LVV-T2190

March 2, 2022

1 MTAOS add aberrations to M1M3+M2+hexapod

This notebook is used for the level 3 integration tests from test plan LVV-P81 (https://jira.lsstcorp.org/secure/Tests.jspa#/testPlan/LVV-P81) as part of test cylce LVV-C176 (https://jira.lsstcorp.org/secure/Tests.jspa#/testCycle/LVV-C176). The following tests are currently run as part of this notebook:

- LVV-T2190 (https://jira.lsstcorp.org/secure/Tests.jspa#/testCase/LVV-T2190)

Execution steps are separated by horizontal lines. Upon completion, save the notebook and its output as a pdf file to be attached to the test execution in JIRA.

Last updated by E. Dennihy 20211020

Load all the needed libraries. Get the remotes ready Code in the notebook including section: "Check the summary state of each CSC".

```
[1]: %load_ext autoreload
     %autoreload 2
```

```
[2]: import rubin_jupyter_utils.lab.notebook as nb
     nb.utils.get_node()
```

```
/tmp/ipykernel_48638/1665379685.py:2: DeprecationWarning: Call to deprecated
function (or staticmethod) get_node. (Please use lsst.rsp.get_node())
  nb.utils.get_node()
```

```
[2]: 'yagan07'
```

```
[3]: import os
     import sys
     import asyncio
     import logging

     import pandas as pd
     import numpy as np

     from matplotlib import pyplot as plt
```

```python
from lsst.ts import salobj
from lsst.ts.observatory.control.maintel import MTCS, ComCam
from lsst.ts.observatory.control import RotType
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

```python
[4]: logging.basicConfig(format="%(name)s:%(message)s", level=logging.DEBUG)
```

```python
[5]: log = logging.getLogger("setup")
     log.level = logging.DEBUG
```

```python
[6]: domain = salobj.Domain()
```

```python
[7]: mtcs = MTCS(domain=domain, log=log)
     mtcs.set_rem_loglevel(40)
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

```python
[8]: await mtcs.start_task
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

```
[8]: [None, None, None, None, None, None, None, None, None, None]
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Ready M1M3: Raise mirror, turn on FB, clear forces

Need to have M1M3 LUT use its inclinometer.

Ready M2: Turn on FB, clear forces

Need to have M2 LUT use its inclinometer

Get camera hexapod ready: check config; make sure LUT is on, and has valid inputs; make sure hex is at LUT position

---

Get M2 hexapod ready: check config; make sure LUT is on, and has valid inputs; make sure hex is at LUT position

---

Slew to the next target. Choose a target such that the rotator stays within a couple of degrees of its initial position. This is because the CCW is not running (MTmount in simulation mode).

```
[11]: target = await mtcs.find_target(el=60, az=120, mag_limit=8)
      print(target)
```

```
HD   11064
```

```
[12]: await mtcs.slew_object(target, rot_type=RotType.PhysicalSky, rot=1.9)
```

```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

add 1um of z7 to the system via OFC

Compare the corrections sent vs forces and position changes applied. This is currently done in a separate notebook.

```
[13]: wavefront_errors = np.zeros(19)
```

```
[14]: wavefront_errors[3] += 1.0 # add1 um to z7
```

```
[15]: await mtcs.rem.mtaos.cmd_addAberration.set_start(wf=wavefront_errors,␣
      ↪timeout=10)
```

```
[15]: <ddsutil.MTAOS_ackcmd_fd03e870 at 0x7fb819c836d0>
```

This command primes the corrections, the issueCorrection command is needed to actually command them to be sent

```
[16]: await mtcs.rem.mtaos.cmd_issueCorrection.start(timeout=60.)
```

```
[16]: <ddsutil.MTAOS_ackcmd_fd03e870 at 0x7fb7c4e870a0>
```

Make plots using telemetry from each component to verify the changes in the DOFs. This step does not currently involve running any commands in this notebook. This step must be verified using a separate noteboook.

---

reset the corrections using the resetCorrection command

Compare the corrections sent vs forces and position changes applied (these are all expected to be zero). This is currently done in a separate notebook or on Chronograf.

```
[17]: await mtcs.rem.mtaos.cmd_resetCorrection.start()
```

```
[17]: <ddsutil.MTAOS_ackcmd_fd03e870 at 0x7fb81992c580>
```

```
[18]: await mtcs.rem.mtaos.cmd_issueCorrection.start(timeout=60.)
```

```
[18]: <ddsutil.MTAOS_ackcmd_fd03e870 at 0x7fb7b3c9dfd0>
```

---

add 2um of z7 to the system via OFC

Compare the corrections sent vs forces and position changes applied. This is currently done in a separate notebook or on Chronograf.

```
[19]: wavefront_errors[3] = 2.0 # add 2.0 um of z7
```

```
[20]: await mtcs.rem.mtaos.cmd_addAberration.set_start(wf=wavefront_errors,␣
      ↪timeout=10)
```

```
[20]: <ddsutil.MTAOS_ackcmd_fd03e870 at 0x7fb7b378bd90>
```

```
[21]: await mtcs.rem.mtaos.cmd_issueCorrection.start(timeout=60.)
```

```
[21]: <ddsutil.MTAOS_ackcmd_fd03e870 at 0x7fb7c4d2e670>
```

Stop Tracking

```
[22]: await mtcs.stop_tracking()
```

```
<IPython.core.display.HTML object>
```

Check that the corrections in step 10 are twice of those in step 7. This step does not currently involve running any commands in this notebook. This step must be verified using a separate noteboook.

Wrap up. Put each component to the following states: mtaos –> standby m1m3 –> lower mirror –> standby m2 –> standby camera hex –> standby m2 hex –> standby

```
[ ]: await mtcs.set_state(salobj.State.STANDBY, components=["mtaos"])
```

```
[ ]: await mtcs.lower_m1m3()
```

```
[ ]: await mtcs.set_state(salobj.State.STANDBY, components=["mtm1m3"])
```

```
[ ]: await mtcs.set_state(salobj.State.STANDBY, components=["mtm2"])
```

```
[ ]: await mtcs.set_state(salobj.State.STANDBY, components=["mthexapod_1"])
```

```
[ ]: await mtcs.set_state(salobj.State.STANDBY, components=["mthexapod_2"])
```

```
[ ]: await mtcs.standby()
```