KPIC BMC DM Control Notes

Contents

Section 1 – DM Setup (Start Sylvain's DM control script) 2

To Start Control Script

To Check that the Control Script is Running

To Kill the Control Script

Section 2 – DM Control from prtcserver: 4

To Open Sylvain's DM Surface Viewer

To Control the DM using KPIC DM Library from the prtcserver

Section 3 – DM Control from nfiuserver: 5

To Start the Socket Server

To Control via the Socket Server

To Check that the Socket Server is Running

To Kill the Socket Server

Section 4 – Kill Switch Control: 7

To Query the Kill Switch Status from Command Line

To Connect to the Kill Switch via Web Browser

To Reset the Kill Switch

To Connect to the Kill Switch via Telnet

GENERAL NOTES:

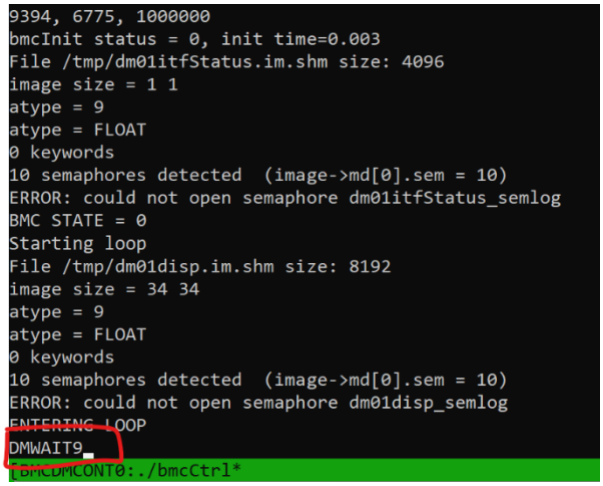
1. **The DM should not be operated above 32% humidity for extended time periods. A quick test for a few minutes is okay if truly necessary but should be avoided if possible.**

2. **When not in use, please zero the DM if possible. This can be done with the *DM.zeroAll()* or *DM.close()* functions.**

Section 1 – DM Setup (Start Sylvain's DM control script)

To Start Control Script:

1) Connect to k2ao server vnc:2:
    a) *k2aoserver-new.keck.hawaii.edu:2*
2) Open a local terminal:
    a) right click anywhere on the desktop
    b) select *Login Windows*
    c) select *xterm (local)*
3) In that terminal ssh to the prtc:
    a) ssh -X prtc@prtcserver
4) In that terminal, start a tmux session to hold the DM control script:
    a) tmux new -s BMCDMCONTROL
    b) You should now be in that tmux session
5) Navigate to the requisite directory:
    a) cd ~/dev/prtc/bmcDriver
6) Set some requisite system path variable:
    a) export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
7) Start Sylvain's DM control code script:
    a) ./bmcCtrl -L 27BW044#059
    b) If the script started successfully, it should print something like what's shown below. The key point is that the counter at the bottom (DMWAIT9) is incrementing.



To Check that the Control Script is Running:

Assuming you followed the procedure above, connect to the tmux session and see if it's running:

1. Connect to tmux:
    a. In a terminal: *tmux a -t BMCDMCONTROL*
    b. If the script is running, you should see the incrementing counter (see picture above)

If someone started the control script from elsewhere, you'll need to *ps aux* it to check for running instances elsewhere on the machine:

1. Check *ps aux*:
    a. In a termina: *ps -aux | grep bmcCtrl*
    b. If the control script is running, you will see an entry that says "./bmcCtrl -L 27BW044#059" as shown below. If you only get the "grep" entry (second output row in image below), then the control script isn't running.

To Kill the Control Script:

1. Connect to the control tmux: *tmux a -t BMCDMCONTROL*
2. Kill the control script using <ctrl-C>
    a. Note: this usually throws a seg-fault. Sylvain says this is okay.
3. Kill the tmux session using a regular *exit* command in its session.

(See next page for Section 2 on DM Control from prtcserver)

(Section 4 has details on the Kill Switch)

Section 2 – DM Control from prtcserver:

NOTE 1: Make sure to start the DM control first following Section 1 above.

NOTE 2: Sylvain's script just applies whatever is in the */tmp/dm01disp.im.shm* shared memory to the DM. So if you know how to work with his shared memories, you can control the DM surface by modifying that shm as done with other Sylvain shm's.

To Open Sylvain's DM Surface Viewer:

1. In any prtc terminal: *dm01Disp.py*

To Control the DM using the KPIC DM Library from the prtcserver:

1. (Start the DM control script following Section 1 above)
2. In a prtc terminal, navigate to the right directory: *cd /home/prtc/KPIC2/DM*
3. Start a python session: *python3*
4. In that python instance, instantiate the DM object:
    a. *from DM import DM as dmlib*
    b. *DM = dmlib()*
5. Example functions that are helpful:
    a. Set the DM flat: *flat = DM.setFlatSurf()*
    b. Set a specific surface: *surf = DM.setSurf(<surface_as_np_variable>)*
    c. Query the current surface: *surf = DM.getSurf()*
    d. Zero the DM surface (should be done when not in use): *DM.zeroAll()*
    e. Apply a Zernike (focus with 0.1 amplitude here): *surf = DM.pokeZernike(0.1, 4)*
    f. Get help on a function: *help(DM.<function_name>)*

```
Help on method pokeSin in module DM:

pokeSin(amplitude, frequency=1, angle=0, phase=0, bias=None) method of DM.DM instance
    Poke actuators in a sin pattern

    Arguments:
      amplitude         - poke amplitude RELATIVE TO THE BIAS (peak to 0)
                          -- Range: 0 to 1
      frequency         - frequency of sinusoid in cycles across pupil
      angle             - angle of sinusoid
      phase             - phase of sinusoid
      bias              - (optional) surface over which grid should be applied

    Returns:
        The surface that was applied
(END)
```

    g. See, and get help on, all available functions: *help(DM)*

```
class DM(builtins.object)
 |  Methods defined here:
 |
 |  __init__(self)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  addBMCAct(self, amplitude, bmc_ind)
 |      Same as pokeBMCAct() except it adds the desired grid to the current DM state
 |
 |  addCols(self, amplitude, col_ind)
 |      Same as pokeCols() except it adds the poke to the current DM state
 |
 |  addCrosshair(self, amplitude, width=None)
 |      Same as pokeCrosshair() except it adds the crosshair to the current DM state
 |
 |  addGrid(self, amplitude, row_ind, col_ind, grid_spacing)
 |      Same as pokeGrid() except it adds the desired grid to the current DM state
 |
 |  addRows(self, amplitude, row_ind)
 |      Same as pokeRows() except it adds the poke to the current DM state
 |
 |  addZernike(self, amplitude, Noll)
 |      Same as pokeZernike() except it adds the desired Zernike to the current DM state
 |
 |  close(self)
 |      Alias to zeroAll()
 |
 |  flashEdgeActs(self, amplitude=-0.2, pausetime=0.3, Nitr=20, bias=None, verbose=True)
 |      Flash edge actuators to easily identify them
 |      - This will alternate between poking one half of the edge actuators
 |        and poking the other half so that the edges are identifiable.
 |      - Uses pokeEdgeActs()
 |
 |      Arguments (ALL OPTIONAL):
 |        amplitude      -poke amplitude RELATIVE TO THE BIAS
 |                        -- Range: 0 to 1  --- default -0.2
 |        pausetime      - Time between alternating pokes (speed of flash)
:
```

Section 3 – DM Control from nfiuserver:

NOTE: Make sure to start the DM control first following Section 1.

To Start the socket server:

1. (Start the DM control script following Section 1 above)
2. **In a prtcserver terminal**, navigate to the right directory: *cd /home/prtc/KPIC2/DM*
3. Start a tmux for the socket server: *tmux new -s BMCDMSOCKET*
4. In that tmux, start the socket: *python3 DM_Socket_Server.py*
    a. If the script started successfully, it should print something like what's shown below. The key point is the "Waiting for connection" line.

```
prtc@prtcserver:~/KPIC2/DM$ python3 DM_Socket_Server.py
WARNING: AstropyDeprecationWarning: astropy.utils.compat.funcsigs is now deprecated - use inspect instead [astropy.utils
.compat.funcsigs]
Starting Socket
Listening on 10.136.1.43:30000
Waiting for connection

[BMCDMSOCK0:python3*                                                    "prtcserver" 16:48 04-Apr-22
```

    b. Note: you should be able to just leave this running indefinitely, just restart it when you reboot the server.

To Control via the socket server:

1. (Start the socket server following the steps in "Start the socket server" above)
2. **In an nfiuserver terminal**, navigate to the right directory: *cd /home/nfiudev/dev*
3. Start a kpython3 instance: *kpython3*
4. In that kpython3 instance, instantiate the DM object:
    a. *from DM_Sock import DM as dmlib*
    b. *DM = dmlib()*
5. Example functions that are helpful:
    a. Set the DM flat: *flat = DM.setFlatSurf()*
    b. Set a specific surface: *surf = DM.setSurf(<surface_as_np_variable>)*
    c. Query the current surface: *surf = DM.getSurf()*
    d. Zero the DM surface (should be done when not in use): *DM.zeroAll()*
    e. Apply a Zernike (focus with 0.1 amplitude here): *surf = DM.pokeZernike(0.1, 4)*
    f. Get help on a function: *help(DM.<function_name>)*

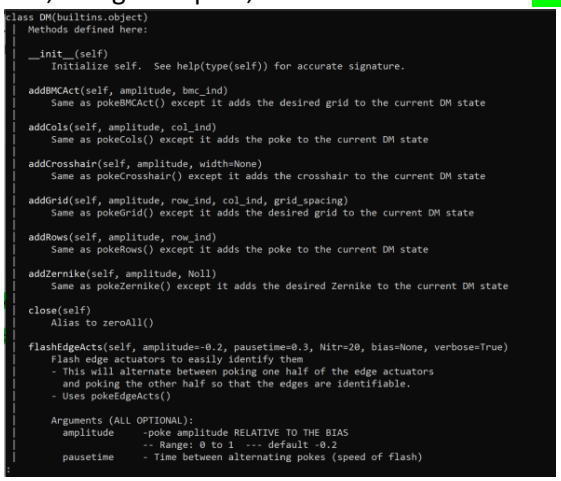
```
Help on method pokeSin in module DM:

pokeSin(amplitude, frequency=1, angle=0, phase=0, bias=None) method of DM.DM instance
    Poke actuators in a sin pattern

    Arguments:
      amplitude         - poke amplitude RELATIVE TO THE BIAS (peak to 0)
                          -- Range: 0 to 1
      frequency         - frequency of sinusoid in cycles across pupil
      angle             - angle of sinusoid
      phase             - phase of sinusoid
      bias              - (optional) surface over which grid should be applied

    Returns:
        The surface that was applied
(END)
```

g. See, and get help on, all available functions: *help(DM)*



To Check that the Socket Server is Running:

Assuming you followed the procedure above, connect to the tmux session and see if it's running:

2. Connect to tmux:
    a. In a terminal: *tmux a -t BMCDMSOCKET*
    b. If the script is running, you should see "Waiting for connection line".

If someone started the control script from elsewhere, you'll need to *ps aux* it to check for running instances elsewhere on the machine:

2. Check *ps aux*:
    a. In a termina: *ps -aux | grep DM_Socket_Server*
    b. If the control script is running, you will see an entry that says "python3 DM_Socket_Server.py" as shown below. If you only get the "grep" entry (second row below), then the control script isn't running.

```
prtc@prtcserver:~/KPIC2/DM$ ps -aux | grep DM_Socket_Server
prtc      152109  0.1  0.0 535028 128280 pts/26  S+   16:43   0:01 python3 DM_Socket_Server.py
prtc      153939  0.0  0.0  12948  1084 pts/19   S+   16:54   0:00 grep --color=auto DM_Socket_Server
```

To Kill the Control Script:

4. Connect to the control tmux: *tmux a -t BMCDMSOCKET*
5. Kill the control script using <ctrl-C>
6. Kill the tmux session using a regular *exit* command in its session.

Section 4 – Kill Switch Control:

NOTE 1: This section is likely to change soon once we make the Kill Switch a keyword.

NOTE 2: The Kill switch is set to trigger at 32% humidity and it is a rising-edge latching trigger. As such, once it triggers, you'll need to reset it to get power back to the DM. Since this is a rising-edge trigger, only send the reset command if you are comfortable with the current humidity level otherwise you could end up operating the DM above the goal humidity.

To Query the Kill Switch Status from Command Line:

1. In a nfiuserver terminal: *kpython3 ~/dev/Kill_Switch.py -THDA*
    a. Sample result:

```
[nfiudev@nfiuserver ~]$ kpython3 ~/dev/Kill_Switch.py -THDA
Alarm: Off (DM Powered) | Humidity: 5.6% (RH%) | Temperature: 1.8 (C) | Dew point: -32.9 (C)
```

To Connect to the Kill Switch via Web Browser:

1. Open a browser and navigate to: *10.136.1.100*
2. When prompted, enter the password: *12345678*
3. You can use the "Device Query" section to query some values:
    a. To query humidity: **X01*
    b. To query temperature: **X02*
    c. To query dew point: **X03*
    d. To query alarm status: **U01*
        i. A response of **U01@* means the alarm is not set and hence the DM is powered
        ii. A response of **U01A* means the alarm is set and hence the DM is powered

To Reset the Kill Switch:

1. (Connect to web browser following steps in "To Connect to the Kill Switch via Web Browser" section above)
2. Go to the "Device Query" section on the left.
3. Send the reset command: **Z02*
4. (Optional) confirm that the alarm is reset. Send the alarm query command: **U01*
    a. A response of **U01@* means the alarm is not set and hence the DM is powered
    b. A response of **U01A* means the alarm is set and hence the DM is powered

To Connect to the Kill Switch via Telnet:

NOTE: the telnet connection will timeout fairly quickly so if you send a command and it says "Connection closed by foreign host", just reconnect and try your command again.

1. In a terminal: *telnet 10.136.1.100 2000*
2. Send commands as usual (see above for command options).