2. Positioner Speed Options

Configuration Options with Existing Firmware

Positioner motors move at 2 speeds, CREEP and CRUISE, with angular speeds (into the gearbox) of:

CREEP = 0.1 deg * 18 kHz / (CREEP_PERIOD=2) = 900 deg/s
CRUISE = 3.3 deg * 18 kHz = 9900 deg/s

The Namiki gearbox reduces this by (46/14 + 1)^4 = 337.36 to 2.667 deg/s (CREEP) and 176.07 deg/s (CRUISE).

The CRUISE speed is hardcoded in the firmware (lines 550, 702), but the integer parameter CREEP_PERIOD can be set independently for both axes (and both directions).

Therefore the main options are:

- change CREEP_PERIOD from 2 to 1 to double the CREEP speed.
- change some CRUISE moves to CREEP by modifying the hardware table sent via CANbus.
- change all CRUISE moves to CREEP for a period of time using ONLY_CREEP.

The main motivation for a change is to reduce the frequency of hard stop collisions at CRUISE speed. Any timing change also potentially affects on-sky efficiency.

Questions:

1. Are the software changes needed to implement these options confined to the plate_control package or are additional packages affected?
2. Which packages implement timeouts that might be triggered? petaalbox, petal, petalman, …?
3. What tests would we need to perform on a test petal before deploying in DESI?

Collisions

Collisions are suspected to contribute to a "linear theta/phi" failure mode that manifests like slippage in the gearbox.

Deliberate CRUISE collisions with hard stops mainly occur (20-Aug-2021 slides):

- when homing positioners in order to measure the hard stop angle or before performing a large-arc calibration sweep (~150 hits / device so far), and
- during the theta disambiguation step of the nightly FP setup (only 678 devices hit so far).

Accidental collisions with a hard stop or neighboring positioner can also occur during nightly operations.

Questions:

1. How important is it to avoid future collisions, i.e. how sure are we that they contribute to the failures?
2. Are some types of collisions worse than others? If theta disambiguation collisions are especially harmful, to what extent can they be minimized by tracking theta history?
3. How frequent are accidental collisions?

Operational Timing

During the blind and correction moves of a positioning loop, each device executes an independent sequence of angular moves, typically lasting 5-10s. Operationally, the longest individual sequence determines the time required.

The timing constraints during the nightly FP setup or engineering time are less stringent.

Questions:

1. what is the impact of each option on the positioning loop timing?
2. how much does the positioning loop timing contribute to overall on-sky efficiency?

Archived Slack Discussion

From Joe Silber in #focalplane 27-Aug-2021:

Joe Silber 4:51 PM

@dkirkby @kfanning @deisenstein I refreshed my memory on forci Yesterday only. They way I had implemented it is independent of any particular script or move command, whether disambig_theta.py or a homing call or whatever. Instead each positioner has a boolean value `'ONLY_CREEP'` . When set `True`, it does what it says. During construction of the move tables, each positioner will check this value to determine whether it is allowed to use cruise moves to get to the target faster, and then generate the tables accordingly.

In the slow + naive implementation, one could e.g.

1. set ONLY_CREEP = True for all positioners
2. run disambig_theta.py or whatever
3. restore ONLY_CREEP = False

But this would incur significant penalties on time for example in disambig theta, where most of the moves aren't actually gunning for hardstops.

We do have a batch setter function available. Calls in pecs typically look like:

```
pecs.ptlm.batch_set_posfid_val(settings, check_existing=True)
```

In this case, the dict `settings` would be like:

```
settings = {'M0001': {'ONLY_CREEP': False},
            'M0002': {'ONLY_CREEP': False},
            ...
           }
```

This same syntax, by the way, can set other values like:

```
{'CURR_SPIN_UP_DOWN': 100,
                'CURR_CRUISE': 100,
                'CURR_CREEP': 100,
                'CREEP_PERIOD': 2,
                'SPINUPDOWN_PERIOD': 12,
                'FINAL_CREEP_ON': True,
                'ANTIBACKLASH_ON': True,
                'ONLY_CREEP': False,
                'MIN_DIST_AT_CRUISE_SPEED': 180.0,
                'ALLOW_EXCEED_LIMITS': False,
                'BACKLASH': 3.0,
                'PRINCIPLE_HARDSTOP_CLEARANCE_T': 3.0,
                'PRINCIPLE_HARDSTOP_CLEARANCE_P': 3.0,
                'MOTOR_CCW_DIR_P': -1,
                'MOTOR_CCW_DIR_T': -1,
                }
```

Such settings are not sticky, in the sense that they live in memory only. They do not get pushed to the constants DB. So this is good --- upon restart of the instrument you'll be back to the defaults.

The function for batch setting values is in petal.py, line 1245:

```
    def batch_set_posfid_val(self, settings, check_existing=False, comment=''):
        """ Sets several values for several positioners or fiducials.
            INPUT:  settings ... dict (keyed by devid) of dicts {state key:value}
                                 same as output of batch_get_posfid_val
                    check_existing ... return None and make no changes if new value matches existing
                    comment ... add string to log_note associated with value, forces check_existing=True

            OUTPUT: accepts ... dict (keyed by devid) of dicts {state key:bool}
                                where the bool indicates if the value was accepted

            NOTE: no comments accepted - one cannot set keys in pc.require_comment_to_store
        """
```

Note that there is a similar function to batch get values:

```
    def batch_get_posfid_val(self, uniqueids, keys):
        """Retrives a batch of state values useful for calling from external scripts
           INPUT: uniqueids ... list, list of posids/fidids to retrive values for
                  keys ... list, state keys to retrieve
           OUTPUT: values ... dict (keyed by uniqueid) of dicts with keys keys
        """
```

So in summary, I think the most efficient and clean thing for `disambig_theta.py` would be to add a command line option where user can opt in/out of forcing creep only during disambiguation move. When desired, we:

1. Grab the initial states of ONLY_CREEP for all positioners.
2. Within the disambiguation move loop, turn on ONLY_CREEP for those positioners about to be joggled.
3. After that test move, restore ONLY_CREEP to original values.

In the case of `run_sequence.py`, the script that runs all the ecsv sequences, there is already an option to force ONLY_CREEP, but you have to do it in the ecsv file, and it is global to the whole sequence.

The simplest thing in that case would be to run for homing move a special 1 line sequence that just homes, while having that ONLY_CREEP parameter set to true.

(Done typing the long documentation string now 🙂

David Kirkby 5:11 PM

Thanks Joe!

Currently, ONLY_CREEP is False for all devices in the constants db. Do we expect that to ever change?

I guess we will still likely run into timeouts using ONLY_CREEP?

Joe Silber 5:14 PM

I think some of this stuff may get juggled around as we learn more about the bad positioners. For example there is a conceivable future where

- new firmware deemed too risky to deploy
- but with faster creep settings we can get bad positioners on target
- we tell the targeting folks they are allowed to use such positioners but have an additional constraint, for example to not go too far of a distance from target A to target B

One can get creative 🙂

Yes, I think the timeouts are likely. I remember escalating values like 45 sec, 60 sec, 120 sec at different layers in the overall system. I'm probably out of date but they are something like that, and scattered about in petalbox, petal, and petalman, etc. Everybody is defensive I guess, nobody wants to be the one who hangs in an infinite loop.

Cruise speed = 176 deg/sec vs creep = 2.67 deg/sec. So it's a huge difference.

Hm. Maybe I'm wrong. Theta disambiguation moves are typically around 40 deg, so going directly for it that's only 15 seconds. There might be some additional overhead, but maybe we don't timeout.

Homing we definitely would time out, since there the moves are like 500 deg or something, to 100% guarantee the robot covers the full range. (edited)

New

Joe Silber 6:47 PM

@dkirkby One more thing to note is that the existing FIPOS firmware allows us to change the `CREEP_PERIOD`. Default value = 2. If we change this to 1, the creeping will be twice as fast. I don't know the provenance where Henry and Irena decided on 2 as the best value. Assuming no physical problem with creeping twice as fast, then this will save lots of time. You can set the new `CREEP_PERIOD` in the same dict as referenced above, i.e. like:

```
settings = {'M0001': {'ONLY_CREEP': False, 'CREEP_PERIOD': 1},
            'M0002': {'ONLY_CREEP': False, 'CREEP_PERIOD': 1},
             ...
           }
```

The value is inherently an integer because it is getting updated on the 18 kHz intervals. So the existing firmware cannot creep any faster than that.

Joe Silber 6:47 PM

@dkirkby One more thing to note is that the existing FIPOS firmware allows us to change the `CREEP_PERIOD`. Default value = 2. If we change this to 1, the creeping will be twice as fast. I don't know the provenance where Henry and Irena decided on 2 as the best value. Assuming no physical problem with creeping twice as fast, then this will save lots of time. You can set the new `CREEP_PERIOD` in the same dict as referenced above, i.e. like:

```
settings = {'M0001': {'ONLY_CREEP': False, 'CREEP_PERIOD': 1},
            'M0002': {'ONLY_CREEP': False, 'CREEP_PERIOD': 1},
            ...
           }
```

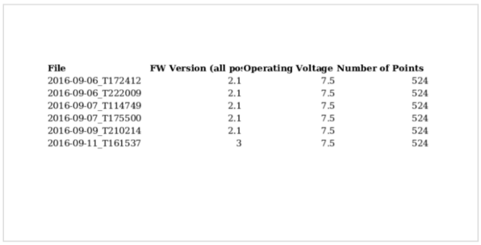
The value is inherently an integer because it is getting updated on the 18 kHz intervals. So the existing firmware cannot creep any faster than that.

Joe Silber 7:02 PM

I located the data Irena took back in 2017. She tested several motor parameter settings on three early positioners she had in hand. Attached an excel download of her google sheet here:

Excel Spreadsheet

Period_Current Parameter Testing.xlsx
17 kB Excel Spreadsheet

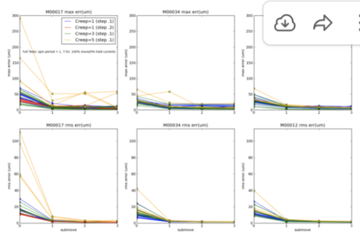


File	FW Version (all po:	Operating Voltage	Number of Points
2016-09-06_T172412	2.1	7.5	524
2016-09-06_T222009	2.1	7.5	524
2016-09-07_T114749	2.1	7.5	524
2016-09-07_T175500	2.1	7.5	524
2016-09-09_T210214	2.1	7.5	524
2016-09-11_T161537	3	7.5	524

According to these (limited) results, there is no accuracy cost to running at creep period = 1.

I also found this plot in the same google drive folder, from that same general test program. I believe these are results from an xytest with those same three positioners. Again creep was fine at minimum period (maximum speed)

errorvcreep.png



I know I'm dumping a lot of text on the channel, but I suspect we could win back a bit of survey time if we switch to this creep period of 1 in general across the board. Would want Henry to vet the idea.