LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO-T1300942-v1 *LIGO* July 6, 2015

SIS15, *FOG inside*, Manual

Hiro Yamamoto

Distribution of this document:
LIGO Scientific Collaboration

This is an internal working note
of the LIGO Project.

California Institute of Technology
LIGO Project – MS 100-36
1200 E. California Blvd.
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project – NW22-295
185 Albany St
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
P.O. Box 1970
Mail Stop S9-02
Richland WA 99352
Phone 509-372-8106
Fax 509-372-8137

LIGO Livingston Observatory
P.O. Box 940
Livingston, LA 70754
Phone 225-686-3100
Fax 225-686-7189

http://www.ligo.caltech.edu/

1 Introduction

1.1 What is "SIS15, *FOG inside*"

There are several FFT-based IFO simulation packages, like DarkF, MIT-FFT and SIS[1)]. These simulation tools calculate fields in optical systems by taking into account realistic details of optical components, like macroscopic shapes and microscopic surface phasemaps. Fields are calculated under static or stationary conditions, and the role of those tools is complimentary to frequency domain and time domain simulation tools.

I developed the original SIS[1)] not only to calculate fields with easy to use user interface, but also to serve as analysis tools of opticcal system characteristics. E.g., after stationary state fields are calculated, mode analysis can be done to see the mode matching and the Gaussian fitting can be done to calculate effective beam size and curvature.

The optical configurations supported by SIS as of 2013 were limited to up to copupled cavities, although an object oriented modular structure was used internally for the expandability in mind. Another constraint was that this program was written in C++ for speed and memory usage, and an executable binary properly built for each platform was necessary to run the code.

Richard Day of EGO developed a FFT-based simulation package FOG, *Fast Fourier Transform Optical Simulation of Gravitation Wave Interferometer*[2)]. This package is consisted of elements to build a IFO simulation setup, and of algorithms how to combine these parts for an arbitrary configuration. FOG was written using matlab. He analyzed the details of the FFT-based simulation framework and improved the acceleration algorithm of field calculations in complex optical systems together with G. Vajente0[3)].

FOG is like a toolbox for building optical systems. By combining tools, any optical configuration could be built and simulated. But the user has to assemble parts to built the setup for a specific optical configuration and various parameters have to be calculated and set by hand.

This FFT-based simulation package, SIS15, *FOG inside*, SIS15 for short, was designed by the collaboration of R.Day and H.Yamamoto, to make a package which combines the strength of SIS and FOG. Yamamoto implemented the program in matlab by combining codes of two of us.

SIS15 is consisted of three parts, first is to *build optical systems*, second is to *setup fields*, and third is to *analyze fields*. For example, "addMirror" is used to install a mirror in the setup, "lock" is used to lock the system and calculate fields and "getField" is used to get field amplitudes. One matlab file is necessary to be prepared to define the optical setup, which is refered to as IFO setup file, and the rest of the analysis can be done interactively or by combining analysis scripts in a file for complex or repetitive analysises, refered to as analysis file.

First, a simple example is used to explain the basic idea of SIS15. Then one section explains the infrastructure of SIS15, followed by functions commonly used for analysis. And bra bra bra.

1.2 Basic procedures

To use SIS15, one needs to prepare one IFO setup file, which defines an optical configuration of the setup to be simulated. [1]

One simple example of a FP cavity setup file is as follows.

```
classdef FP < FFTIFO
    methods
        function defineIFO( obj )
            % add optics
            obj.addLaser( 'src', 'power', 1 );
            obj.addMirror( 'ITM', 'invRoC', 1/1934, 'T', 0.014 );
            obj.addMirror( 'ETM', 'invRoC', 1/2245, 'T', 5e-6 );
            % define connections among optics
            obj.addProp( '', 'src', 'ITM-AR' );
            obj.addProp( 'FPprop', 'ITM-HR', 'ETM-HR', 4000);
        end
    end
end
```

A minimal template setup file is as follows.

```
classdef IFOname < FFTIFO
    methods
        function obj = IFOname( varargin )
            obj@FFTIFO( varargin{:} );
        end

        function defineIFO( obj, varargin )
            %=> set variables
            eval( UtilTK.setVars( who, true, varargin{:} ) );
            %=> add optics
            %=> connection optics
            %=> setup details of optics, like maps
        end
    end
end
```

IFOname can be anything, like FP or IMC, and the file name should be the same as the name chosen for IFOname, like FP.m or IMC.m. Example codes have detail explanations of matlab scripts. At each %=> in the template, you insert your implementations. "varargin" and setVars are necessary to support run time arguments, which is explained in each example.

Once this setup file is prepared, there are three common steps.

A) ifo = IFOname; % prepare setup

B) ifo.lock; % prepare fields

C) [E,xaxis,yaxis] = ifo.getField('ITM-HR-o'); % analyze fields

Step A) creates an cavity object, like building an interferometer by assembling hardware parts. No laser field is ready yet, but the mode analysis is completed based on the optics specification, like RoC of mirrors and distances between optics.

[1] From the programming point of view, this is a derived class of the superclass FFTIFO.

Step B) turns on the laser source and the cavity is locked. The default locking algorithm used is to make the round trip phase of the stationary field to be 0. After calling lock (lock the cavity and calculate fields) or calc (without adjusting cavity length, stationary fields are calculated), all fields in the cavity are ready to be analyzed.

Step C) is to analyze the cavity field. getField returns the field and the coordinate. SIS15 uses adaptive coordinates in the entire system, and the coordinates at each location can be different. The complex field E(yj, xi) is the amplitude of the field in units of √W / m at (x,y) = (xaxis(xi), yaxis(yj)). For most of the cases, the xaxis and yaxis are the same, so you can use xaxis for both x and y.

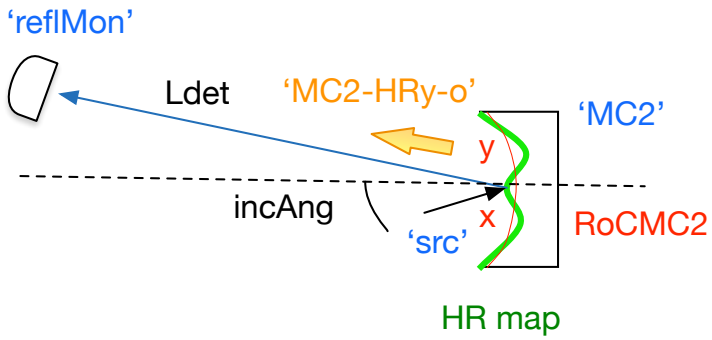
In the following section, several example programs are used to explain the outline of SIS15. Highlights in each example are shown by red underlined names.

1.3 Simple examples

1.3.1 Reflection by a mirror

IFO setup file : MirrorRef.m.

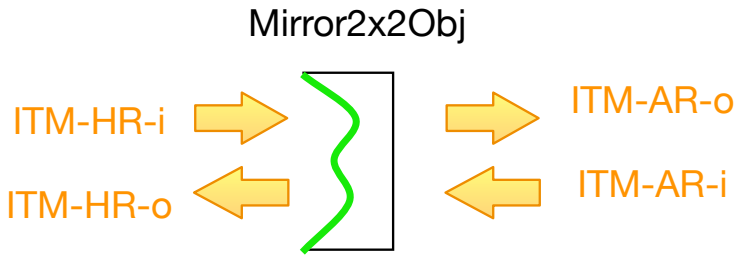
Optical configuration



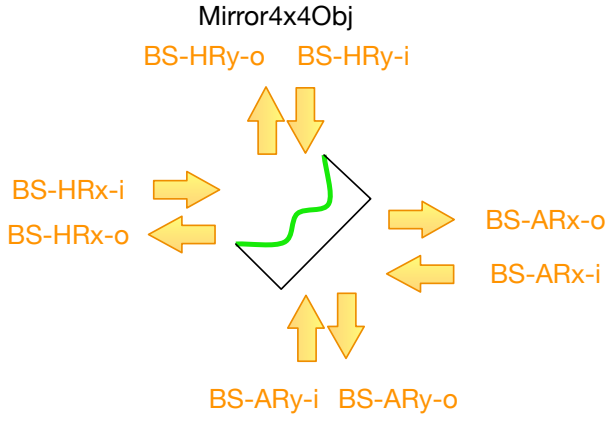
addMirror

addMirror('incAngle', angval) creates a BS like mirror, while addMirror without incident angle specification creates a test mass like mirror.

Mirror with no incident angle specification



Mirror with incident angle specification

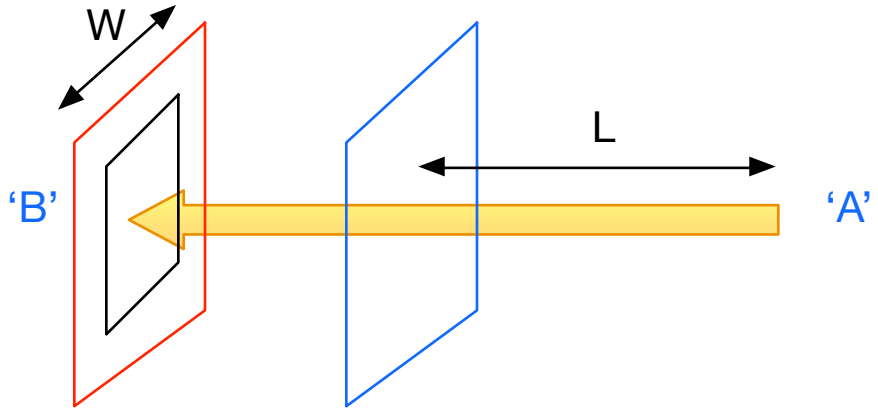


getField

[E, x, y] = ifo.getField(‘B’) returns the field at location B using the window size automatically setup by SIS15, like in the black square.

[E, x, y] = ifo.getField(‘B’, W) returns the field at the same location but using a larger window size specified at the second argument, like in the red square.

[E, x, y] = ifo.getField(‘B’, W, L) returns the field at distance L, the third argument, from the source of the field, A, like in the blue square. If W is positive finite, that value is used as the window size, otherwise a default size is used. The default window size is determined based on the beam size at the location field is calculated, and the default window size at L will be different from the default window size at ‘B’.



setHRfiles

The maps of a mirror are specified by setHRfiles (HR surface maps), setTRfiles (transmission maps) and setARfiles (AR surface maps).

```
setHRfiles( 'mirrorName', 'dataFile',
            'formula',
            apertureForRoC, valueOfRoC, orientation,
            ‘var1Name’, var1Val, ‘var2Name’, var2Val, … );
```

The first argument is the name of the optic, like ‘MC2’, and the second is the name of the map file. Mirror maps are measured using a reference surface, and a map comes with a measured value of RoC calculated in a certain aperture. E.g., power or RoC of aLIGO test masses are measured using data in an aperture of 160mm. The fourth argument and fifth argument specifies these aperture and measured RoC. When this map data is used, first tilt and curvature terms are subtracted using map data within this aperture, then curvature is added using the specified RoC. When either of these values is 0, they are not used and the map data is used as is on top of the mirror with RoC specified in addMirror.

The third argument ‘formula’ can be used to modify the map. In the formula, the map data can be referred using a variable name map. Special variables available are x and y. If you want to add a tilt and extra curvature, you specify the formula as follows.

```
setHRfiles( 'mirrorName', 'dataFile',
            'map + tiltX * x + (x.^2 + y.^2)/(2*ExtraRoC)',
            apertureForRoC, valueOfRoC, orientation,
            'tiltX', 1e-8, 'ExtraRoC', 100e3 )
```

When the formula is specified like this, extra tilt of 1e-8 radian and extra power term of 100km are added. Once the HR surfaces are defines using this formula, you can change these values by calling

```
ifo.updateMirMaps( 'mirrorName', 'variableName', newVariableValue).
```

E.g., ifo.updateMirMaps('MC2', 'tiltX', 1e-7). When these values are updated, fields need to be recalculated, using ifo.calc or ifo.lock. For the case of MirrorRef, there is no cavity and fields do not need to be recalculated.

setupBaseMode

The mode needs to be specified in some way. If there is any cavity which can be used as a reasonable default mode base, like the arm cavity of PRFPM, then that mode is automatically used to define the mode of the entire system, by default.

Either there is no good reference cavity which can be used to define the default mode, like this mirror reflection, or if a different mode wants to be used, like intentionally shifting the waste position of the input beam, setupBaseMode is to be implemented in the IFO setup file.

In this function, call obj.cavities.fillQvals(root, qInv), where root is the ID of the field you are going to use to specify the mode and qInv is the inverse of the q value of the field,

```
function setupBaseMode( obj, varargin )
    qInv = 1/R - i lambda/pi 1/w^2
    root = obj.optics.findLaser;
    obj.cavities.fillQvals( root, qInv );
end
```

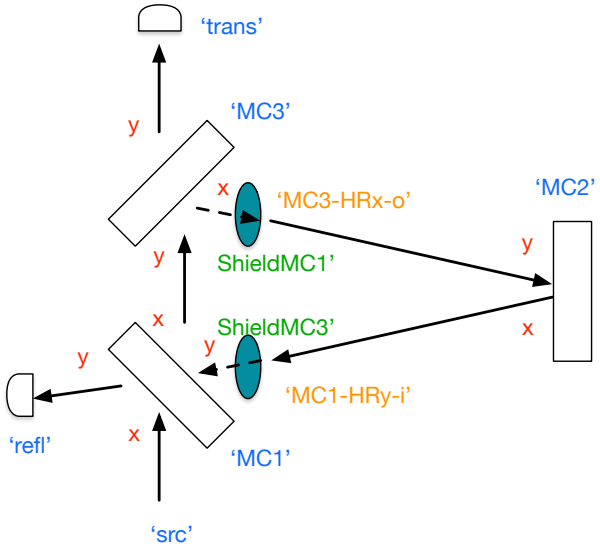
Any field can be used to specify the base mode, and the field ID can be found by

```
ID = findNamedField( 'field name' )
```

1.3.2 Input Mode Cleaner

IFO setup file is AdVirgoIMCIFO.m.

Optical configuration



setModMap('mirrorID', @functionHandle)

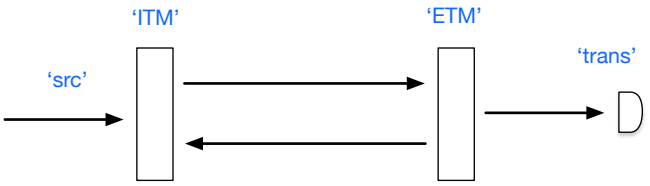
Before the simulation starts, various mirror maps are prepared to calculate the transmission and reflections on the mirror. After nominal calculations of maps are finished, if any functionHandle is registered for mirrors, those handles are called so that one can do arbitrary modifications of the maps.

In the example code for AdVirgoIMC, the hold of the shield in front of the DIHEDRON is applied on the maps prepared without assuming the shield.

1.3.3 FP cavity

IFO setup file is FPIFO.m

Optical configuration



ifo.setRF(RF)

The nominal laser wavelength is defined in ConstantTK.m. If you want to calculate the RF sideband, use this function to change the frequency. Example is in scanRF which called in the example script of FPIFO.

findBeamSize

The mode base are not readily available when the setup is being built in defineIFO function. But they can be calculated after mirrors and connections among them are done. The function defined in this example code shows how to do. In short, after call analyzeCavity and setupBaseMode, the field information based on the modal analysis are available, like the beam size or curvature.

But the analysis in the example is not quite right. After the mirror is heated and the surface is deformed, the beam size start to change. So the correct way is to do a recursive calculation. Calculate the beam size using cold state parameter. Simulate the stationary state field. Calculate the thermal distortion. Simulate the stationary state field using the thermal distortion. Calculate the thermal distortion using the new beam size. Simulate the field with the new deformation, … Repeat until the change of the distortion is small enough.

2 Basic functions

2.1 Building IFO

2.2 Modifying IFO

1) FFTIFO(MirrorObj).updateMirMaps : change parameters in the formula defined in setHRfiles, setTRfiles and setARfiles.

   ifo.updateMirMaps('mirrorName', 'var1Name', var1, 'varName2', var2, …)

2.3 Analysis tools

In the following, ID values used to specify a field can be either ID number or character string, like 'ITM-HR-o', for most of the case…

2.3.1 loss = ifo.roundtipLoss(ID) : round trip loss.

If no argument is specified, losses in all cavities are calculated. If you want to get the loss of a specific cavity, use the field ID in that cavity, like 'ITMX-HR-o' to get the loss in the X arm.

2.3.2 P = ifo.power(ID) : power of the field

2.3.3 Cnm = ifo.LGCoef(ID, n,m) or ifo.HGCoef(ID, n,m) : Complex amplitudes of Laguerre and Hermite mode of the field

like ifo.HGCoef('ETM-HR-o', 0:5,0:5).

2.3.4 Fnm = ifo.LGFrac(ID,n,m) or ifo.HGFrac(ID, n,m) : Power fractions of Laguerre and Hermite mode of the field

2.3.5 [infoStr, infoVal] = ifo.gaussFit(ID) : the Gaussian fit of the field near the center of the field

infoStr returns the result in string, so just by tying ifo.gaussFit(ID), you can see the fit result. E is fit as

$$\boldsymbol{E(x,y) = Exp(-(x-cx)^2\left(\frac{1}{wx^2} + ik\frac{1}{2\,R_x}\right) - (y-cy)^2\left(\frac{1}{wy^2} + ik\frac{1}{2\,R_y}\right))}$$

and infoVal(1) is for x axis and infoVal(2) is for y axis. infoVal(i).c0 is the center of the gaussian distribution, w is the width, and infoVal(i).invRk = k/R.

If the value for x and y are very close, only one value is shown for both.

This fit may fail is the distribution is very different from a Gaussian shape.

2.4 Utilities

2.4.1 UtilTK.mapping : map 2D data to different coordinate system, rotation, scaling, etc

```
function mapped = mapping( x, y, dat, newX, newY )
mapped = mapping( x, y, dat, newX, newY )
```

This calculate data at new coordinate system by changing the data by rotating, moving or expanding.
x and y are 2D coordinates in the original frame and dat(Ny,Nx) is a 2D data set defined in the original coordinate system. The result is a 2D data in the new coordinate system defined by the newX and newY, which are functions of the original coordinate (x,y).
newX/newY defines coordinate change, opposite to object change
examples :
1) newX = 'cosVal * x - sinVal * y', newY = 'sinVal * x + cosVal * y'
   This will rotate mat. With cosVal=cos(pi/3) and sinVal=sin(pi/3), 60 degree rotation
2) newX = '2*x', newY = '0.5*y'
   This will shrink object by factor 2 in x direction and expand by factor 2 in y direction

2.4.2 `loadOneDataFile` : read data from a data file

```
function [amap, xl, yl] = loadOneDataFile( fileName, mirrorAperture [0] ,
ApertureFit [-1], removeZ [2] )
```

Load data from a file and return data(NY,NX), xl(1,NX) and yl(NY,1)
Datafile format supported are (as of June 2015)
Metropro data, Wyko OPD, LMA col and SIS_Nxxx_Wyyy

If mirrorAperture > 0 is provided, data out of this aperture is filled by NaN
If ApertureFit is given and positive, zernite terms are removed.
removeZ = 1 : up to tilt
removeZ = 2 : up to power, which is default

To support a new format data, add the data loader of that format in this function
This function cashes data to speedup the calculation. To clear the cash, call
loadOneDataFile('CLEARCASH').

2.4.2.1 SIS_Nxxx_Wyyy : simple format for ascii data

This is a format to access NxN data stored using an ASCII format. The name of the file, "SIS_Nxxx_Wyyy.txt", is used to specify the window size and number of grids. "SIS" can be any string, xxx is the number of grids and yyy is the size of the window where the data are defined. E.g., if a map is defined in a 340mm x 340mm window using128 x 128 data points, the file name will be myData_N128_W340mm.txt.

The widow size specification, yyy, can have a unit specification at the end : cm, mm and um for centimeter, millimeter and micrometer. If there is no unit at the end, the default is mm. 34cm, 340mm and 340 all specifies the same window size of 34cm x 34cm.

If you have the data in matlab, you can create this data file as

```
save myData_N128_W340mm.txt –ascii transpose( data )
```

The reason that the data is transposed is the convention of the array structure in matlab. You can check if the orientation of the data in the file is correct, load the data as

```
[inData, xy] = loadSISData( 'myData_N128_W340mm.txt')
```

and confirm if

```
plot( xy, inData(end/2,:) )
```

is the distribution along x axis.

2.4.3 UtilTK. setVars : change variable values

UtilTK.setVars(nameList, showIt, varargin)

nameList is a list of variable names which can be specified in varargin.

showIt turns on and off the printout of the variables actually changed.

varargin = 'name1', val1, 'name2', val2, …

2.5 Support functions

2.5.1 **calcThermal :** Thermal deformation using Hello-Vinet formula

function dat = HVThermal(elph, r, Psub, Pcoat, w, thickness, radius)

Calculate thermoelastic (elph = 0) and thermal lens(elph=1) using Hello-Vinet formula.

r is an array of radial distance where the deformations are calculated.

Psub and Pcoat are powers absorbed in the substrate and in the coating.

w is the beam size on the mirror.

Thickness and radius are the size of the optic.

This calls HVThermal and it provides more flexibility. clcThermal assumes the substrate is a fused silica, but, other substrates can be simulated by calling HVThermal.

3 Program Basic

3.1 Simulation package structure

3.2 Basic Function calls

4 Program Details

5 Physics in the simulation

6 Reference

1) H.Yamamoto “SIS (Stationary Interferometer Simulation) Manual”, LIGO-T070039-v8, 2013.

2) R.A. Day “A new FFT code : FOG” GWADW 2012, Hawaii USA

3) R.A. Day and G. Vajente “Accelerated Convergence method for the FFT simulation of coupled cavities”, 2013.

4) B.Bhawal, H.Yamamot, et al “e2e primitive module – Reference Manual” LIGO-T00047, H.Yamamoto “e2e – LIGO end to end time domain simulation” LSC/Virgo Collaboration Meeting, October 2007, Hannover Germany