Computer Modeling and Simulation in Support of the Stiff-Suspension Active Seismic Isolation for LIGO II.

Brian Lantz, Wensheng Hua, Sam Richman, Jonathan How, and the Stiff Team

Feb 14, 2000, Version 1.0.1, LIGO-T000016-01

We have developed a very general and versatile approach to model and predict the behavior of complex isolation systems that are being designed for LIGO II. The code is currently being validated against the Stanford prototype, it has already been validated against the GEO 600 code for their triple pendulum, and we will also be validating it with the Stiff Double-Active Stage prototype now under construction. As we test the code against the prototypes, we will gain confidence in its predictions about control and performance. Before April, this code will be used to model the performance of the entire suspension and isolation system we are proposing for LIGO II.

1. Description of the modeling tools

The model consists of three fundamental parts: a model constructor, routines for determining the static equilibrium positions and linearizing the equations of motion around those equilibrium points, and a Simulink interface for evaluating the performance of the isolation system. The system is written completely in Matlab, which has an excellent set of tools for conducting system identification and controller design. The code models the isolation system as a set of springs, masses, sensors, and actuators, then creates a linearized model of that system around the equilibrium condition. Control laws can then be added and the performance of the design is evaluated.

1.1 Modeling Techniques and the Numerical Perturbation Method

All of the various mechanical systems are treated as mass-spring mechanical systems in the same way. For example, the GEO triple pendulum is made of three stages. Each stage has certain mass and moments of inertia. These stages are connected to each other by some springs. The springs could be extensible wires or blade springs. There are sensors monitoring the motions of the stages and actuators applying forces and torques to the stages. Another example is the double stage active isolation system which comprises two serial active platforms from which two (or three) quadruple pendulums are suspended. This is a much more complicated system, but it is still a system made of mass stages, springs, sensors and actuators.

In the model, each mass stage is allowed to translate and rotate in three dimensions. Hence, each stage has six degrees of freedom. Thus, the total number of degrees of freedom of a system is six times the number of stages.

The dynamic model of the system is based on a numerical perturbation technique. Because all the stages in the isolation system work only in small ranges around the equilibrium positions, we assume that the reaction forces and torques are linear to the changes of positions of the stages. To set up the dynamic model of the system, we need to know the reaction forces and torques on the stages due to the motions of those stages in all degrees of freedom. We define the stiffness matrix of a system as the matrix which maps the small changes of positions of the stages onto the reaction forces and torques on those stages. Such reaction forces and torques come from both the deformations of the springs and the changes in the directions of the static force on the springs in the system (e.g. small changes to the position of a pendulum mass change the direction of the static forces applied to it, which results in linear restoring forces). A detailed description of the technique can be found in a document by Wensheng Hua, *How to Construct a Mechanical Model of a Mass-Spring System*, available on the stiff team website.

The stiffness matrices are obtained by linearizing the mass-spring systems around their equilibrium positions. However, this information is not precisely known for the real system, but we can calculate the equilibrium positions of the stage given the physical parameters of the stages and springs. This problem is solved using iteration. The steps are as follows:

1) First, we assume that the stages are at the design position and calculate the net static force and torque on each stage. If the net force and torque on each stage is zero, then the current position is the equilibrium position, and we are done.
2) If we are not in equilibrium, then we linearize the system around the current position and calculate the stiffness matrix.
3) We use this stiffness matrix to calculate the new positions the stages move to so that the reaction force and torque balance the net static force and torque on the stages.
4) In the new position, we recalculate the net forces and torques. If the net force and torque on each stage is zero, then the current position is the equilibrium position, and we are done, otherwise, we go back to step 2.

The iterations are continued until the stages are at their equilibrium positions. Once we have reached static equilibrium, we calculate the final stiffness matrix. This iteration method is equivalent to Newton's method for solving nonlinear equations, and it converges quickly (usually in 3 to 5 iterations).

Once we have the stiffness matrix, we can easily construct a model that represents all of the mechanical dynamics of the system (called the mechanical model). For convenience, the mechanical model is written in a state space form. The states are the positions and velocities of all the stages in all degrees of freedom. The inputs of this model are the ground motions and the actuator forces and torques on every degree of freedom of the system. The outputs of this model are the position of the stages and force on the ground. The ground motion is left as an input and the force on ground as an output. This step provides a convenient way to link our model to a dynamic model of the ground (or a mechanical model of another system) which could be added later.

Next, we make the sensor model and actuator model. The inputs of the sensor model are the positions of the stages. The outputs are the electrical outputs of the sensors. For each sensor, we project the motions of the stages into the sensitive direction of that sensor. We then calculate the output of that sensor based on the sensor's transfer function. The whole sensor model is a collection of all the different sensors in the system, although the modeling tools allow them to be arranged in arbitrary groups in the control simulations.

We follow similar steps in the construct the actuator model. The inputs of the actuator model are the electrical inputs to the actuators. The outputs are the force and torque on the stages.

1.2 Model Connections and Queries

Now we construct the whole model of the isolation and alignment system by connecting the actuator model, the mechanical model, and the sensor model using Simulink.

Once the model of the physical system is assembled, we design the control laws, construct the control model, and close the loop of the whole system. Figure 1 shows the model of the GEO triple pendulum, adapted from the geo_example.m code distributed by Ken Strain, Calum Torrie, et. al. The Simulink interface simplifies the interaction between the blocks in the model, so

the user can open and close loops, add sensor banks, change control laws, etc. with a minimum of difficulty.

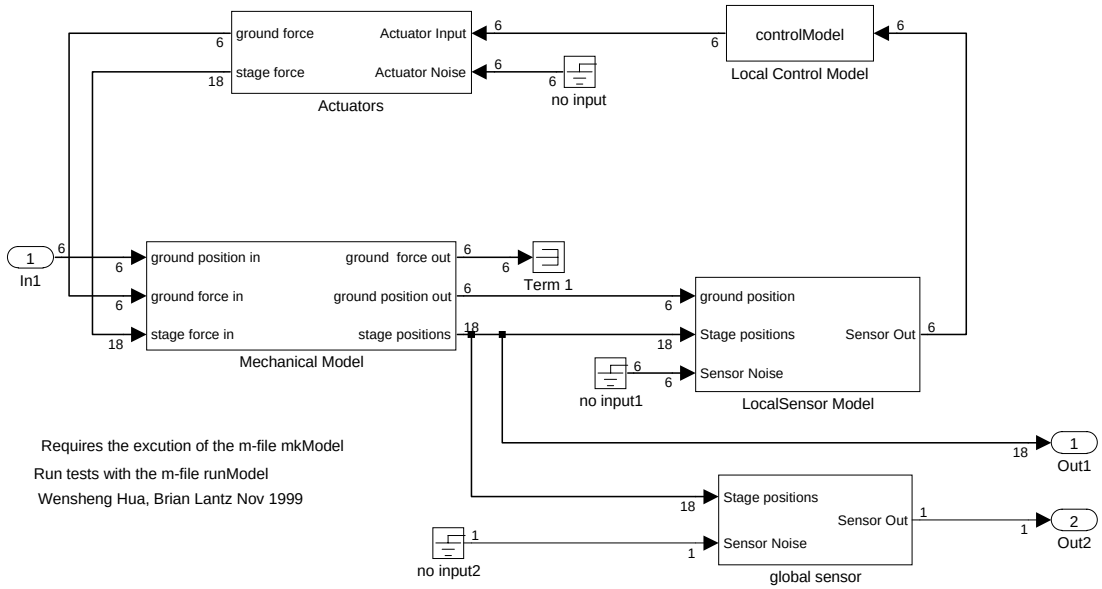

Figure 1. Simulink diagram of the GEO triple model. The parameters match the geo_example.m code distributed by GEO 600.

1.3 The Model Constructor

In the LIGO II isolation system research, we need to model many different systems, such as the GEO pendulums, the various experimental prototypes, and at least two different LIGO II suspension systems (for the BSCs and for the HAMs). These different systems have different kinds of mechanical arrangements, springs, sensors, actuators, and control laws. In order to model these systems effectively, we designed uniform data structures for all the different systems. There are five data structures: stages, springs, actuators, sensors and control laws. These data structures contain all the physical and geometrical parameters of the system. Examples of these data structures can be found in the appendix. These data structures are used as the interface between the system designer and the computer code which constructs the state space model of the system. The designer can easily change the parameters in these data structures and study the performance impacts on the system.

2. Description of the model validation

We are currently engaged in several efforts to validate the performance of the model. The objective is to confirm that the model can be used to accurately predict the performance of our various prototype systems, since this will greatly increase the confidence in our ability to accurately predict the performance of the reference design.

2.1 GEO Triple Pendulum.

We have developed a model of GEO triple pendulum using our model constructor code. We did this in order to compare it with another model of the same system written by GEO. These two models were made independently of each other using different mathematical algorithms. The GEO model has already been experimentally verified. We compared the two systems' stiffness matrices term by term, and the result shows that these two models are exactly the same. These two models produce same transfer functions as well. Figure 2 shows the closed loop transfer function of the GEO triple pendulum from horizontal ground motion to global sensor output.

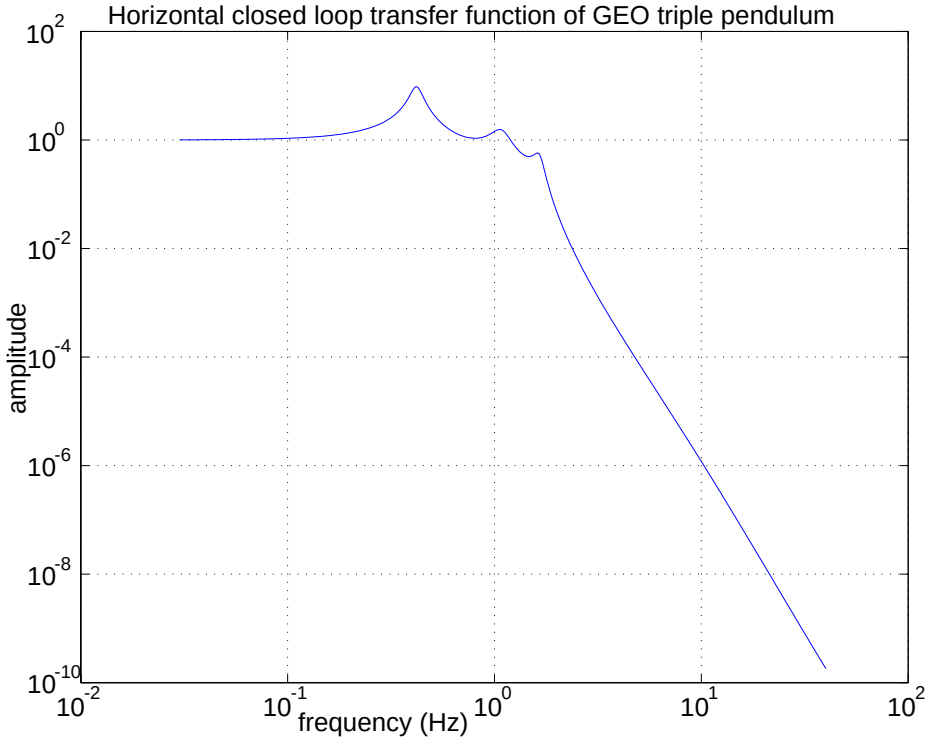

Figure 2. Transfer function from horizontal ground motion to horizontal test mass motion of the GEO triple pendulum as described in the geo_example.m code distribution. This transfer function was generated with the new modeling tools.

2.2 GEO Quadruple Pendulum.

We are constructing a model of the GEO quad pendulum. The GEO group is working on the same model at the same time using their own mathematical algorithms. We are in the process of comparing the models by exchanging the state space matrices and examining them for differences.

2.3 Stanford Prototype

In order to further test our model construction code against a real isolation experiment, we made a model of the single layer active platform built at Stanford. Because it is difficult to measure some of the physical parameters of the system, such as the center of the mass and moment of inertia tensor, we have had to adjust those uncertain physical parameters in the model to fit the transfer function of the model to the experimental results.

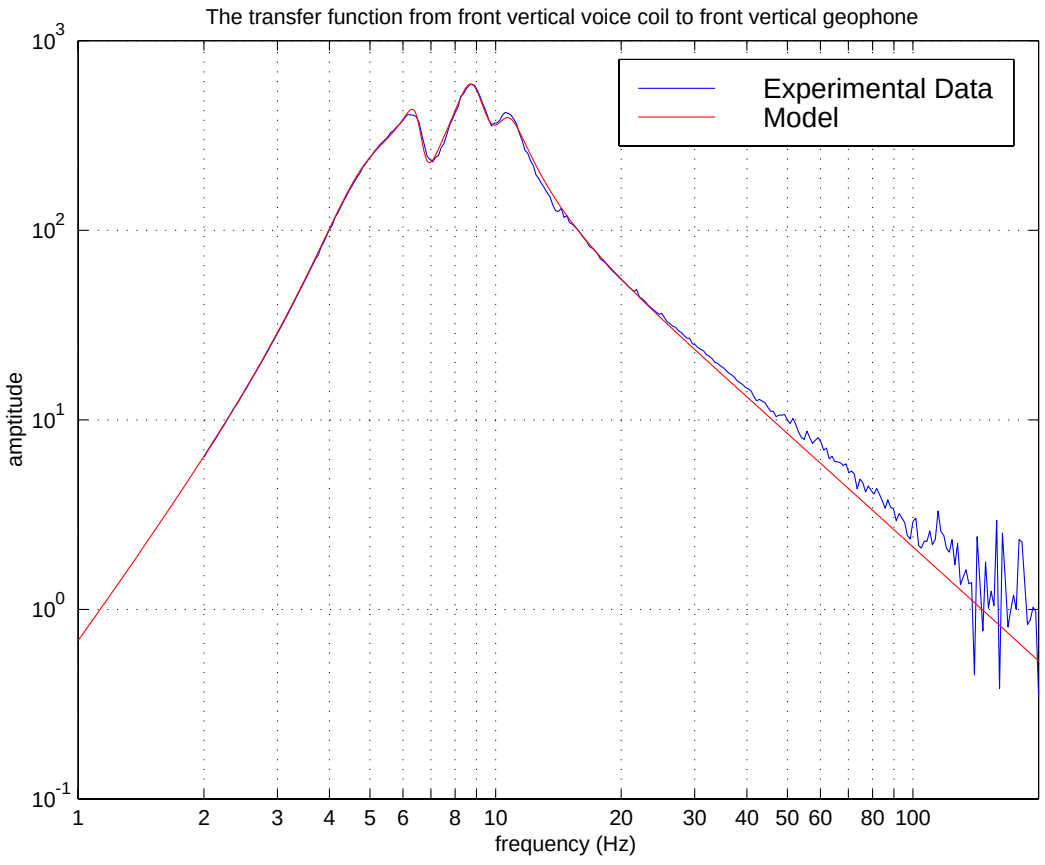

Figure 3. Transfer function of the Stanford single layer active platform and the model, after the model has been tuned.

The red line in Figure 3 shows the transfer function from the front vertical voice coil input to front vertical geophone output according to the model. The blue line shows the same transfer function obtained from the actual system. This transfer function shows a very good match up to about 100Hz. Some of the cross-coupling transfer function fits are not as good as the one in figure 3. These errors in the cross-coupling reflect the sensitivity of the system to slight imperfections in the mechanical system. We will continue to tune the physical parameters of the model to match the measured data as well as possible, and then design the controllers to be robust to any errors in the cross-coupling.

The model validation will continue in the following ways:

1) We will measure all of the necessary transfer functions for the system and form measurement models by fitting a state space system these transfer functions (Liu and Miller '94, Liu et. al. '94). This model will provide very accurate estimates of the modal parameters of the prototype structure, which can be used to further update the physical parameters.
2) These identified models will also be used to design more single-input / single-output (SISO) and multiple-input / multiple-output (MIMO) controllers for the isolation platform. These controllers will be evaluated on the actual testbed and the physical model to determine how accurately we can predict the closed-loop performance.
3) The physical model will also be used to design SISO / MIMO controllers to establish its accuracy as a *control design model*.

The goal is to use this physical model of the system to accurately predict the behavior of the controlled prototype. Close agreement in the close-loop behavior and the ultimate limits in the obtainable performance will greatly improve the confidence of the modeling approach and our ability to develop good control design models using this approach.

When this sequence is complete, we will add the triple pendula to the model to predict the performance of complete Stanford prototype. This model will then be used to evaluate the various control designs and predict the overall performance. The combined model will then be compared with the measured response (e.g. testmass motion and / or forced transfer functions) of the actual system.

2.4 Stiff Double-Active Stage

Preliminary modeling of the two-stage stiff prototype isolation system was done using an adaptation of the model of the JILA technology demonstrator (Richman et. al. '98). This is a state-space representation constructed in Matlab that models the dynamics of the two stages and their respective seismometers. The state-space equations were derived from a Lagrangian of the mechanical system. A controller was developed using single input-single output (SISO) feedback from the seismometers to nearby forcers. This model was used to guide the initial mechanical design of the prototype. The natural suspension frequencies were chosen such that a total isolation factor of 1000 at 10 Hz could be achieved with a 70 Hz bandwidth for the two stage system. The noise from the commercial seismometers used in the prototype was introduced into the model and it was verified that this would not compromise the vibration-limited performance.

We are currently developing another model of the stiff prototype using the numerical perturbation tools and model constructor described in this document. The mechanical model comprises three rigid bodies: the ground and two isolation stages, all with six degrees of freedom. These three are connected in series via idealized blade springs with wires. The outputs of the seismometers and position sensors will be modeled as frequency-dependent combinations of the states of the stages. There is also be a facility for sensor and actuator noise inputs. This model should better represent the stiff prototype than did the preliminary model. Notably, it will include the blending of the low- and high-frequency sensors, and the relative position sensing between the two isolation stages.

The schedule for development of the constructor model of the prototype is slightly advanced with respect to the work on the prototype itself. By February 18 there will be a mechanical model and idealized actuator model in place. This will let us compute transfer functions from actuators and ground motion to motion of the stages. By February 25 we will incorporate transfer function descriptions of the low- and high-frequency seismometers. This will allow for a comparison of driven transfer functions, which will be the first measurements performed with the prototype.

By March 3 we will model a feedback system based on local loops, where the signals from a single position sensor/geophone pair are combined and sent to a single co-located forcer. The algorithm for blending of signals will be based

on previous work done at Stanford (H. S. Bae, '99). The next step will be to construct the model of a feedback system in which a low-frequency seismometer is also blended into the signal from a position sensor / geophone pair for the upper stage loops.

By March 10 we will add sensor noise into the model to make a prediction of the performance limit of the two-stage prototype. Based on results of the preliminary model, it is not expected that sensor noise will preclude the target isolation factor of 1000 at 10 Hz. At that point, further work on the model will be driven by experimental results from the prototype. Possibilities for future work include modeling MIMO feedback, and feed-forward from the low-frequency sensors on the upper stage to actuators on the lower stage.

3. Models of the Reference Design

The goal of the modeling effort is to provide tools for designing and understanding the control and performance of prototypes, so that we can use these tools to establish a reference design for LIGO II. By April, we will have a reference design of all the double active stages and quadruple pendula for the BSC and the HAM. The model will include:

1) Dynamics of the system for all degrees of freedom. This is automatic for this modeling system.
2) The isolation control loops for the double active stage.
3) The pendulum dynamics of the quadruple pendulum with its local damping, and the impact of those dynamics on the performance of the system.
4) Couplings of the ground noise, sensor noise, and actuator noise into the motion of the test mass to show that we meet the noise requirements.
5) Global position feedback and calculations of the rms position of the test mass. We are currently examining the global model distributed by Ken Strain so that we can leverage off of those results.
6) Calculations to show that all states of the system are observable and controllable.

Conclusions

We have developed a powerful tool for analyzing isolation systems. The numerical perturbation technique allows any assembly of masses and springs to be converted into a linear state space model. The model constructor allows

simple modification of the physical structure of the system, and allows simple positioning of sensors and actuators on the system. Since this is done with Matlab, we have a powerful set of modern control tools available for the development of the control systems. By assembling the pieces with Simulink, we have a convenient way of organizing the model connections to simplify the development process.

We have tested the code against the GEO triple results, and it matches perfectly. We are currently testing the code against the experimental prototypes, to further validate the code and gain insight about the control design. This gives us confidence that the simulations of the reference design will be reasonably accurate.

References

Hong Sang Bae, *Active Vibration Isolation and Alignment Issues for LIGO*, Masters Thesis, Department of Mechanical Engineering, Stanford University, August 1999.

Liu, K. and D. W. Miller, ``Structural System Identification: A Study of Different Algorithms,'' *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Scottsdale, AZ, 1994, pp. 702--714.

Liu, K., R. Jacques, and D. Miller, ``Frequency Domain Structural System Identification by Observability Range Space Extraction,'' *Proceedings of the American Control Conference*, Baltimore, MD, 1994, pp. 107--111.

S. J. Richman, J. A. Giaime, D. B. Newell, R. T. Stebbins, P. L. Bender, and J. E. Faller, *Rev. Sci. Instrum.* **69**, 2531 (1998).

Wensheng Hua, *How to Construct a Mechanical Model of a Mass-Spring System*, this document is available at the stiff team website.

Appendix – Examples of data files used by the model constructor

Mechanical Model: stages

Here is the model constructor for a single stage of a isolation system, many of the parameters are defined elsewhere.

```
% stage.m holds the data structure of stage.


StgSGrd=1;

StgStck=2;


%physical Parameters

stage(StgStck).name='StgStck';

stage(StgStck).center=[0 0 -frameMcz]';

stage(StgStck).axis=eye(3);

stage(StgStck).mass=fmass;
stage(StgStck).Ix=fIx;
stage(StgStck).Iy=fIy;
stage(StgStck).Iz=fIz;
stage(StgStck).inerAxis=Rt;
%derived parameters
stage(StgStck).forceInputAddr=2;
stage(StgStck).posiOutputAddr=2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Mechanical Model: springs

This is an excerpt of the file used to define the springs of the single layer active platform.

```
% spring.m is the data structure of spring.

SprSGrdStak_1=1;
SprSGrdStak_2=2;
SprSGrdStak_3=3;
SprSGrdStak_4=4;
SprSGrdStak_5=5;
SprSGrdStak_6=6;

% data inform parameters:
```

```
spring(SprSGrdStak_1).name='SprSGrdStak_1';
% geometry parameters:
spring(SprSGrdStak_1).uStageNum=StgSGrd;     % The stage number of the upper stage.
spring(SprSGrdStak_1).uPosition=[fNXOffset -fSprUy -SprUz]'; % The contacting point of
     the spring on the upper stage in the upper stages local coodinate.
spring(SprSGrdStak_1).uDirc=cookUnit([0 0 -1]',[0 1 0]');
spring(SprSGrdStak_1).lStageNum=StgStck; %The stage number of the lower stage.
spring(SprSGrdStak_1).lPosition=[fNXOffset -fSprLy -SprLz]'; % The contacting point of
     the spring on the lower stage in the lower stages local coodinate.
spring(SprSGrdStak_1).lDirc=cookUnit([0 0 1]', [0 1 0]');
% physical parameters:
spring(SprSGrdStak_1).type=STypeWire;  % The type of the spring enable the model to use
     different type of springs.
spring(SprSGrdStak_1).elastic=[sprK sprIniL 0 0]; % some elastic constance, the first one
     is the zero-force spri
%constant the other constance can be used for non-linear springs.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Sensors

Here is the code which defines the first two local sensors of the GEO 600 local control.

```
%GeoTripleLocalSensors.m

% sensor.m The data structure of sensor.

SenPen1PGrdH=1;
SenPen1PGrdT=2;
SenPen1PGrdV=3;
SenPen1PGrdY=4;
SenPen1PGrdP=5;
SenPen1PGrdR=6;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sensor(SenPen1PGrdH).name='SenPen1PGrdH';
% geometry parameters:
sensor(SenPen1PGrdH).mainStageNum=StgPen1;   % The stage number of the main stage, the
     stage that holds the sensor.
sensor(SenPen1PGrdH).mainPosition=[0 0 0]'; % The sensor's position on the main stage in
     the main stage's local coordinate.
sensor(SenPen1PGrdH).mainBaseDirc=[1 0 0;0 0 1]'; % The sensor's sensitive direction in
     the main stage's local coordinate.
sensor(SenPen1PGrdH).refStageNum=StgPGrd; %The stage number of the sensor's reference
     stage.
sensor(SenPen1PGrdH).refPositionA=[0 0 0]'; % The reference points of the sensor on the
     reference stage in the reference stages local coodinate.
sensor(SenPen1PGrdH).refBaseDirc=[-1 0 0;0 0 1]'; % The sensor's sensitive direction in
     the main stage's local coordinate.

% physical parameters:
```

```
sensor(SenPen1PGrdH).type=SenType.IdealRelate;  % The type of the sensor enable the model
    to use different type of sensors.
sensor(SenPen1PGrdH).gain=[1 0 0 0 0 0]; % The gain the sensor
sensor(SenPen1PGrdH).transFunc=tf([1],[1]); % The transfer function of the sensor

%derived parameters:
% Note: All the dumy settings here will only be used for date struture,but not for real
    calculation.
sensor(SenPen1PGrdH).geoFunc=zeros(1,12);
% The sensor's transfer function from the stages' position to the senser output.
% .sensFunc is multi-input-single-output.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sensor(SenPen1PGrdT).name='SenPen1PGrdT';
%geometry parameters:
sensor(SenPen1PGrdT).mainStageNum=StgPen1;
sensor(SenPen1PGrdT).mainPosition=[0 0 0]';
sensor(SenPen1PGrdT).mainBaseDirc=[1 0 0;0 0 1]';
sensor(SenPen1PGrdT).refStageNum=StgPGrd;
sensor(SenPen1PGrdT).refPositionA=[0 0 0]';
sensor(SenPen1PGrdT).refBaseDirc=[-1 0 0;0 0 1]';
% physical parameters:
sensor(SenPen1PGrdT).type=SenType.IdealRelate;
sensor(SenPen1PGrdT).gain=[0 1 0 0 0 0];
sensor(SenPen1PGrdT).transFunc=tf([1],[1]);
%derived parameters:
sensor(SenPen1PGrdT).geoFunc=zeros(1,12);
```

Actuators

Here is example code to define two actuators. It references several variables which are defined elsewhere.

```
% actuator.m this is the data structure defining the actuators.
ActSGrdStckH1=1;
ActSGrdStckH2=2;
ActSGrdStckH3=3;
ActSGrdStckV1=4;
ActSGrdStckV2=5;
ActSGrdStckV3=6;

actuator(ActSGrdStckH1).name='ActSGrdStckH1';
% geometry parameters:
actuator(ActSGrdStckH1).mainStageNum=Stg.Stck;       % The stage number of the main stage,
    the stage that holds the actuator.
actuator(ActSGrdStckH1).mainPosition=[fNXOffset -fHGeoVoiceBy -hGeoVoicez]'; % The
    actuator's position on the main stage in the main stage's local coordinate.
actuator(ActSGrdStckH1).mainBaseDirc=cookUnit([fNXOffset -fHVoiceTy -hGeoVoicez]'-
    [fNXOffset -fHGeoVoiceBy -hGeoVoicez]' ,[0 0 1]'); % The actuator's Active direction
    in the main stage's local coordinate.
actuator(ActSGrdStckH1).refStageNum=Stg.SGrd; %The stage number of the actuator's
    reference stage.
```

```
actuator(ActSGrdStckH1).refPosition=actuator(ActSGrdStckH1).mainPosition+stage(actuator(ActSGrdStckH1).mainStageNum).center; % The reference points of the actuator on the reference stage in the reference stages local coodinate.
actuator(ActSGrdStckH1).refBaseDirc=actuator(ActSGrdStckH1).mainBaseDirc; % The actuator’s Active direction in the main stage’s local coordinate.

% physical parameters:
actuator(ActSGrdStckH1).type=ActType.Voice;  % The type of the actuator enables the model to use different type of actuators.
actuator(ActSGrdStckH1).gain=[1 0 0 0 0 0]’; % The gain the actuator
actuator(ActSGrdStckH1).damping=dampLamda; % The damping of the actuator
actuator(ActSGrdStckH1).transFunc=voiceTrans; % The transfer function of the actuator

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
actuator(ActSGrdStckH2).name=’ActSGrdStckH2’;
% geometry parameters:
actuator(ActSGrdStckH2).mainStageNum=Stg.Stck;
actuator(ActSGrdStckH2).mainPosition=[-osHVoiceBx +osHVoiceBy -hGeoVoicez]’;
actuator(ActSGrdStckH2).mainBaseDirc=cookUnit([-osHVoiceTx +osHVoiceTy -hGeoVoicez]’-[-osHVoiceBx +osHVoiceBy -hGeoVoicez]’ ,[0 0 1]’);
actuator(ActSGrdStckH2).refStageNum=Stg.SGrd;
actuator(ActSGrdStckH2).refPosition=actuator(ActSGrdStckH2).mainPosition+stage(actuator(ActSGrdStckH2).mainStageNum).center;
actuator(ActSGrdStckH2).refBaseDirc=actuator(ActSGrdStckH2).mainBaseDirc;
% physical parameters:
actuator(ActSGrdStckH2).type=ActType.Voice;
actuator(ActSGrdStckH2).gain=[1 0 0 0 0 0]’;
actuator(ActSGrdStckH2).damping=dampLamda;
actuator(ActSGrdStckH2).transFunc=voiceTrans;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```