Features in the Quad State Space Model

(Excerpt from G1401132-v8)

Contents

- Feature summary
- Where in the SVN does the model live?
- Hierarchical Structure
- Calling the model
- Model Output

Send questions or comments to Edgard (edgard [at] stanford.edu)
and/or Brett (shapirob [at] stanford.edu)

Feature Summary

- **Built in features** – always included in the model
  - Relative SUS/cage sensors and actuators – new as of 1 Jan 2016
  - Suspension point reaction forces – new as of 1 Jan 2016

- **Optional features** – your choice to include or not
  - Two chain model – includes both a main and reaction chain
  - Violin modes
  - Damping
    - Import live damping from sites
    - Import damping from a prior GPS time from sites
    - Include optical lever damping (PUM and UIM actuation)
    - Load damping from a saved filter file or command space variable
    - PUM Pitch damping from PUM OSEMs.
  - UIM length force input filter (accounts for unmodeled dynamics).
  - SUS point displacement to TOP drive feedforward.

- **Possible future features** – there is always room for improvement
  - Global control
  - Radiation pressure

Note: Graphical Simulink representations of the model layouts are cited in step 1 of the '3 steps to edit the model features' section. These will provide more detail for some of the descriptions below.

Where in the SVN does the model live?

The model files are found in

`…/SusSVN/sus/trunk/QUAD/Common/MatlabTools/QuadModel_Production/`

The model is compiled with the function

`generate_QUAD_Model_Production.m`

The supporting files called by this script are:

a) Simulink layout files:

```
generate_QUAD_SingleChainUndamped_Simulink.slx
generate_QUAD_SingleChainDamped_Simulink.slx
generate_QUAD_BothChainsUndamped_Simulink.slx
generate_QUAD_BothChainsDamped_Simulink.slx
```

b) Import damping filters from the sites:

```
M0LiveDampingFilters.mdl -> imports main chain damping from sites
R0LiveDampingFilters.mdl -> imports reaction chain damping from sites
L1L2_OplevLiveDampingFilters.mdl -> imports oplev damping from sites
```

Where in the SVN does the model live?

c) Adding violin modes:

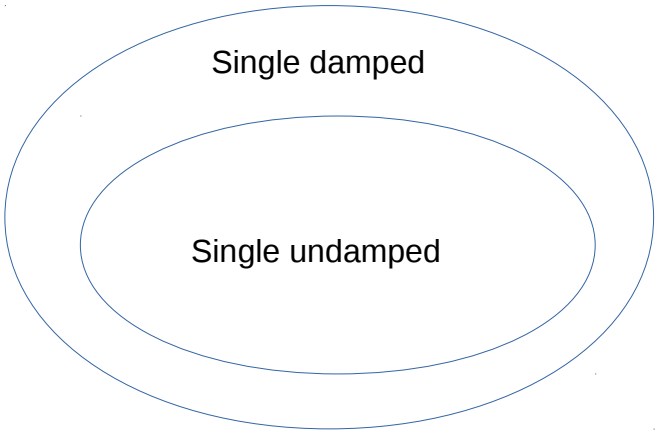
```
makequad_with_modal_fibers.m  -> PUM-TST violin modes
makequad_with_modal_uimpum_wires -> UIM-PUM violin modes
makequad_with_modal_topuim_wires -> TOP-UIM violin modes
makequad_with_modal_top_wires.m -> SUS-TOP violin modes
```

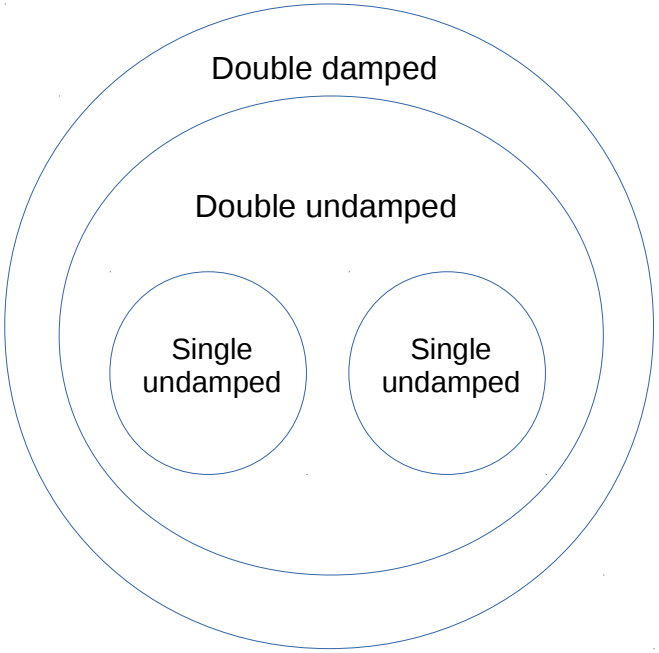
d) Simulink diagrams and user interface management:

```
In_Out_Parser.m -> parses the input and output indices from simulink
QUAD_Model_input_options_template.m -> Template to call the model
ExtractUserOptions -> Manages the backwards compatible call to the script
UndampedChain.m -> Creates a class that compiles the undamped chain model
```

Hierarchical Structure

The models are built in a hierarchical way. More complicated models are built around simpler ones. If one of the ‘inner’ models gets modified the changes propagate to the ‘outer’ models:





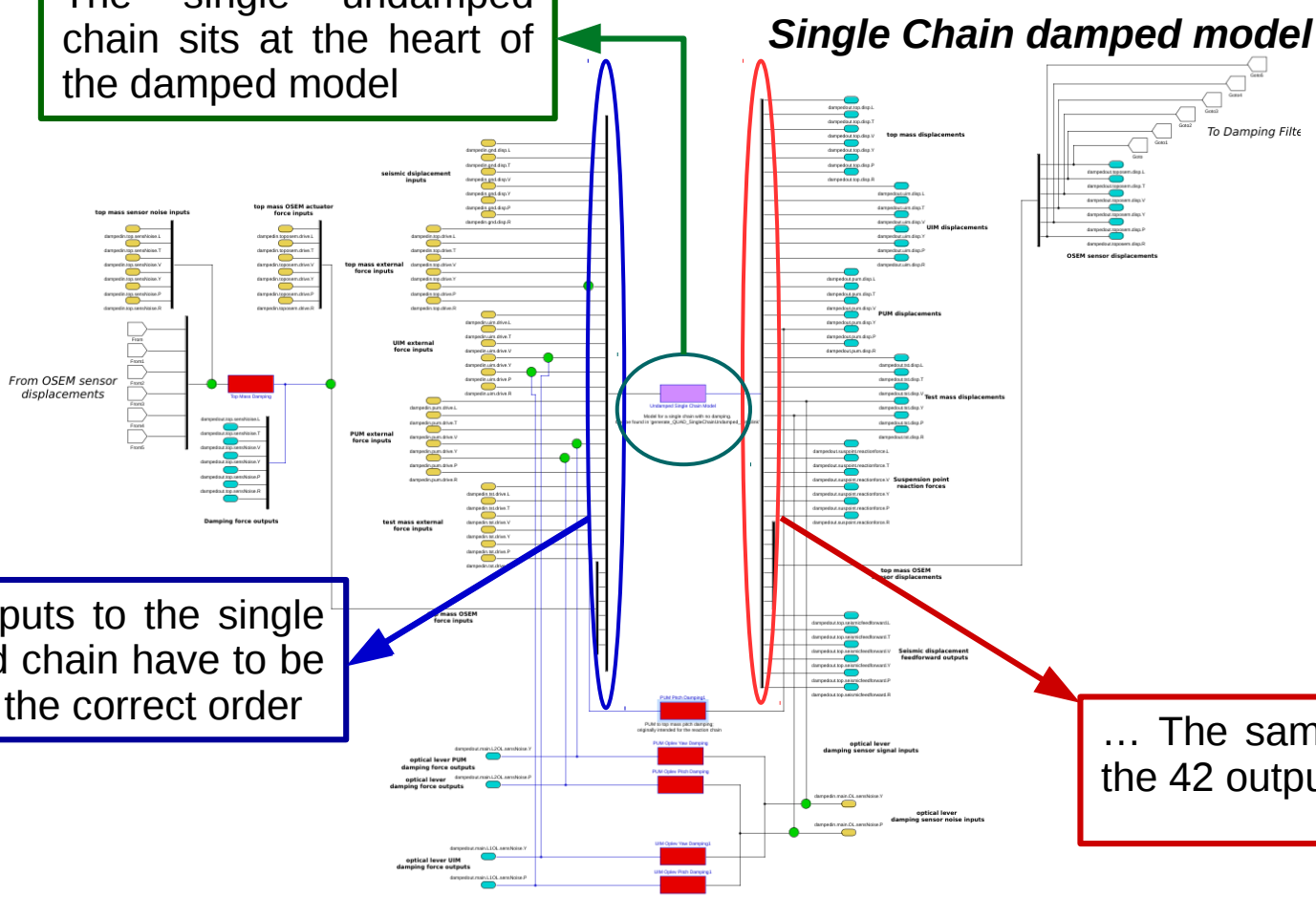
By modifying the single undamped chain, we modify every single model. Sometimes the outer diagrams have to be modified to account for changes in the inner ones. *(see step 3 for more information)*

Filters and features for the single undamped chain must be included and handled through the **UndampedChain.m** class script.

Hierarchical Structure

If the inputs and outputs are modified, one would need to make the appropriate changes in the ‘outer’ simulink diagrams:

The single undamped chain sits at the heart of the damped model



The 36 inputs to the single undamped chain have to be fed to it in the correct order

… The same applies for the 42 outputs.

Calling the Model

1. The model can be called with no inputs, which gives the default build:

```
>> generate_QUAD_Model_Production()
You have chosen to construct the default QUAD Model
Building a single chain model for a fiber QUAD suspension ...
```

This call creates the default quadruple suspension model:

- Single chain fiber build (No reaction chain).
- No feedforward filters or feedback loops are applied.
- Does not include any of the UIM Mysterious Dynamics.
- No violin modes are applied.

The Options struct call modifies the default settings with the user defined options.

(See Next Slide)

Calling the Model

2. The model runs by using an 'options' struct to modify the default behavior:

Detailed instructions for the model call, and a template for managing the features can be found in: `QUAD_Model_input_options_template.m`.

```
>> options=struct();
>> options.singleChainBuildType='fiber';
>> options.reacBuildType=[];
>> options.topMassDamping = 'default';
>> options.violinModes.fiber= 4;
>> quadModel = generate_QUAD_Model_Production(options)
Options struct accepted as input
```

Create an options struct and fill the fields with the options you want

Call the script using this struct as input

Options:
- Single chain fiber QUAD suspension
- The default top mass damping filters are applied
- The first 4 violin modes between test mass and PUM are included

Any option not set manually is going to get set to the function's default behavior.
(see the previous slide)

For more information check
`QUAD_Model_input_options_template.m`.

Model Output:

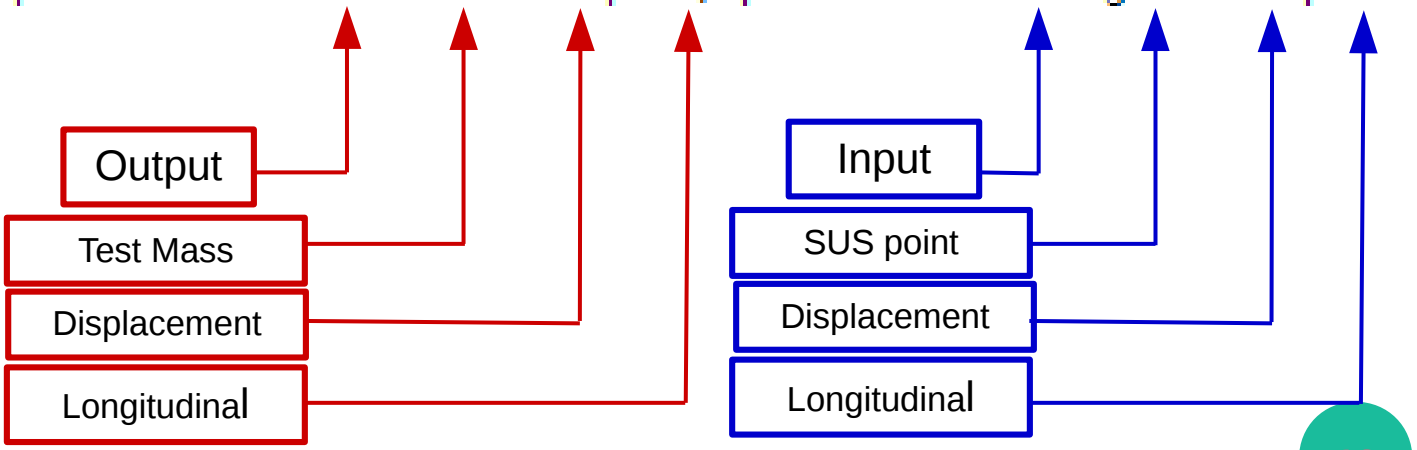
Getting a **Transfer Function** from output:

a) Run the model script

```
>> quadModel = generate_QUAD_Model_Production();
You have chosen to construct the default QUAD Model
Building a single chain model for a fiber QUAD suspension ...
     Using QUAD model:ssmake4pv2eMB5f_fiber
     Using Params file: quadopt_fiber
Calculating the frequency response of the open loop model ...
   Finished open loop in 0.014798 seconds.
```

b) Use **bode** or **bode2** with the **ss** (*resp.* **dampedss**) field of the model:

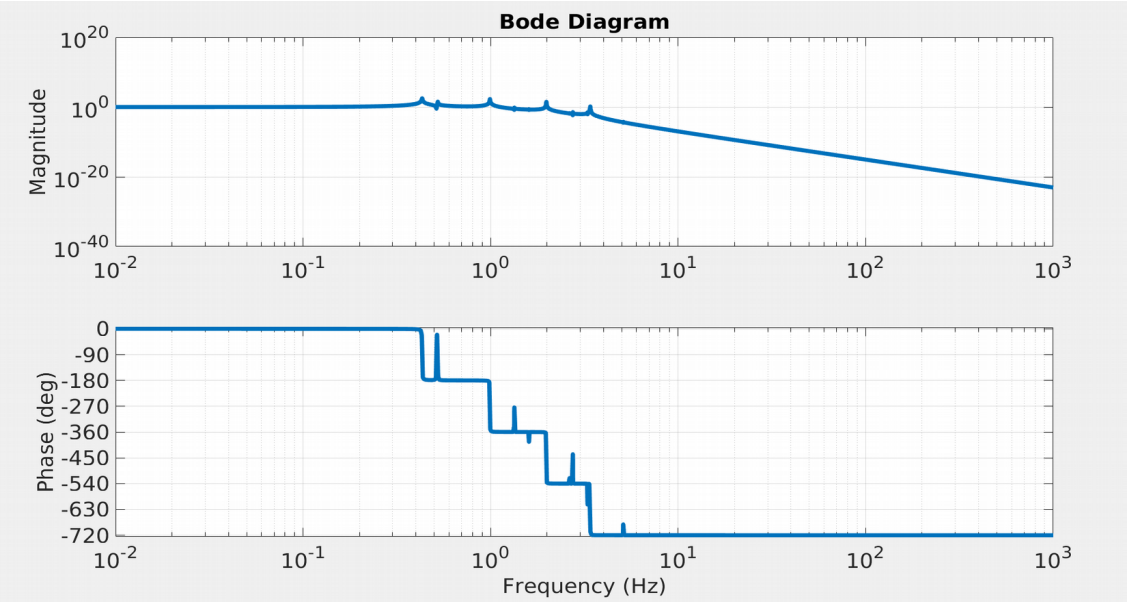
```
>> bode(quadModel.ss(quadModel.out.tst.disp.L,quadModel.in.gnd.disp.L))
```



Model Output:

Getting a **Transfer Function** from output:

c) See the output:



d) If you have a model with damping loops active, you can compare the performance with and without them:

```
>> bode2(quadModel.ss(quadModel.out.tst.disp.L,quadModel.in.gnd.disp.L))
```

Undamped

```
>> bode2(quadModel.dampedss(quadModel.dampedout.tst.disp.L,quadModel.dampedin.gnd.disp.L))
```

Damped