ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV040

A PYTHON PACKAGE FOR GENERATING MOTOR HOMING ROUTINES

A. S. Palaha, T. Cobb, G. Knap, Diamond Light Source, Didcot, UK

Abstract

Diamond Light Source uses hundreds of Delta Tau Turbo PMAC2 based motion controllers that control motors with precision and repeatability. Homing is critical to these requirements; it safely moves axes to a well-known position using a high-precision device for detection, leaving the overall system in a well-known state and ready for use. A python package called "pmac_motorhome" has been developed to generate homing routines for multiple motors across multiple motion controllers, allowing the user to write a script that is terse for standard/typical routines but allows for customisation and flexibility where required. The project uses jinja templates as `snippets' to generate the homing routine code written in Delta Tau PLC notation (PLC is the name for logic programs in Delta Tau motion controllers). The snippets can be re-ordered and grouped together, supporting the design of homing routines for multi-axis systems with mechanical limitations that require an orchestrated approach to safely home the axes. The python script using the package is kept terse using a context manager and can group axes together to the same homing group easily.

WHAT IS "HOMING" A MOTOR/AXIS?

Motors, sometimes referred to as "axes", typically turn rotational movement into moving some load. In the operation of an x-ray synchrotron and its associated beamline laboratories, there are many motors used on scales that require precise and repeatable movements.

The position of axes is typically tracked with an encoder that produces signals as the axis turns; these signals are monitored by the motion controller and converted into positions. Assuming no power loss or miscounting of encoder signals (e.g. due to speed or slippage) then the position will remain accurate. However, the counting must start from a known position, and this must be recoverable in the event of power loss or miscounting. This is where homing procedures are critical.

The known position can be provided either by a dedicated home switch, activated when the axis reaches a certain position, or one of the end-of-travel limit switches that are usually present. The manner of activation is also important; sometimes requiring moving to the switch and then in a particular direction to release the switch, or to approach the switch from a particular direction.

Another method of creating a homing signal can be to drive against a hard stop which may generate a following error (the difference between the demanded move and measured move). This is generally used for small axes that can tolerate driving into a hard stop.

Consideration must be made for axes that are coupled; that is when movement in one axis will affect another such as when they are attached to the same load. Such scenarios require homing routines that can act on multiple axes. With the possibility of many different combinations of axis types and multi-axis systems, the ability to create tailored homing routines for each axis or group of axes is a necessity.

WHY IS PMAC_MOTORHOME NECESSARY?

The first homing routine generator written in python for Diamond Light Source (DLS) was called motorhome.py. It generated homing routines as PLC code for Delta Tau motion controllers and started as a single script. As different homing scenarios became necessary, this script grew into a large monolith that was difficult to maintain and became inflexible in some scenarios. The python interface to motorhome.py could not be used to add a custom piece of PLC code that might be used for unique or rare homing scenarios.

PROJECT REQUIREMENTS

The new homing routine generator had to satisfy the following requirements to be a viable replacement for motorhome.py, and to ensure minimal disruption/effort in converting the original generator scripts into the new style:

Maintain the EPICS interface

The EPICS Input Output Controller (IOC) used to control and monitor the Delta Tau motion controller device also monitors the status of the homing routines by polling the program variables, or "p-variables". For instance, the State, Status and Group number of homing routines are accessed through $(PLC)00, $(PLC)01 and $(PLC)02 respectively; where $(PLC) is the number of the PLC program (there can be up to 32 programs stored on the motion controller). So, for the homing routine stored in PLC9, the Status of the routine would be held in p-variable 0901.

The State and Status are enumerations that indicate if a routine is operating, has completed, failed, or been aborted. The Group number indicates whether a particular axis in a group is being homed, or all axes in the group are being homed. The group is configured in the python script that imports the generator module.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV040

Have a Python File/Script per Motion Controller or Group of Motion Controllers

The PLC code generated and stored for motion controllers at DLS is organised firstly by domain (either beamline laboratories or storage ring areas), and then by motion controller device. For example, there is a directory for a particular beamline laboratory (BLxxI) inside which there is a directory for each motion controller. The domain level directory typically contains the python script that imports the generator python module. This script configures which axes will be grouped together and how they will be homed.

Some domains have a python script per motion controller, so this must also be achieved by pmac_motorhome.

Home up to 16 Axes at once

The original homing routine generator allowed the grouping and homing of 16 axes together from one motion controller. Although rarely used in practice, this must be achievable with the new generator to maintain compatibility.

Allow Insertion of Custom Routines

Some motion setups require a specific set of homing instructions that are uniquely used, and therefore not worth adding to the generator module as a pre-defined homing type. Originally this was achieved by generating the closest matching homing routine and then modifying it manually. Though this worked, and was stored under version control, it was not automatically repeatable when the generator script was run.

The pmac_motorhome module should allow the insertion of text strings into the generated homing routine. As the typically used pre-defined homing routines would be a list of actions that would apply to motor axes, users of the python configuration script should be able to define custom sequences of actions to make a new homing routine, with custom code inserted where required.

User Friendly Python Interface

The structure of the python configuration script using the original generator, motorhome.py, was arguably simple but did not make an easily human-readable layout. Because the python configuration script is called once for each motion controller using a Makefile system, the configuration script is set out in blocks according to the name of the homing PLC to be generated, so there may be multiple blocks per motion controller. As a homing PLC is configured by adding motors to the PLC object when using motorhome.py, the configuration and grouping is done per added motor, so it is not clear which motor axes are grouped together without studying the assigned group numbers.

The pmac_motorhome generator will make use of context managers to structure the configuration script. This should group motor axes together and has all the groups for a particular PLC object listed under it, with the required python indentation providing a visually easy to understand structure. This also allows changes or configurations to be applied to motor axes, groups, or to the overall homing PLC explicitly.

Additionally, it should be possible to explicitly define a homing routine as a sequence of actions for a particular group or motor axis. For repeated sequences, such a list of actions can be defined as a separate function. The most commonly used homing sequences are provided in pmac_motorhome as pre-defined functions.

During most of Diamond's operation, it was the controls engineer's responsibility to create and maintain the PLC code for homing routines. Moving forward, this responsibility will be shared between the controls and motion engineers, combining experience and expertise in motor control and operation. This should also assist in the commissioning process of new motion axes. Having a more user-friendly interface will help the creation and maintenance of the homing routines between the two groups of engineers.

The new generator tool and interface should use python3, instead of python2 like the original generator. This should bring the tool into line with currently supported python versions and allows the use of some of the new features in the underlying pmac_motorhome code.

Can Reproduce Existing Homing Routines

To have as minimal disruption as possible during the replacement of the original homing routine generator with pmac_motorhome, a tool should be provided to convert an existing python script that imports the original generator into a pmac_motorhome style python script. This should automate the majority of the conversion process, to simplify the adoption of the new generator by controls engineers and provide confidence that existing and working homing routines will not be changed in the process. This should be demonstrated during the conversion process by comparing the outputs of the old and new generators when run for a particular beamline or domain of motion controllers.

Enforce DLS Homing Conventions

Conventions for homing PLC code at DLS includes only using PLC numbers above 8 for homing routines (1-7 are reserved for specific non-homing functions) and having all axes in the same group home with the same homing sequence.

PYTHON INTERFACE

The user interface of the pmac_motorhome generator is a python3 configuration script that imports the pmac_motorhome package. This python3 script will configure PLC objects that will render a homing routine PLC file.

The python interface uses context managers to specify the configuration of the different layers of the PLC file; the top-level object is the PLC, within this there are

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV040

groups, and within those are axes. Even solitary axes need to be defined in a group, as the group number is used to monitor homing progress and state.

In python, a context is created using the key word "with" followed by the instantiation of the object, and finally a colon; and then the next indented lines are now within the created context. Context managers allow the allocation and release of resources as needed. Boiler plate actions such as inserting a tidy-up snippet of PLC code or writing a PLC code to file can be implemented in the PLC object class. These boiler plate actions are necessary for every homing PLC, and would be executed implicitly without including it in the user level configuration script.

A simple example of a configuration script is shown in Figure 1. The editor used in the example, and in the development of this package, is Visual Studio Code.

```
from pmac_motorhome.commands import group, motor, plc
from pmac_motorhome.sequences import home_hsw

with plc(
    plc_num=12,
    controller="GeoBrick",
    filepath="/tmp/PLC12_SLITS1_HM.pmc",
):
    with group(group_num=3):
        motor(axis=1)
        motor(axis=2)

        home_hsw()
```

Figure 1: simple configuration script for a homing PLC

The use of the python language lends itself to defining custom routines that are repeatedly called. Such routines can be called inside a group to operate on some given axes, and can make use of pre-defined functions that insert blocks of code, e.g. "drive_to_limit(direction)" or "jog_if_on_limit(direction, limits)". An example of a custom routine for a group of four axes is shown in Figure 2.

```
from pmac_motorhome.commands import PostHomeMove, group, motor, only_axes, plc
from pmac_motorhome.sequences import home_hsw
from pmac_motorhome.snippets import drive_to_limit


def custom_slits_hsw(posx, negx, posy, negy):
    drive_to_limit(homing_direction=False)  # drive all slits to limit away from home

    with only_axes(posx, posy):  # home and return to limit only positive slits
        home_hsw()
        drive_to_limit(homing_direction=False)

    with only_axes(negx, negy):  # home and return to limit only negative slits
        home_hsw()
        drive_to_limit(homing_direction=False)


with plc(
    plc_num=12,
    controller="GeoBrick",
    filepath="/tmp/PLC12_CUSTOM_SLITS_HM.pmc",
):
    initial = PostHomeMove.initial_position
    with group(group_num=2, post_home=initial):
        motor(axis=1, jdist=-400)
        motor(axis=2, jdist=-400)
        motor(axis=3, jdist=-400)
        motor(axis=4, jdist=-400)

        custom_slits_hsw(posx=1, negx=2, posy=3, negy=4)
```

Figure 2: example of custom homing routine definition

Custom code snippets can also be introduced, and with the control of the sequence of pre-defined and custom blocks even more control of the customisability of homing sequence generation is possible.

Finally, the layout and indentation due to the use of context managers creates a more readable configuration script; where groups of axes are visually one block of code. The naming of the snippet functions and routines aims to be as descriptive as possible, while the availability of pre-defined homing sequences for the most typical and simple of homing scenarios will allow for terse and concise configuration scripts for most homing scenarios.

GENERATING PLC CODE

The python configuration script creates a PLC object that contains groups of axes, and all these objects are instances of classes that contain data members detailing the homing sequence to be performed. These PLC objects are fed to the homing sequence PLC generator.

The pmac_motorhome generator compiles the homing routine PLC code by piecing together blocks of code formed from code templates. The Jinja engine is used to interpret and fill in the templates. The templates contain special place-holders allowing python-like code to be inserted and eventually rendered into text strings. These templates are referred to as "snippets" in the pmac_motorhome package, and generally correspond to functional blocks of code, such as the "drive_to_limit(direction)" function, among others.

The interpreted code in the templates allows loops to be performed, and more snippets to be invoked and inserted in place, such as for multiple motors in the same group. It also provides conditional logic, allowing for snippets to be included if, for example, an option to perform a move after homing is supplied. When invoking a particular snippet with this option, the corresponding code would be rendered, otherwise it would be omitted. The snippet shown in Figure 3 shows the post home action snippet, using an if-statement to control whether the code template is rendered or not. If the python class member group.post contains a non-empty string, the template will render.

```
{% if group.post and group.post != "" %}
{% include "debug_pause.pmc.jinja" %}

;---- PostHomeMove State ----
if (HomingStatus = StatusHoming or HomingStatus = StatusDebugHoming)
    HomingState=StatePostHomeMove
    ; Execute the move commands
    if (HomingStatus = StatusHoming or HomingStatus = StatusDebugHoming)
        {{ group.post }}
    endif
endif

{% endif %}
```

Figure 3: snippet inserting code for post home actions

The snippets are structured in such a way as to represent simple and broadly singular functions, for the benefit of developers and maintainers of the code and to correspond to the required structure of the homing PLC code that already exists at DLS for its Delta Tau motion controllers. A particular snippet will usually perform one function, however there is a base snippet that includes all the beginning and ending boiler plate, and contains loops for all the groups and axes contained in the PLC.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV040

The final step is writing the rendered code into a file, the path for which is specified in the configuration of the PLC object.

CONVERTING TO THE PMAC_MOTORHOME GENERATOR

A converter tool is provided in the package to convert the original python configuration scripts in the DLS motion directories to the new style of python script using the pmac_motorhome package. This is to facilitate the adoption of the new style of homing code generation with as little intervention as possible, and to check that the new generator can produce the same homing PLC code as that which already exists.

The converter tool (written in python3) operates on a domain level or motion controller level directory and creates two copies of these directory structures in a temporary location: old and new. The old copy has the original generator script called through a subprocess call in a python2 shell (as the original generator is a python2 tool).

The new copy is also run through a python2 shell with the original generator script, but the generator code is "shimmed" (or substituted) with code that records the data structures produced without creating a file on disk. These data structures are the PLC objects that contain group and axis objects and are required in order to create a pmac_motorhome style configuration script. As the converter tool is run in python3 and the shimmed motorhome.py generator is run in a separate python2 shell, the PLC data object must be loaded in some way into the python3 instance of the converter tool. This is achieved by creating a FIFO (First-In-First-Out) pipe file from inside the python3 converter tool. Then the shimmed motorhome.py generator in python2 will write to that FIFO file a copy of the PLC data objects it creates.

To ensure the data is compatible between python2 and python3, the PLC data object is first serialized into a byte stream using the pickle package before being encoded using the struct package into a packed form. This encoded byte stream is written to the FIFO file, where it is read out by the converter tool in python3 that is monitoring it. The byte stream is unpacked and then un-pickled to extract the native python data structure of the PLC object. Using this PLC object, the configuration script using pmac_motorhome can be generated in python3 and run in its own python3 shell to create the homing sequence PLCs in the new copy of the motion directories.

The final step of the conversion process is to compare the homing PLCs created using motorhome.py with those created using pmac_motorhome; this is done with a call to the "diff" command with options to ignore blank lines and space changes. The result is then reported to the terminal indicating whether any comparisons or conversions failed, and provides a short, generated script to copy the new python3 configuration script to the original motion directory for the beamline or motion controller, if the user chooses to do so.

It is expected that not all conversions will be successful, however enough development has been done to ensure that about 85% of existing DLS motion homing directories will convert successfully, and of the remaining directories the majority of individual homing PLC files will be successfully converted. The remaining homing PLCs that were not converted can then be investigated manually, the expectation being that most will be custom PLCs or edge cases that will require some tweaking of the configuration script.

CONCLUSION

The pmac_motorhome package has been specified, designed and developed with the aim of improving the tool to generate critical motion homing routines at DLS, and other institutions that use Delta Tau motion controllers. The improvements include a more friendly and readable user interface, a better structured code base that is more maintainable and a testing framework with continuous integration built in. Documentation is generated in a web page format with explanations, how-to guides, tutorials and references; this will be beneficial to developers who want to add another defined homing routine or new snippets. Consideration has been given to the roll-out and adoption of the new tool, with effort put into removing any barriers to adoption and creating a conversion process that is as simple and confidence inspiring as possible for the controls engineers as the targeted users.

This tool provides the means to create customised homing sequences where required, and always be able to recreate it without manual intervention or editing. Finally, it will and allow better collaboration between motion and controls engineers through its more readable user interface and documentation.

It is hoped that pmac_motorhome may prove useful to other institutions using many Delta Tau motion controllers that require homing routines.