Tips & Tricks: Compiling and Running KiteFAST

Notes by Rick Damiani of RRD Engineering LLC, April 2020.

Table of Contents

Introduction

The following is a collection of tips compiled by RRD Engineering over the course of the KiteFAST development. See the report “Makani Final Thoughts, Project WE-201904”, April, 2020, by Rick Damiani for a full summary of the project findings and recommendations. Note that these Tips & Tricks do not constitute a thorough guide for getting up and running with KiteFAST on the Linux Debian “Stretch” operating system, but rather catalogue the experience of getting things running on a particular machine. This document should provide useful hints for an experienced linux user.

Compiling/recompiling KiteFAST

So there is a possibility that I will need to recompile, via make, parts of the code.

KiteFastMBD.cc (abbreviated KFM.cc) or its offshore version KiteFastMBD-os.cc must be compiled “with” MBDYN. The way this is done, is simply configuring MBDyn to look for a module-kitefastmbd (or [...]-os for offshore). The way to do this is in the install script: [-]\sandbox\glue-codes\kitefast\ kitefast_install.sh (look for configure).

However, MBDYN will need the shared object .so (not a static .a), which means we can just **recompile the user module without rebuilding mbdyn**. To do so,

```
cd $mbdyn_directory/modules

make clean && sudo make install  # build the user defined element
```

Note that the “make install” will both make and install the .so under /usr/local/mbdyn/libexec

Note that this may not work on linux for **errors related to aclocal or libtool**.

```
What I had to do was:

1. Change am__api_version='1.15' from ‘1.14’ in $mbdyn_directory/configure

2. in $mbdyn_directory run >>sudo autoreconf -fi
```

> Also note that **if you edit files from windows you may get an issue with line 1: $'\r**': command not found
>
> So from Debian WSL >>> **dos2unix** on the files edited will fix that
>
> Also **use WSL>> code <file> to edit with WSL code rather than windows VScode**. The terminal in code is a nicely formatted debian terminal at that point too.
>
> I have a new script to build KiteFAST portion (including user element install in the right place): RRD_KiteFASTLib_install.sh (under ~/sandbox/glue-codes/kitefast)

Note: One problem I would like to see resolved is the last time step failure message :

>Simulation diverged after 12 iterations, before reaching max iteration number 1000 during Step=60001, Time=60.001; TimeStep=0.001 cannot be reduced further; aborting...

An error occurred during the execution of MBDyn; aborting...

To compile a single module of FAST (kitefast) do the following under sandbox:

>mkdir build

```
>cd build

> cmake .. -DDOUBLE_PRECISION=OFF

> cd sandbox/build/modules-local/kitefast-controller

> make (optional “make install” would create an install dir with bins and libs and the proper files)
```

The files are created in the local (e.g. build/kitefast-controller) directory, in particular libkitefastcontrollerlib.a which is symbol-linked from the module-kitefastmbd directory under mbdyn.

When I modify any module in KFAST I need to relink them in the usr-module in mbdyn. For that, use the instructions above.

Sign conventions/indices for KiteFAST and CSIM

The index of the motors is (looking from tail toward nose) **according to KiteFAST**:

6	4	|	0	2	positive, or about kite x
7	5	|	1	3	negative (along –x)

The index of the motors is (looking from tail toward nose) **according to CSIM**:

5	6	|	7	8	positive, or about kite x
4	3	|	2	1	negative (along –x)

The index of the flaps is (looking from tail toward nose) **according to KiteFAST**:

A1	A2	A4	|	A5	A7	A8	sign in KiteFAST is only in the aerotables!

Rudder: Positive (T.E. to port) | sign in FAST is only in aerotables

Elevator: Positive if its Trailing Edge is up sign in FAST is only in aerotables

The index of the flaps is (looking from tail toward nose) **according to CSIM**:

A1 A2 A4 | A5 A7 A8 **positive, if Trailing Edge DOWN**

Rudder: Positive (T.E. to port) | negative

Elevator: Positive if its Trailing Edge is DOWN _____-_____

+

_____

Setting up the KiteFAST controller

The following inputs to the controller are set in KiteFastSubs subroutine AssRes_OnShore

```
m%KFC%u%tether_forceb = matmul(OtherSt%FusODCM, OtherSt%totalFairLeadLoads)
   m%KFC%u%dcm_g2b     = matmul(OtherSt%FusODCM, transpose(p%DCM_Fast2Ctrl))
   m%KFC%u%pqr         = matmul(OtherSt%FusODCM, OtherSt%FusOomegas)
   m%KFC%u%acc_norm    = TwoNorm(OtherSt%FusOacc)
   m%KFC%u%Xg          = OtherSt%FusO - AnchorPt
   m%KFC%u%Xg          = matmul(p%DCM_Fast2Ctrl, m%KFC%u%Xg)
   m%KFC%u%Vg          = matmul(p%DCM_Fast2Ctrl, OtherSt%FusOv)
   m%KFC%u%Vb          = matmul(OtherSt%FusODCM, OtherSt%FusOv)
   m%KFC%u%Ag          = matmul(p%DCM_Fast2Ctrl, OtherSt%FusOacc)
   m%KFC%u%Ab          = matmul(OtherSt%FusODCM, OtherSt%FusOacc)
   m%KFC%u%rho         = p%AirDens

   m%KFC%u%apparent_wind = IfW_FusO - OtherSt%FusOv
   m%KFC%u%wind_g      = matmul(p%DCM_Fast2Ctrl, IfW_ground)
```

```
m%KFC%u%apparent_wind = matmul(p%DCM_Fast2Ctrl, m%KFC%u%apparent_wind)
```

Note that: OtherSt for KFAST is under Global Data and declared as public, so it is available everywhere

```
type(KFAST_OtherStateType), save :: OtherSt
```

FusODCM is to go from global to local(kite)

DCM_Fast2Ctrl is to go from FAST global to CTRL global local

_____

MD: One thing that is annoying is that MD input file requires vessel1 (not just vessel) even if there is only one vessel.

A Few Notes on Running CSIM, Makani's M600 simulation tool

CSIM is fully documented under ~/makani. The notes below relate specifically to extracting output from CSIM to compare with KiteFAST output in a Jupyter notebook (see next section)

Note: Tobin helped me with setting up and compiling the basic C-code with Scripts under ~/makani/lib/scripts/install (I think it was an .sh), lib/scripts/install/install_packages.sh

then i needed to get some extra permissions done by IT, and then I had to have multicast set up as well. This was under /etc/network/interfaces and Jerome made me add a couple of lines that enabled opening up several monitoring windows while the simulator is running.

Note: Anything relating to the kite configuration that might need to be changed is in the *config* folder.

run_sim is the main command, and --help provides a list of help topics

run_sim actually recompiles a bunch of python scripts that contain input info. These input files are under *config*

Views can be changed with the mouse (left click) and/or with 1,2-xx

run_sim uses *hdf5* files as turbsim files (actual turbsim files from TurbSim). It also stores hdf5 files, which are made after the .pkap files are created. SO DON'T double ctrl+c to stop a sim, just once, or you will kill the transformation from pkap to hdf5.

To read/postprocess output hdf5 files, there are scripts in matlab under a*nalysis/plot/mabraham* or *python.*

To run run_sim:

1. you can recompile new input each time (going into **config** and changing base_station for buoy stuff, or **common** (for stuff that applies to every kite, both m600 and octoberkite), for example for wind stuff, you go to **configure/common/sim/phys/wind**

2.use override flag "-o" and then you need '{....}' dictionary of things that must be overridden, with the entire path as in common/sim/phys_sim/wind...

There is some documentation on the shared G-drive, which I do not have access to, called gomakani-turbsim for wind files

Basically, they smartly have run a bunch of wind files, for all conditions and TI etc, so that they can pick among those to run a specific case. When run_sim runs, a turbsim wind file is moved to the local /tmp from a remote server.

T**o save (log) data you must specify "-l"** and this will create pcap and then **hdf5** files. When the hdf5 file is written the pcap can be deleted. Note hdf5 files are always linked to with a lasthdf5 or so, so that you can use that name if you pay attention, else use the full name with date/time in the title.

To sa**ve state and then rerun beginning with the saved state, do the following:**

>>run_sim --save_state_time 300 (for save at 300s the states)

then kill the run and then rerun

>>run_sim --load_state_time

To review telemetry monitors: >>**run_sim -S -M <monitor_name>**

where, for example, monitor_name is flight/crosswind/debug/ and there are also others

Structure of output channels: c (ctrl telemetry) → state_est

s(sim telemetry, AFTER estimator) → so this is likely the one to use to compare to KiteFAST

"--nowith_online_turbsim_databases" seems to be another option to try and get the wind to follow what i want and bypass the default. So far no success.

'webmonitor sim' will start the webmonitors for the sim

--sim_time argument to run_sim with a certain time perhaps --folks have not been very sure of these options

run_sim --nowith_online_turbsim_databases -S -M M seems to be the one that worked to bypass the default. However, the wind direction is set at 40 deg and test_site is parker_ranch. How can I bypass the test_site? Where is test_site set?

I ended up modifying ParkerRanch in phys_sim.py, because it just did not work for me otherwise. So I changed the wind_direction in there (set it to 0 deg), and then used the above command.

Also I have used the following that works well:

>**run_sim --nowith_online_turbsim_databases -S -M M -t 600 --save_state_time 480 -l**

This runs for 600s and saves states at 480s and saves a log (-l)

then run

>**run_sim --nowith_online_turbsim_databases -S -M flight -t 580 --load_state -l**

to run 100 seconds (580-480s)

E.g., for kite with hi-start: This saves new states at 480s

>> **run_sim --nowith_online_turbsim_databases -S -M M -t 490 --save_state_time 480**

then I save the CSIM run starting there and lasting 47 s.

>> run_sim --nowith_online_turbsim_databases -S -M flight -t 527 --load_state -l

The above approach renders a bit of transient for CSIM. So I cut off the first 7 seconds of output in the Jupyter notebook, and start the data comparison at time 487s.

Post-processing w/Jupyter Notebooks

Under RRDscripts, I have jupyter notebooks.

Under makani run the following:

>**bazel run lib/bazel:jupyter ~/RRDscripts/KFASTvsCSIM.ipynb**

The problem with this jupyter is that I cannot change the kernel to my conda, and not sure there is a way. so it does not really work for me.

Jerome has recommended the following:

The procedure is: 1- bazel build //lib/bazel:pyembed 2- bazel-bin/lib/bazel/pyembed jupyter notebook (or whatever command you use to open your python IDE) 3- Run the plotting script.

I managed to compile a new version of KiteFASTcontroller.f90 with IO and new registry in debug/release in IVF, but only after changing Library Debug DLL in the properties (under Fortran).

It is working pretty well, and I think I will be able to get an input file with ICs and output to have everything I want back to c-controller to check against CSIM. (remember upper cases for channels in fast) --out for 3 days now--

Still having problems with linux make

Note that to restore a single file in git: you need to

>>> git fetch --all

>>> git checkout origin/dev-offshore -- KiteFASTController.vfproj (so no path to file)

Installing/Running Blender 2.8

For blender 2.8

First: install anaconda3

then make an environment with the same python 3.7.x as blender

then install all blendyn's packages with ~/anaconda3/bin/pip3 instead of pip, and **from within** the env created. Also remember to check auto python script in blender (under file).

then go to **blender's dir and inside 2.80 you will find python, mv to python_old and create a soft link** to the new env dir instead (under anaconda3/envs)

You might then have issues with libz.1.2.9 or so (https://stackoverflow.com/questions/48306849/lib-x86-64-linux-gnu-libz-so-1-version-zlib-1-2-9-not-found) . google it up, you need to download the correct package, ./configure, make and then copy the libz file to somewhere (right now it is on my desktop, but the instructions say /lib/x[...] and then move the link that is in there to an "_old" and then point to the version just copied.) That can generate problems because compiled .mod files are created through a zip process. So make sure that only blender points to this libz folder, and that it is not under /usr/local

Another problem might have been cairo

processing triggers for libc-bin

ldconfig: /lib/x86_64-linux-gnu/libcairo.so.2 is not a symbolic link

ldconfig: /lib/x86_64-linux-gnu/libcairo-script-interpreter.so.2 is not a symbolic link

ldconfig: /lib/x86_64-linux-gnu/libcairo-gobject.so.2 is not a symbolic link

This is what I ended up doing: in the anaconda python env that was dedicated to blender, I replaced the symlink: in ~/anaconda3/envs/py370/lib

I replaced **libz.so.1 -> libz.so.1.2.11 with libz.so.1 -> libz.so.1.2.9 (I had the libz.so.1.2.9 available already and copied it under). Then I executed (from the activated env) the following export :**

rdamiani@kitefast: ~/anaconda3/envs/py370/lib$ **export LD_LIBRARY_PATH=$PWD (per https://stackoverflow.com/questions/48306849/lib-x86-64-linux-gnu-libz-so-1-version-zlib-1-2-9-not-found) I am not sure this will keep working, but it let me activate blendyn without errors.**

>>>>I replaced the symlink to its original one now, and things seem to be working.

It quit working after quitting, so I created a script blender.sh as follows:

```
source ~/anaconda3/etc/profile.d/conda.sh

conda activate py370

cd ~/anaconda3/envs/py370/lib

export LD_LIBRARY_PATH=$PWD

blender &
```

and I think this could become a shortcut somewhere in the main screen

Blender tips for viz:

To have the **camera follow the aircraft** do the following:

1. Select a representative node (MIP) and then Create an empty with Shift+A add empty

2. Move the empty to the side of the aircraft (hit “g” then move)

3. Select the empty then shift+select the MIP node then Ctrl+P and parent to object, this will make the empty follow the aircraft

4. Move the camera close to the empty and select camera then Shift+select the MIP and then Object (at the bottom) track-to, this will point the camera at the plane

5. Now select camera then Shift+select empty and then Ctrl+P and parent to object, this will link the camera to the movement of the empty

5. Select the empty, then Shift+S and send cursor to selection

6. Select camera, then Shift+S and send the select to cursor, this will put the camera on the empty

7. Ctrl-Alt-Numpad0 sets the camera view to the viewport, so select a view and then Ctrl-Alt-Numpad0

TO **slow playback:**

> set a ratio of 1:9 or so, by going to scene (on the right panel, the printer looking icon) and then remapping say 100 to 900, or reduce 100 as 900 is the max, say 10 to 100 for 1:10), then in the timeline at the bottom, multiply the last time/keyframe by the same factor.

To add **selig curves**

Select beam, then the curve line on the main panel on the right, scroll down to curve selection

To remove dashed lines:

Click on main view screen, then press "n", go to display and unclick relationship lines

To do a quick animation movie

1. Click the camera icon in the main panel on the right, then Set AV JPEG under Output, and then set the output folder to the dir of interest

2. Press Alt+F3, then capture button at the top, then start animation, then stop it and press capture button

To show motion paths

1. select the view you want the path shown on

2. select the cube icon in the main panel on the right

3. go to motion paths

Scripting and Add on (see also https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Advanced_Tutorials/Python_Scripting/Addon_Custom_Property)

1. This is my first Add-on to move nodes to a different layers : MyPanel4Blender.Py

2. I need to open up a text editor window in Blender: first split the view by dragging a corner,

then next to "view" select text editor, and then open text block and navigate to the current dir

3. At the top, if you drag down the window edge you should be able to get the "info" window, if not there with the leftmost selector (next to view) select "info"

4. a lot of info is shown in the info box

5. Note that i cannot seem able to save he add-on for whatever reason, so need to load it in via text editor and run it

---

12/10: Today we fixed the issue related to libz, I commented above that it cannot be under /urs/local

Also Raf created for me links to ./libs within the mbdyn/module-kitefastmbd and -os dirs; this way I do not have to re-install with make install any time. Still do make clean and sudo make within the modules dir per my updated script

---

If you have problems with the cinnamon panels, follow this link: [SOLVED] Menu and desktop items unavailable - Linux Mint Forums

---

Debugging in VSCODE

I have found out how to do **debugging in VSCODE**!!! I can likely step through everything, which is an amazing feat.

Create a task.json file from the template in vscode after ctrl+P search for task and it should come up.

In there put {

```
// See https://go.microsoft.com/fwlink/?LinkId=733558
// for the documentation about the tasks.json format
"version": "2.0.0",
"tasks": [
{//This is to build KiteFAST in debug mode
"label": "Kitefast_dbg_build",
"type": "shell",
"command": "/home/rdamiani/sandbox/glue-codes/kitefast/RRD_KiteFASTLib_dbg_install.sh",
"args": [] ,
```

```
"options":{"sudo"},
"runOptions": {"sudo"}
}
]
}
```

Then make a config that looks like this:

```
{ "preLaunchTask": "Kitefast_dbg_build",
"name": "GDB-Debug Kitefast",
"type": "cppdbg",
"request": "launch",
"program": "/usr/local/mbdyn/bin/mbdyn",
"args": ["/home/rdamiani/sandbox/glue-codes/kitefast/test_cases/RRD_m600_landbased/KiteMain.mbd"],
//"printCalls": false,
//"showDevDebugOutput": false,
//"valuesFormatting": "prettyPrinters",
//"gdbpath": "gdb",
//"target": "/usr/local/mbdyn/bin/mbdyn",
//"preLaunchTask": "/usr/bin/gfortran",
"cwd": "${workspaceFolder}/glue-codes/kitefast/test_cases/RRD_m600_landbased"
}
```

this uses a special install script (dbg) that has special flags in CMAKE to do the debug.

Note when I compile in debug mode, then even if I relaunch ./RRD_KiteFASTLib_install.sh, it will still be slow. TO avoid that, you need to recreate the build dir, i.e. wipe its content first, then relaunch the script : ./RRD_KiteFASTLib_install.sh.

For Debug what I usually do is: in VScode, ctrl+shift+P then run task Kitefast_dbg_build

then I click on the ladybug, and then from the dropdown I pick the config that has (in the launch.json) the right working dir; I have to comment out the pre-step (the one I run manually above) because somehow they run in parallel otherwise.

Building the KiteFAST documentation

The web-based KiteFAST documentation is under ../**sandbox/docs/src/user** (source) and needs to be built in order to access it. To build:

cd to **sandbox/build**

cmake .. -DBUILD_DOCUMENTATION=ON or use ccmake .. and then turn on flags in there

make sphynx-html

the documents end up in **../build/docs/html**

to open, either type **open docs/html/index.html** or

To access from the browser, open a Chrome (e.g.) page and type **ctrl-o**. Navigate to **../docs/html** and select **index.html**.
