CHOOSING THE CAPTURE RADIUS IN MULTI-AXIS ABSOLUTE PSO MODE

1. Scope

This note covers how to select the capture radius value used by the multi-axis absolute PSO feature of the XPS-D controller. It assumes the reader is already familiar with the distinction between absolute and relative PSO modes and has read the relevant sections of the XPS-D Multi-Axis PSO User Guide.
The focus of this note is narrow: it reviews only what the capture radius is doing during motion, why its value matters, and how to pick a value that will reliably generate all expected pulses.

This note does not cover the mechanics of configuring the controller, the API calls used to load pulse arrays, or the numerical constraints on the radius value itself. The User Guide addresses those directly. A companion Tech Note on detecting missed pulses covers how application software can detect at runtime whether all expected pulses were generated during a multi-axis absolute PSO run.

2. Why absolute mode requires a capture radius

In multi-axis absolute PSO mode, trigger pulses are generated at a predefined list of target coordinates. The controller compares the current position of each axis against the coordinates of the currently-indexed targets in the list, and fires a trigger pulse when a match condition is satisfied.

The match condition cannot require the two coordinates to be exactly equal. Target coordinates are stored as finite-precision floating-point values, and the actual position of the stage — reported by the encoder, sampled at the controller's update rate — is also a finite-precision value. The probability that any two such values are exactly equal at any given sample is effectively zero. Therefore, a tolerance is required.

The capture radius defines that tolerance. It establishes a region around each target coordinate within which the controller considers the stage to have reached that target for the purpose of generating a pulse

3. How the controller decides when to fire

The controller's pulse-generation logic operates in two phases for each target in the pulse array.

The first phase is the capture-radius entry check. While the current position lies outside the capture radius for the currently-indexed target, no pulse can be generated for that target. The controller is effectively waiting for the stage to come close enough to begin evaluating whether a trigger should fire.

The second phase begins once the current position enters the capture radius. At that point, the controller begins evaluating the instantaneous distance from the stage's actual position to the target coordinate. This distance is computed continuously as the stage moves. The controller watches for the closest-approach condition — the moment at which the distance stops decreasing and begins to increase. When that condition is met, the pulse fires. At this moment, the array index advances to the next target, and the two-phase cycle begins again for that next target.

A specific case makes this easier to follow. Consider two adjacent pulse targets at coordinates (0, 8) and (0, 11), three units apart, with the capture radius set to 8 units. Assume the stage is moving vertically upward along the line x = –2, and track the first target at (0, 8). While the stage is well below y = 0, it is more than 8 units away from the target and the controller is in phase one for that target: no pulse evaluation is occurring. As the stage crosses approximately y = 0.25, its distance to (0, 8) drops below 8 units — it has entered the capture radius — and phase two begins. The controller

now tracks distance continuously. The distance decreases steadily as the stage moves upward, reaching a minimum of 2 units when the stage passes y = 8 (with its x-coordinate still at –2). Immediately past that point, the distance begins to increase, which the controller detects, fires the pulse, and advances the array index to the next target at (0, 11).

At the instant of firing, the stage is already within the capture radius of the new target — its distance to (0, 11) is approximately 3.6 units, well below 8. Phase two for this new target begins immediately. The stage continues upward, distance decreases, reaches a minimum of 2 units at y = 11, begins to increase, and the second pulse fires. The cycle repeats for subsequent targets.

Note that the closest-approach comparison requires at least two distance samples to tell whether the distance is increasing or decreasing. This is sufficient to prevent a spurious trigger immediately after the index advances to a new target, even in cases where the stage is already inside the new target's capture radius at that moment.

Schematic of the key steps in the two-phase trigger technique. The stage starts outside of the capture radius for Target 1 (Phase 1). When it enters that, the system begins evaluating the distance to Target 1 (Phase 2). When closest approach is determined (because the distance begins increasing) the system produces a trigger. The processor then determines the distance to Target 2 and immediately finds that it is already within that point's capture radius (Phase 2). So, it begins the closest approach calculation and fires at the appropriate time. After that, the processor sees that it is not yet within the capture radius for the next target (not shown), so it is in Phase 1 for Target 3.

4. Capture radius and pulse pitch are independent

A critical concept to understand is that having a capture radius larger than the pulse pitch — even much larger — is not a configuration error and does not cause incorrect pulse placement. At first glance this seems as though it should be a problem: if the stage is inside the capture radius of two targets at once, how does the controller know which one to fire for?

The answer is that the controller only evaluates distance to the currently-indexed target. Therefore, this is the only capture radius being evaluated. The pulse fires at the closest-approach point, which is a property of the actual trajectory geometry — specifically, the point on the trajectory that is nearest to that target coordinate. That closest approach point is unique for each target (unless the same target appears more than once in the array) and it is determined by the path the stage takes, not by the capture radius value.

This is a key distinction – the capture radius determines when the controller starts paying attention to the active target (and when it acquires the first data point for the "closest approach" measurement), but the approach detection algorithm determines where the pulse fires. So, capture radius should be understood as an evaluation window and does not affect pulse-placement tolerance.

This independence is what allows tight-pitch applications to work with a comfortable capture radius. Glass cutting is a representative case, where pulse pitches of 2 to 3 µm are common. Consider the walkthrough from the previous section at that pitch: two targets 3 µm apart, with a capture radius on the order of 8 µm. The technique operates exactly as in the generic walkthrough.

The stage enters the capture radius of the first target; the distance decreases; closest approach fires the first pulse at the first target. The array index advances. The stage is already well inside the capture radius of the second target, with a distance that continues to decrease as the stage moves. Closest approach is reached 3 µm further along the path, the distance begins to increase, and the second pulse fires. Two pulses, correctly placed at their respective targets, with a capture radius nearly three times the pitch.

5. Sizing the capture radius from following error

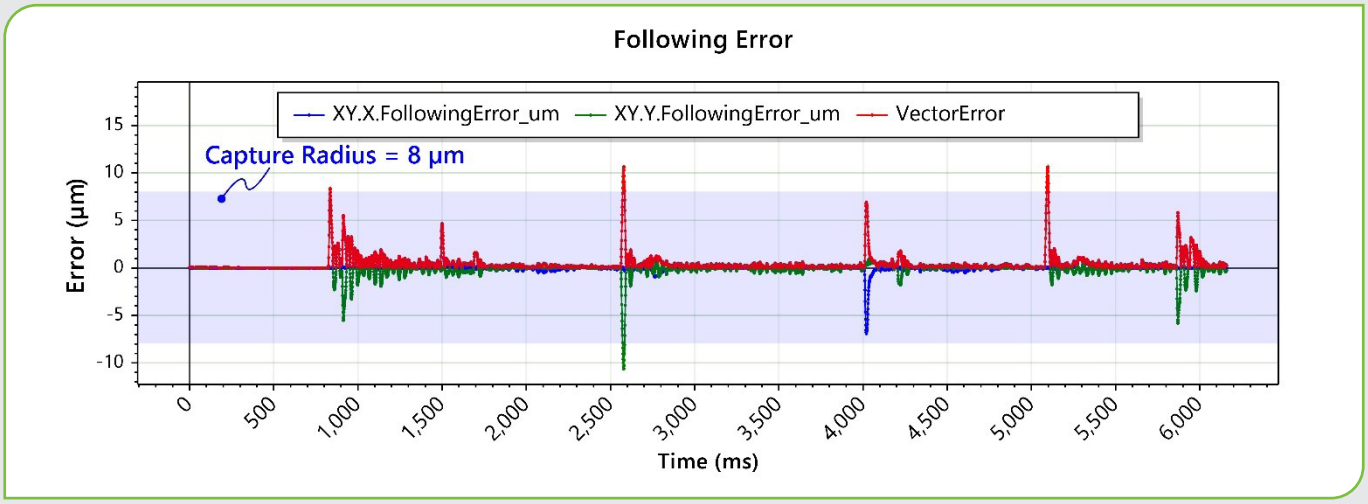
This section turns to the actual sizing consideration for the capture radius — which is not related to pulse pitch at all, but to the expected following error of the trajectory.

The capture radius must be large enough that the stage reliably enters it during the approach to each target. If the trajectory's actual path passes the target at a distance greater than the capture radius, the controller never enters phase two for that target and no pulse fires. This is the single constraint that drives radius selection in practice.

The quantity that predicts how closely the actual path will approach each target is the following error of the trajectory — the instantaneous difference between the commanded position and the actual position reported by the encoders. Following error is present on every servoed axis during motion, and its magnitude varies across the trajectory depending on velocity, acceleration, and the dynamic response of the stage. A well-tuned system has small following error; an aggressive trajectory on a stage at the edge of its capability has larger following error.

The practical sizing rule is straightforward: run the intended trajectory, capture the following error along its length, identify the worst-case value, and set the capture radius comfortably above that worst case. Margin above the worst case accounts for run-to-run variation, for the operating envelope under which the trajectory may actually run, and for other sources of variability that a single bench run will not reveal.

The following error can be captured directly by the XPS-D controller's data-gathering facility while the trajectory runs. A typical workflow is to execute the trajectory once without PSO enabled, log the per-axis following error and combine the per-axis values into a vector magnitude for each sample, and inspect the result as a plot against time or against distance along the path. The peak of that plot is the number the capture radius must exceed.



Trace showing the vector magnitude of following error, with axes (µm vs. time or distance along path). The shaded area indicates the chosen capture-radius value of 8 µm.

The amount of margin to carry above the worst-case following error is a guideline rather than a formula, and how safe a given margin actually is depends on how well the worst case has been measured. A margin of 20% above the observed peak — a capture radius set to 120% of the worst-case vector following error — has proven sufficient in practice for well-tuned stages running well-characterized trajectories. Looser margins cost nothing in most cases, because the closest-approach logic from Section 3 means that a capture radius larger than strictly necessary does not move the pulse or change where it fires.

The important phrase in the previous paragraph is worst case. A 20% margin is only as safe as the peak it sits above. Two considerations determine whether that peak is actually the worst case, or merely the worst case observed in a single run.

The first is run-to-run variability. The following error measured in one bench run is a sample, not a guarantee. Small differences in temperature, servo state, and ambient conditions will produce slightly different traces on subsequent runs.

The second is variability within the application envelope. Following error depends on velocity, acceleration, payload, and the specific region of the stage's travel in which the trajectory executes. If any of these can vary within the intended use of the system — for example, if the trajectory or velocity is programmable by the end customer, if the payload changes between runs, or if the same trajectory may be executed at different locations within the stage's travel area — then the worst case is not a single number but an envelope. Sampling one configuration within that envelope and treating its peak as the worst case will set the radius too small.

The practical recommendation is to characterize following error across the range of conditions under which the trajectory will actually run, identify the peak across that range rather than within a single configuration, and apply margin to that peak. For applications where the operating envelope is narrow and well-understood, a single representative bench run is sufficient. For applications where the envelope is broad or where end-customers configure trajectory parameters, characterization must reflect that breadth.

In some applications — particularly those where pulse points are closely spaced and a larger capture radius would be undesirable — a tighter radius is defensible by accounting for the direction of the following error relative to the trajectory. Following error along the direction of travel does not threaten pulse generation, because the stage will still pass through each target along its intended path; it will simply arrive slightly earlier or later than commanded. Only the component of following error perpendicular to the path needs to fit inside the capture radius. Extracting this component from logged following-error data requires additional computation and is beyond the scope of this note.

6. What happens when the capture radius is too small

Because of how the controller handles a missed target, the consequence of missing a pulse is more severe than it may first appear. Specifically, the controller does not skip the missed target and advance to the next one. It waits.

The pulse array is executed in strict order. When the stage moves past a target without entering its capture radius, the controller remains in phase one for that target, still waiting for the stage to approach closely enough for phase two to begin.

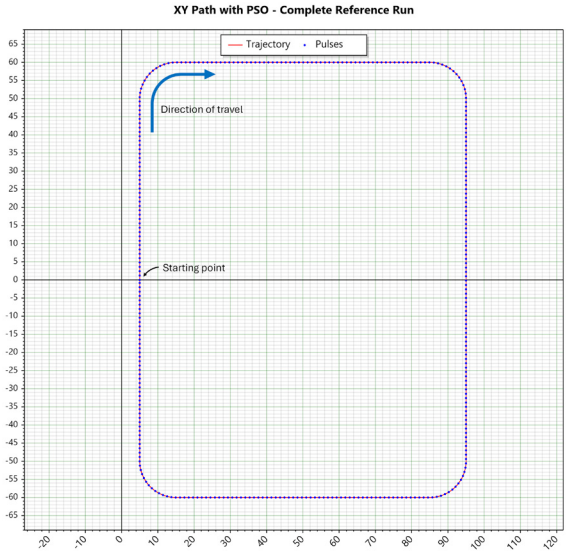
The stage continues to move along its commanded trajectory — the missed target has no effect on motion. But it is now moving away from the target on which the controller is still waiting, meaning it will never advance to the next target in the array for the rest of the run.

The practical effect is that a single missed target truncates the pulse sequence at that point. All subsequent targets in the array produce no pulses, regardless of how well the trajectory passes through them.

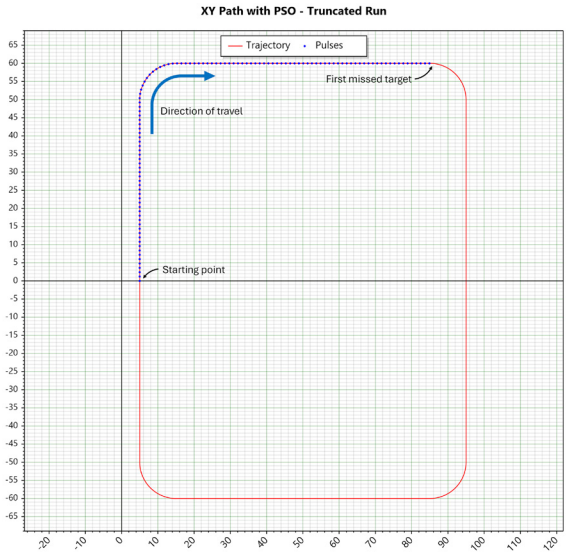
7. Summary

Capture-radius selection for multi-axis absolute PSO is a four-step procedure.

1. Run the intended trajectory with data gathering enabled.



A complete reference run, with the full pulse set generated as expected.



Trajectory plot showing a run with a deliberately-tight capture radius. The trajectory executes to completion, but pulse generation stops at the first missed target. The severity of this failure mode is what motivates the margin for capture radius recommended in the previous section. A capture radius chosen at exactly the worst-case following error has no tolerance for the ordinary variability that distinguishes one run from another; a capture radius chosen below that value will fail immediately. Sizing with margin is inexpensive and, as noted in Section 3, does not compromise the placement of pulses that do fire.

2. Capture the per-axis following error, combine into a vector magnitude, and identify the worst-case peak along the path.

3. Set the capture radius above that peak, with margin large enough to allow for run-to-run variation and for the range of conditions the trajectory will actually run under — a 20% margin is a reasonable starting point for well-characterized applications.

4. Verify the choice by running the trajectory with PSO enabled and confirming that the expected number of pulses is generated.

The capture radius is not constrained by the pulse pitch, and there is no penalty for choosing a radius larger than strictly necessary. The only practical failure mode is a radius chosen too small, which will cause the pulse sequence to truncate at the first missed target.

For the API calls used to set the capture radius, verify its numerical acceptability, and load and execute a pulse array, consult the sections on radius setting and on the GroupMultiAxisPSOAbsoluteRadiusSet API in the XPS-D Multi-Axis PSO User Guide. For runtime detection of missed pulses, see the companion Tech Note on that topic.

DS-072603 Choosing the Capture-Tech Note-B_07/26