintain uniform spacing throughout. This makes distance-based well-suited to applications where the process requires evenly-spaced pulses but does not require those pulses to land at the same absolute positions on repeated passes.

The effect of path length variation becomes visible in applications with large pulse counts. When pulses number in the tens of thousands or hundreds of thousands and multiple passes must overlay or stitch together accurately, small run-to-run length differences accumulate into placement drift that is visible in the process result. For these applications, the absolute placement employed in trajectory-derived triggering is meaningfully superior.

Distance-based triggering is the natural starting point for any application. If the process only needs pulses spaced evenly along the path and does not require repeatable absolute placement across passes, distance-based does that job with the least configuration work.

Trajectory-Derived Triggering

Trajectory-derived triggering is the right choice when the application needs pulses to land at repeatable absolute positions. Because the controller derives the pulse-location array from the trajectory geometry, the locations are tied to the trajectory's coordinate system rather than to the stage's actual path. Small run-to-run length variations in the actual path do not shift the pulse targets; the stage still approaches the same absolute coordinates on each pass. This is the characteristic that makes trajectory-derived triggering the preferred method for overlay processes, multi-pass material removal, and inspection applications where adjacent scan lines must stitch together accurately.

Trajectory-derived triggering also handles pitch variation and gating within a single continuous pass, without requiring a separate point file or a break in motion between segments. A process that needs pulsing in the active regions but no pulsing during turn-arounds and acceleration ramps, or a process that alternates between a coarser pitch along straight segments and a finer pitch through corners, can express either pattern in a single LineArc file. The same file that defines the motion path also defines where pulsing is active, where it is suppressed, and at what density. This embedded control is substantially less work than building an equivalent point file by hand or breaking the trajectory into separate segments with pitch reconfiguration between them.

The main limitation is the LineArc trajectory constraint. Paths that require sharp direction changes, such as hard 90-degree corners, cannot be expressed as LineArc trajectories and must be either broken into segments or handled with a different method.

Explicit Point File Triggering

Explicit point file is the method with the most flexibility and the highest setup burden. The user defines the pulse locations directly, which means any geometry, any spacing pattern, and any combination of active and inactive regions is possible. PT and PVT trajectories, which can express nearly any motion profile including sharp corners and complex curves, are available for use with this method. The cost is that the developer must build both the trajectory and the point file, ensure they are geometrically consistent, and keep them consistent as either changes.

This method is most appropriate when the required pulse pattern genuinely cannot be expressed as a LineArc trajectory — either because the motion profile includes features that LineArc cannot represent, or because the pulse locations are derived externally from a design file, an inspection result, or another process that produces coordinates independently of the motion trajectory.

Explicit point file places the most responsibility on the developer. Mismatches between the trajectory and the point file are the most common failure mode and produce missed pulses that may be difficult to diagnose. For this reason, this method benefits most from the runtime detection check described in the companion Tech Note on runtime detection of missed pulses.

4. Choose the simplest method that meets your requirements

Choosing the right method is a question of what the application actually needs, not of which method is most capable. More capability means more configuration to manage and more ways for the implementation to drift out of sync as the system evolves. A method that provides more than the process requires is not more robust — it is unnecessarily complex.

Distance-based triggering is the appropriate default for any application that has not yet identified a reason to use more. It requires the fewest inputs, works with any trajectory type, and produces no configuration artifacts that must be kept synchronized with anything else. If the process only needs pulses spaced evenly along the path, distance-based does that job with the least work.

The step up to trajectory-derived is justified by two distinct requirements, either of which is sufficient on its own. The first is repeatable absolute placement — when the process requires pulses to land at the same positions across multiple passes, or when adjacent passes must stitch together. The second is the need for pitch variation or gating within a single continuous pass, when expressing that structure in a LineArc file is more convenient than either managing a separate point file or breaking the trajectory into segments.

The step up to explicit point file is justified when the application cannot be served by either of the preceding methods. The most common reasons are that the motion path does not suit LineArc, or that pulse locations are derived externally and must be loaded directly.

A useful check before committing to a method is to ask whether requirements that seem to call for a more complex method would actually be satisfied by a simpler one. An application that appears to require explicitpoint file triggering because its pulse locations are computed externally may in fact be expressible as a LineArc trajectory, which would allow trajectory-derived triggering to generate those locations automatically. An application that appears to require trajectory-derived triggering because different regions of the path need different pulse densities — for example, a coarser pitch for one feature, a finer pitch for another, and no pulsing between features — may be achievable in distance-based mode by breaking the trajectory into separate segments with pitch reconfiguration between them. If that approach is acceptable for the application, distance-based triggering remains the simpler choice. If the process must run as a single continuous trajectory, the embedded pitch and gating of trajectory-derived triggering is the right tool.

5. Capture radius and runtime verification

Two topics that apply specifically to trajectory-derived and explicit point file triggering are treated in companion documents. This section names them and points to those documents.

Capture radius

Trajectory-derived and explicit point file triggering both use the controller's absolute-mode trigger technique, which evaluates the distance from the stage's current position to each target coordinate in the loaded array. A capture-radius parameter defines the tolerance within which the controller considers the stage to have reached a target. This parameter does not apply to distance-based triggering, which derives pulse triggers from incremental traveled distance rather than from proximity to stored target coordinates.

A capture radius set too small relative to the stage's following error will cause the controller to miss targets and truncate the pulse sequence. The User Guide describes the numerical constraints on the radius value. For guidance on sizing the radius from the stage's following error characteristics so that pulses fire reliably, see the companion Tech Note on capture-radius selection.

Runtime verification

For applications using trajectory-derived or explicit point file triggering, the controller exposes a register that allows application software to verify at runtime whether all expected pulses were generated. A nonzero remaining count at the end of a run indicates that the pulse sequence truncated before completion. This check is inexpensive and is recommended for any application that participates in a process-control or quality-assurance chain. For the detection pattern and a worked example, see the companion Tech Note on detecting missed pulses.

6. Summary

Multi-axis PSO on the XPS-D Universal High-Performance 1-8 Axis Motion Controller and Driver supports three methods of defining where trigger pulses fire. Distance-based triggering generates pulses from incremental traveled distance along the actual path; no pulse locations are pre-defined.

Trajectory-derived triggering derives an absolute pulse-location array from a LineArc trajectory file before motion begins, enabling repeatable placement,

segment-level pitch variation, and gating within a single continuous pass. Explicit point file triggering loads a user-defined array of target coordinates; the motion trajectory must be independently provided and kept coordinated with the point file.

At the API level these three methods map to two modes: Distance-based uses relative mode, while both trajectory-derived and explicit point file use absolute mode configured differently in each case.

The selection principle is to use the simplest method that still provides the required placement behavior. Distance-based triggering is the appropriate default. The step to trajectory-derived triggering is justified by a need for repeatable absolute placement across passes, or by a need for pitch variation and gating that the LineArc file structure can provide without breaking motion. The step to explicit point file triggering is justified when the motion path does not suit LineArc, or when pulse locations are derived externally and must be loaded directly.

Two companion Tech Notes cover topics important for reliable operation for applications using trajectory-derived or explicit point file triggering. These topics are selecting the capture radius from the stage's following error and verifying at runtime that all expected pulses were generated. For configuration mechanics, API calls, and full specifications, consult the XPS-D Multi-Axis PSO User Guide.

DS-072601 Multi-Axis PSO-Tech Note_07/26
©2026 MKS Inc.
Specifications are subject to change without notice.

MKS products provided subject to the US Export Regulations. Diversion or transfer contrary to US law is prohibited. Newport™ and mks™ are trademarks of MKS Instruments, Inc., Andover, MA.