Lock Acquistion for the arms

VIR-0656A -10

F.Cavalier

26/11/2010

Table of contents

1. Aim of the simulation

The aim of this simulation was to check if the auxiliary lasers are mandatory for the lock acquisition of the arms. The full lock acquisition problem is not tackled in this document.

2. Description of the simulation

- We simulated a single Fabry-Perot cavity with a finesse equal to **450**.
- A dynamical simulation is used for the optic in order to see the impact of deformation of Pound-Drever signal.
- We use the same corrector as the one currently used on site (see Annex 2) for details with a unity gain frequency around 50 Hz.
- No noise has been put at the actuation level but usual limitations on voltage and current have been applied. By default, we used a N/A conversion factor equal to 9.85 10^{-3} (see Annex 1 for details).
- The correction is activated only when the transmitted power is above a given threshold (0.01). Such low threshold is not needed at low speed but is imposed by the dynamical effects which tend to lower the high of the resonance peak at high speed.
- The lock sequence was the following one:
  1. Let the cavity free for 10 seconds
  2. Try to lock during 60 seconds
  3. Cavity is considered locked if the transmitted power is above threshold during at least 1 second
  4. Unlock at the end of the 60 seconds
  5. Go back to step 1

3. Impact of residual motion

The amplitude of the seismic noise has been varied from 2.10^{-9} to 8.10^{-8} (second argument of GRound in the siesta card). An amplitude equal to 2.10^{-8} leads to a motion with amplitude about 1.2 µm and a mean speed about 1.5 µm/s. For amplitude lower than 2.10^{-8}, the lock is always acquired in less than 60 seconds. No failure has been observed on the 142 tries performed for each simulation.

The table below gives the success fraction as a function of the seismic amplitude:

Amplitude (x 10^{-8})	Fraction of successful tries (%)
2	100
4	80
6	63
7	33
8	13

$A=8.10^{-8}$ corresponds to motions with large amplitudes (5 µm) and high speed (6 µm/s). Probably, the success fraction at this level could be increased with a cleverer algorithm. In reasonable conditions, the lock is acquired quite easily.

4. Impact of delays

Using the USdelay structure, we investigate the effect of the delay in the control loops. Under reasonable hypothesis for residual motion ($A=2.10^{-8}$), the lock is always acquired if the delay is below 1 ms. As the current delay is about 350 µs, no problem is expected from this side.

5. Impact of force limitation

The table below presents the fraction of successful tries for various values of the conversion factor. A high power mode is needed for lock acquisition but a conversion factor about 5 mN/A remains acceptable.

Current to Force conversion factor	Fraction of successful tries (%)
$9.85\ 10^{-4}$	1.4
$2\ 10^{-3}$	23
$3\ 10^{-3}$	64
$4\ 10^{-3}$	90
$5\ 10^{-3}$	91
$9.85\ 10^{-3}$	100

6. Conclusion

It seems that under reasonable assumptions (residual motion and actuation similar to current ones) that lock acquisition for the arms can be easily performed with duration below one minute using the Virgo strategy and it can be claimed that no auxiliary laser is needed for this purpose. Obviously, if the finesse is decreased to 250 (LPRC option), it will be even easier.

7. Annex 1: siesta card

Here is a copy of the siesta card which has been used for the simulation. The starting point was a card used by Lisa Barsotti for lock acquisition studies on Virgo.

```
UJconst duration 100000000
UJconst duration2 1000000

UJclock masterClocks duration 2 10.e3 10e1

USignal false 0.0
USignal true  1.0
USsweep time 0 0. 1.

GRound 0 2.e-8 0.1  0
GRoundPt pI 0. 0. 0.
GRoundPt pE 3000. 0. 0.

/*--------------------------------------------------*/
/* Mirrors */
/*--------------------------------------------------*/

UJconst E_reflectivity 0.99995
UJconst I_reflectivity 0.986
UJconst I_Losses 430e-6
UJconst E_Losses 20e-6

/* Input */
MIrror   MirI  0  disp_total_I_6.out NULL  NULL  MiSuIf 0. 0. 0.  1. 0. 0
MIsurf   MiSuIf 0.0 0.1 0. 0. 0. I_reflectivity I_Losses

/* End */
MIrror   MirE  0  disp_total_E_6.out NULL  MiSuEb  NULL  3000. 0. 0.  1. 0. 0.
MIsurf   MiSuEb 3450. 0.165 0. 0. 0. E_reflectivity  E_Losses

/* Optics */
IOlaser  laser 0 NULL 1.06e-6 10. NULL NULL 0. .02 .40 NO 0
OPmod    mod  0  laser.oBeam  3  0. 6.268428e6 -6.268428e6 carrier NULL sb1 NULL sb2
NULL
USignal carrier  0.96
USignal sb1 0.196
USignal sb2 -0.196

OPcavity fp  0  mod.oBeam  MiSuIf  MiSuEb  YES  YES
OPdiode dt 0 1. 6.268428e6 NULL fp.tBeam NO
OPdiode dr 0 1. 6.268428e6 NULL fp.rBeam NO

/* Force computation */
/* Acting conditions (starting time) */
USif starting_time 0 time.out > 0 true false              /* feedback - z  starting time */

USignal corr 0
USignal speed 0

USmultiply  errorsignal 0 corr starting_time.out
USadder zLock 0 1 errorsignal.out -3.e8 /* Sign - UG @ 50 Hz*/

/* ADC-->GC-->DAC delay 300 microsec (3 clock cycles) */
USdelay  zLock_delayed 0 zLock.out 300.e-6

/*------------------------------------------------------------*/
/* actuators for longitudinal locking */
/*------------------------------------------------------------*/

MIact force 0 zLock_delayed.out  3.7e3  20  10   0.0e-9  2  20.  3.  11.1  480. 9.85e-3

USfilter2 disp_force_filtered force.out 0 0.0 0.0 5e-2 7.03619e-2 2.6525e-6 1.0
/* SIMPLE PENDULUM */
USfilter2 disp_seismic_filtered_E pE.dxyzt.s2 0 0.0 0.0 1.0 7.03619e-2 2.6525e-2 1.0

USadder disp_total_E 0 2 disp_force_filtered.out  disp_seismic_filtered_E.out 1. 1.

US6Set disp_total_E_6 0 NULL NULL disp_total_E.out  NULL NULL NULL

USfilter2 disp_seismic_filtered_I pI.dxyzt.s2 0 0.0 0.0 1.0 7.03619e-2 2.6525e-2 1.0
US6Set disp_total_I_6 0 NULL NULL disp_seismic_filtered_I.out  NULL NULL NULL

USadder length 0 2 MirI.dxyzt.s2 MirE.dxyzt.s2 -1 1

UHplotTime 1  100  "dt DC"    duration2  1 0  dt.dc  NULL
UHplotTime 1  201  "dr PHASE" duration2  1 0  dr.phase  NULL
UHplot1D 0  299  "Length" 1000 -1.e-5 1.e-5 length.out 0 10000000 AUTO
UHplotTime 1  300  "Length" duration2  1 0  length.out NULL
UHplot1D 0  398  "Speed" 1000 -1.e-5 1.e-5 speed 0 10000000 AUTO
UHplotTime 1  399  "Speed" duration2  1 0  speed NULL
UHplotFFT  0  301  "LFFT" 17 1 0 length.out NULL
UHplotFFT  0  302  "zFFT" 17 1 0 zLock.out NULL
UHplotFFT  0  304  "fFFT" 17 1 0 corr NULL
UHplotFFT  0  305  "fFFT" 17 1 0 disp_force_filtered.out NULL
UHpTfct 0 501 502 "TF" 17 1 0 dr.phase disp_force_filtered.out NULL NULL
```

8. Annex 2: Simulation code

The simulation has been performed using siesta v5r0 and RIOT_Algo v1r7 for the filtering part. The siesta.c file (userSim routine) has been modified in order to handle the filters implemented in Gc. Here is the code of userSim.

```
/*-----------------------------------------------------------------*/
void userSim()
/*-----------------------------------------------------------------*/
{
  static bool First = true;
  static struct USignal *dt_dc;
  static struct USignal *dr_p;
  static struct USignal *corr;
  static struct USignal *the_length;
  static struct USignal *the_speed;
  static int cpt = 0;
  static int cpt_lock = 0;
  static int locked = 0;
  static double inv_optical_gain;
  static Gc_Filter filter;
  static double speed = -1;
  static bool speed_computed = false;
  static double prev_length = 0;

  static FILE *output;

  double length;
  double correction;

  if( First == true )
    {
     First = false;
     dr_p = USignalFind("dr.phase");
     if (dr_p == NULL)
           UJError(3,"userSim","dr_p find failed");
     dt_dc = USignalFind("dt.dc");
     if (dt_dc == NULL)
           UJError(3,"userSim","dt.dc find failed");
     corr = USignalFind("corr");
     if (corr == NULL)
           UJError(3,"userSim","corr find failed");
     the_length = USignalFind("length.out");
     if (the_length == NULL)
           UJError(3,"userSim","the_length find failed");
     the_speed = USignalFind("speed");
     if (the_speed == NULL)
           UJError(3,"userSim","the_length find failed");
     // Compute optical gain
     inv_optical_gain = .38e-10;

     // Define controller : exactly the one used in Cascina for Virgo arm locking
     double zeros_freq[ 2 ];
     double zeros_Q[ 2 ];
     int nb_zeros = 2;
     double poles_freq[ 3 ];
     double poles_Q[ 3 ];
     int nb_poles = 2;

     zeros_freq[ 0 ] = 1;
     zeros_Q[ 0 ] = 0.;
     zeros_freq[ 1 ] = 10;
     zeros_Q[ 1 ] = .7;

     poles_freq[ 0 ] = 1.e-4;
     poles_Q[ 0 ] = .7;
     poles_freq[ 1 ] = 800.;
     poles_Q[ 1 ] = .7;

     filter.define_filter( nb_zeros, zeros_freq, zeros_Q,
                           nb_poles, poles_freq, poles_Q);
     filter.set_sampling_frequency( 10000 );
     filter.init_filter();
     filter.set_gain_at_given_frequency( 1.,
                                          50 );
     filter.print_coeff();
    }
  // Trigger level

  // Compute lengths
  length = dr_p->value * inv_optical_gain;

  if( cpt < 10. * 10000. )
    {
     corr->value = 0;
     cpt ++;
     speed = -1;
     speed_computed=false;
     the_speed->value = (the_length->value - prev_length) * 10000.;
     prev_length = the_length->value;
     return;
    }

  // Compute Correction
  if( dt_dc->value > .01 )
```

```
   {
     if( speed_computed == false )
         {
           speed = the_length->value - prev_length;
           speed *= 10000 / 1.e-6; //Speed in micrometers/s
         }

     correction = filter.filter( length );
     corr->value = correction;
     cpt_lock ++;
   }
 else
   {
     filter.reset_history();
     corr->value = 0;
     cpt_lock = 0;
     speed_computed = false;
   }

 the_speed->value = (the_length->value - prev_length) * 10000.;
 prev_length = the_length->value;

 // Locked during 1 sec
 if( cpt_lock > 10000 )
   {
     locked = 1;
   }


 // Unlock after 60 sec
 if( cpt > 70. * 10000. )
   {
     if( locked == 1 )
         {
           nb_ok ++;
           printf("Lock %d OK (OK=%d/%d) after %g sec, speed=%g    \n",
                  nb_try,nb_ok,nb_try,60.-cpt_lock/10000.,speed);

           output = fopen("result.dat","a");
           fprintf(output,"%d 1 %g %g\n",nb_try,60.-cpt_lock/10000.,speed);
           fclose( output );
           mean_time += 60.-cpt_lock/10000.;
         }
     else
         {
           printf("Lock %d failed (OK=%d/%d) after %g sec, speed=%g    \n",
             nb_try,nb_ok,nb_try,60.-cpt_lock/1000.,speed);
           output = fopen("result.dat","a");
```

```
          fprintf(output,"%d 0 %g %g\n",nb_try,60.-cpt_lock/10000.,speed);
          fclose( output );
         }

      cpt = 0;
      filter.reset_history();
      corr->value = 0;
      nb_try ++;
      locked = 0;
      cpt_lock = 0;
      speed_computed = -1;
      return;
     }


  cpt ++;
 return;
}
```