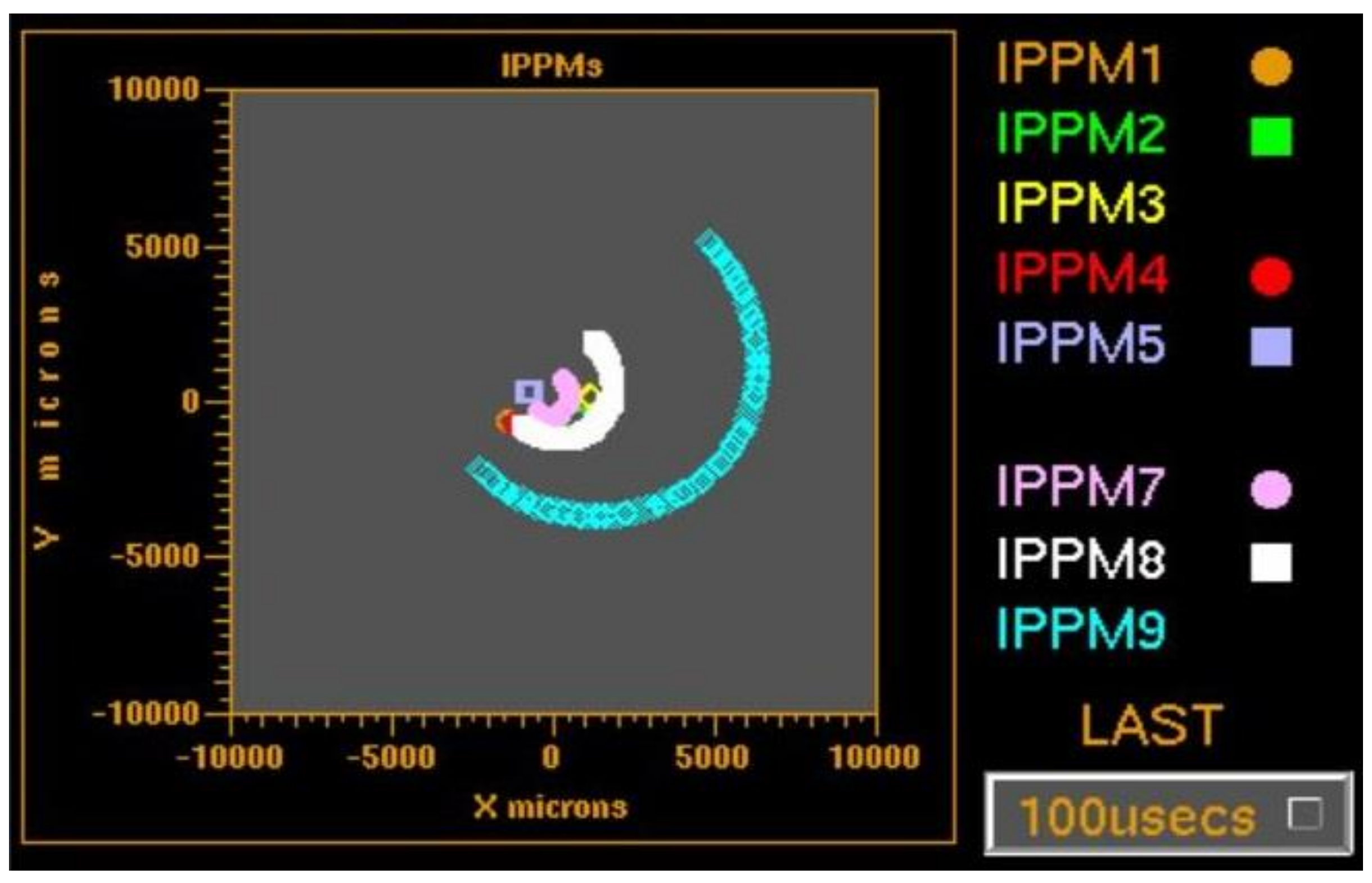
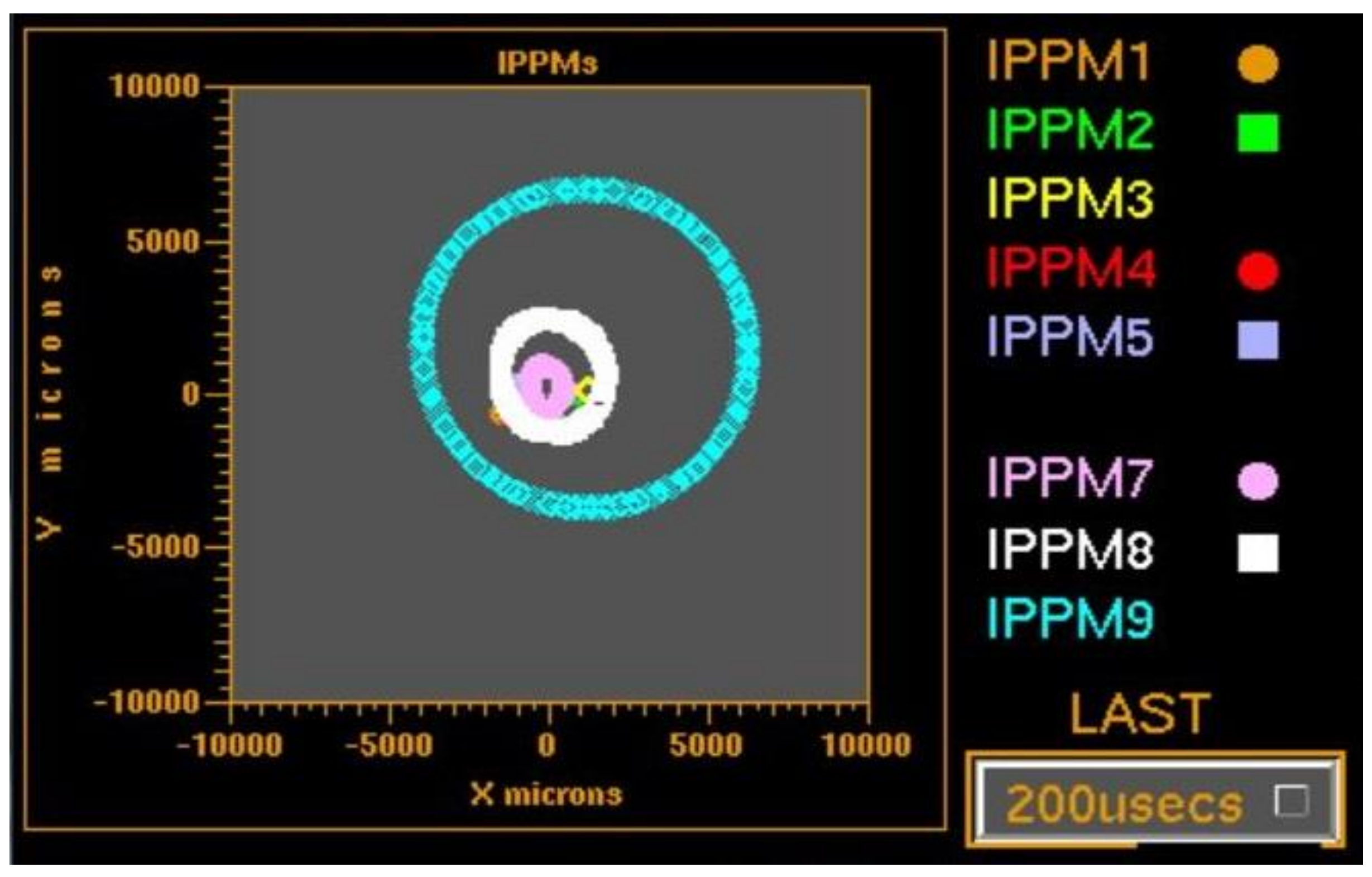
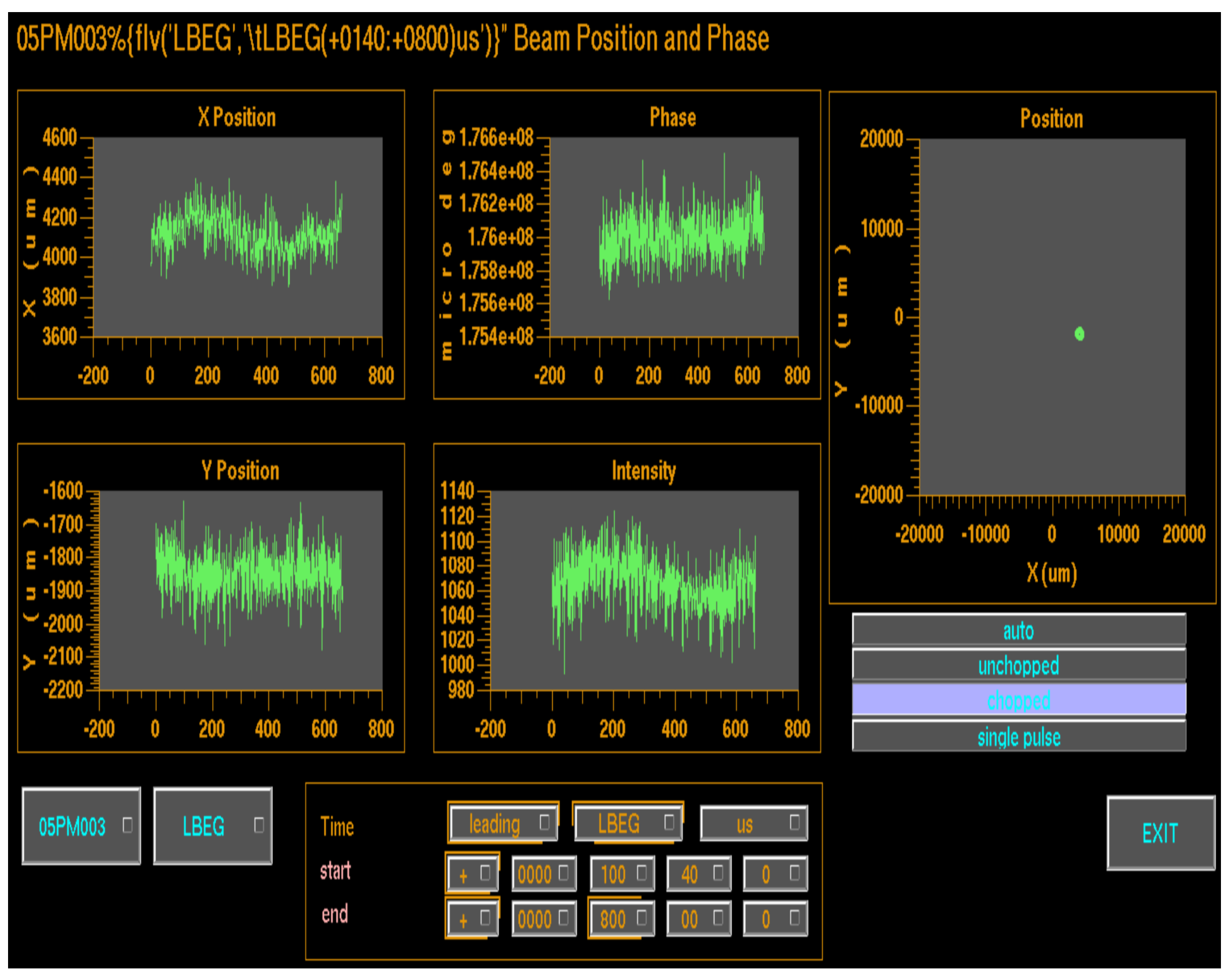
Lua–Language–Based Data Acquisition Processing EPICS Subscription Filters

Jeffrey O. Hill AOT-IC







Abstract

EPICS has been upgraded enabling client side tools to receive subscription updates filtered selectively by Lua–language–based data acquisition processing subscription update filters, specified by snippets of Lua-language-source-code embedded within the EPICS channel-name's postfix. We will discuss the generalized utility of this approach across a wide range of data acquisition applications, projects, and platforms; the performance and robustness of our production implementation; and our operational experience with the software at LANSCE.

Lua

"Lua", a language designed specifically to be embeddable within other software, was created in 1993 by members of the Computer Graphics Technology Group (Tecgraf) at the Pontifical Catholic University of Rio de Janeiro, in Brazil. "Lua" (pronounced LOO-ah) means "Moon" in Portuguese. It is a dynamic typed language, allowing automated conversion between string and numeric types, with a mixture of C-like and Pascal-like syntax. Lua is easily interfaced with C-language software.

Lua - benefits

Lua provides unique features suitable for its embedding within the core of EPICS, and for improving the overall utility of EPICS. Lua provides efficient, compiled to byte-code virtual machine execution, a compact footprint, a portable implementation, and incremental garbage collection. Lua exception handling ensures that the sequence of nested function calls conveying execution to a failure-source code line might be reported. Lua has been successfully deployed into many industrial applications, and based on this reputation it is expected to be robust. Lua has a comprehensive set of features, and powerful adjunct-libraries written by an active user-community. Lua is well proven for configuration, scripting, and rapid-prototyping, and is a strong return-for-effort candidate functionally upgrading weak areas in the pre-existing implementation of EPICS. Finally, Lua has a liberal MIT license, compatible with EPICS.

Lua - negatives

There are some negatives. In particular, with Lua the default scope of variables is global, arrays start at one nonetheless storing data at index zero isn't prohibited, and there is ambiguity between nil-valued contrasted with non-existent table elements. Lua lacks support for user-defined-type dedicated memory allocators appropriate within memory-constrained systems.

Filter Syntax

Our design *configures* the subscription update filters with a snippet of Lua code specified within a CA channel name postfix. This approach avoids revising the source code of CA Client general-purpose community obtained application-programs. Two basic forms of this channel name postfix both begin with a percent character followed by Lua source code enclosed by square or curly brackets specifying respectively a direct-action filter or a factory.

Lua Filter Specification Channel Name Postfix Examples

myPV% [=[val >= 3.2 and val <= 3.4]=]
myPV%[val.alarm.condition.severity~=0]
myPV %[3.4<val]
myPV %[==[val%3.4]==]
myPV %[===[val==3.2]===]

Lua Factory Specification Channel Name Postfix Examples

myPV % {=={myFilterFactory ('blue')}=}
myPV%{myChannelFactory()}
myPV %{ myApplicationsFactory(10,2)}
myPV % {==={flavour('savoury')}===}

Lua Filter Interface

Filters are called, passing the subscription update payload, in an argument named *val*. Filters return *nil*, *false*, *true*, or a *data-object* for conveying *supress*, *supress*, *send*, or *send replacing the update payload's value with the returned data-object* respectively.

Lua Factory Interface

A factory may return either type Boolean, a direct-acting filter function, or a channel-object . Factories return Boolean false or true for when *all* subscription updates will be permanently disabled or enabled respectively. A channel object can provide a method named `filterFactory` returning type Boolean or a channel specific direct-acting filter function.

Lua Data Access Objects

A set of Lua *classes*, serve as proxies for *each* of the Lua primitive types. They enclose a Lua primitive type, and also a C++ 11 smart-pointer to the Data Access Catalog container introspection interface. Lua operators behaving transparently as proxies for the enclosed Lua primitive type are provided. The Lua index operator returns a subordinate property object. Subordinate properties are conveniently accessed simply as ordinary variables within Lua source code. See below.

val.alarm.condition.severity

Lua Error Handling

When asynchronous requests fail within the server, a detailed multi-line diagnostic message is conveyed to the application's response callback method. A diagnostic message is essential when providing the sequence of nested function calls leading up to the source line of a Lua execution-failure. Detailed diagnostics messages are also forwarded to clients when there are Lua compilation errors.

LANSCE Filters

At LANSCE site-specific flavour filters and time-slice filters have been implemented. Example filter syntax is provided below. Filter one selects cycles with gate H+IP and also sans both gates H-GX and MPEG. Filter two replaces the CA payload with elements 50 through 150 of the waveform data. Filter three selects cycles that have beam gate H+IP, replacing the payload with the first 150 µs of the waveform. Filter four replaces the CA payload with -30 through -10 µs of waveform data before the falling edge of gate MPEG, selecting only cycles containing MPEG. Filter five, replaces the CA payload with 100 µs after waveform rising edge through 150 µs before waveform falling edge selecting only cycles containing LPEG. Filter six, selects 100 µs after gate T0 through 15 µs before waveform end for *any* flavour.

1	`XXTDAQ001D01%{flv('H+IP no H-GX MPEG')}`
2	`XXTDAQ001D01%{tim('(50:150)em')}`
3	`XXTDAQ001D01%{flv('H+IP','(0:150)us')}`
4	`XXTDAQ001D01%{tim('~MPEG(-30:-10)us','MPEG')}`
5	`XXTDAQ001D01%{flv('LBEG','(100:~(-150))us')}`
6	`XXTDAQ001D01%{tim('(T0(100):~(-15))us')}`

Site-Specific Filters

We emphasize, that *site-specific polymorphic Data Access container interface adapters are fully supported and easily implemented for site specific properties*. We emphasize, that *site-specific Lua filters are fully supported and easily implemented for site specific purposes*. The LANSCE specific Lua filters, implemented as C-language source code Lua snap-ins, along with LANSCE specific polymorphic Data Access container interface adapters are provided as examples on the world-wide-web.
https://git.launchpad.net/~johill-lanl/+git/lansce-filters
https://code.launchpad.net/~johill-lanl/epics-base/server1