Piezo Iterative Learning Control - Implementation Note

Bo Bernhardsson [*]

June 22, 2020

Abstract

This note further explains the Piezo control method proposed and described in [1] and gives further implementation guidance. The method can be used for Lorentz-force compensation for the long pulses with high beam current in the ESS accelerator.

1 Background

To maintain high efficiency, RF cavities need to operate close to resonance. This is especially challenging for superconducting cavities since their resonance frequency can change considerably during operation. Superconducting cavities are made with thin walls to allow them to be kept cool and to reduce cost. The thin walls make the cavities deform easily. The main deformation force comes from the accelerating electromagnetic field on the cavity walls, the so called Lorentz force. There can also be pressure variations in the surrounding cooling system and vibrations from external mechanical sources inducing deformations. For ESS the Lorentz force is assumed to be the major source of resonance frequency variations.

The method explained in this note was presented in [1]. We will here focus on implementation aspects, and refer to the original note for further description of the algorithm.

We will change one assumption from what was made in [1]:

[*]Department of Automatic Control, Lund University. Email: bob@control.lth.se

- We allow for a nonzero beam current, to facilitate operation during normal conditions of the accelarator. In the previous report the beam current was for simplicity assumed zero.

We will assume that an approximation of the complex-valued baseband amplitude of the beam (i.e. magnitude and phase) is available from previous calibration.

2 Contents of this report

The report describes code for several aspects needed for implementing and evaluating the Piezo ILC. The Matlab code, and some Julia code for time critical parts, is given in a github repository. The following code is provided:

1 Cavity field simulator, including model of Lorentz Force Detuning (LFD) and Piezo mechanical dynamics.
   - `initsystem.m` Sets system and simulation parameters
   - `cavitysimulator.m` Simulation of the cavity with LFD
   - `runilc.m` Runs simulation of the full ILC algorithm

2 System identification and ILC filter design
   - `ident01.m` Identification of cavity parameters from time domain data
   - `piezoident.m` Identification of linear transfer function for piezo dynamics
   - `ilcfilterdesign.m` Calculates parameters of the ILC filter (L), an approximate low pass inverse of the piezo dynamics

3
   - `estimatedetuning.m` Estimates frequency offset, i.e. the LFD $\delta(t)$
   - `estimatedetuning.jl` Same but in Julia

4
   - `ilcupdate.m` Updates the ILC piezo signal $V^{\text{piezo}}(t)$
   - `ilcupdate.jl` Same but in Julia

2.1 Workflow

The code for 1 is first used offline to simulate the algorithm for training purposes and to find suitable algorithm parameters.

When the cavity is available for measurement, the code for 2 can then be used for identification of the system dynamics. There are other alternatives to perform this step. The code is only included as example of how it can be performed.

The code for 3 and 4 is run during online operation and is more time critical. The frequency offset estimation code could be run either each pulse, or more seldom, e.g. after trigger by an operator. The code for the Piezo ILC update could either be run each pulse, or initialized as an event based on detection of large frequency deviations, or based on human operator input. The details on practical operation of the piezo ILC loop should be developed in cooperation with experienced accelerator personnel, and is not in the scope of this report.

To run a simulation of the Piezo ILC in matlab execute

```
>> runilc
```

3 Cavity field simulator

- `initsystem.m`
- `cavitysimulator.m`
- `runilc.m`

Matlab code is available on github and in the appendix. Since the code will be used offline, and the implementation is therefore not performance critical, the code was not translated to Julia.

To explain the code we repeat the assumed dynamics for the cavity field control, the Lorentz force detuning, and the piezo system. Following standard literature the cavity field dynamics will be written as

$$\frac{d\boldsymbol{V}}{dt} = (-\omega_{1/2} + i2\pi\delta(t))\boldsymbol{V}(t) + \boldsymbol{C}_1\boldsymbol{u}(t) + \boldsymbol{C}_2 I_b(t) \tag{1}$$

$$\frac{d^2\delta_k(t)}{dt^2} + \frac{\Omega_k}{Q_k}\frac{d\delta_k(t)}{dt} + \Omega_k^2\delta_k(t) = \Omega_k^2 K_k^{LFD}|\boldsymbol{V}(t)|^2 + \Omega_k^2 K_k^{\text{piezo}} V^{\text{piezo}}(t) \tag{2}$$

$$\delta(t) = \delta_0 + \sum_{k=1}^{N_{\text{modes}}} \delta_k(t), \tag{3}$$

where $\mathbf{V}$ is the complex valued baseband cavity voltage, $\omega_{1/2}$ is the cavity bandwitdth in rad/s, and $\delta(t)$ the time-varying detuning in Hertz which we want to control by the Piezo ILC algorithm. A nonzero detuning δ corresponds to that the cavity is operating at a frequency offset from its resonance frequency. Furthermore, $\mathbf{u}$ is the LLRF baseband control signal, $I_b(t)$ is a real-valued signal representing the amplitude of the beam (the phase of the beam will be included in $\boldsymbol{C}_2$), and the complex constants $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ represent the gains and phase shifts from cabling and equipment. For further details, see e.g. [2].

For the mechanical system, Ω_k and Q_k describe the mechanical resonances, and K_k^{LFD} and K_k^{piezo} are real constants indicating the gain of Lorentz force detuning and piezo actuator respectively. The offset value δ_0 indicates the frequency offset when the cavity is at rest with $\boldsymbol{V} = 0$ and $V^{\text{piezo}} = 0$.

The method in [1] for finding the piezo control signal did not require that an additional control system keeps the cavity field $\boldsymbol{V}(t)$ under closed loop control. This assumption is useful for commissioning, but is not realistic if there is a beam present, since this would be unsafe. In the present report we will allow for the presence of a nonzero beam.

To illustrate the dynamics, Figure 1 shows some typical waveforms, for a cavity bandwidth of 500 Hz (i.e. $\omega_{1/2} = 1000\pi$ rad/s) and a mechanical mode of frequency 350 Hz with $Q = 3$. The figure illustrates a situation where the LLRF control system is operated in open loop, and set to fill the cavity. No beam is simulated. There is no piezo-control active. The purpose of the figure is to illustrate the possible negative impact of a large frequency offset, if piezo-control would not be active.

Since no measurements of cavity parameters were available the simulation is based on best available estimations. For example, the Q-value of the cavity determines the damping of the mechanical modes and it has been specified for the construction of the cavities that the construction should be sufficiently stiff to damp vibrations from one pulse ends to the next starts.

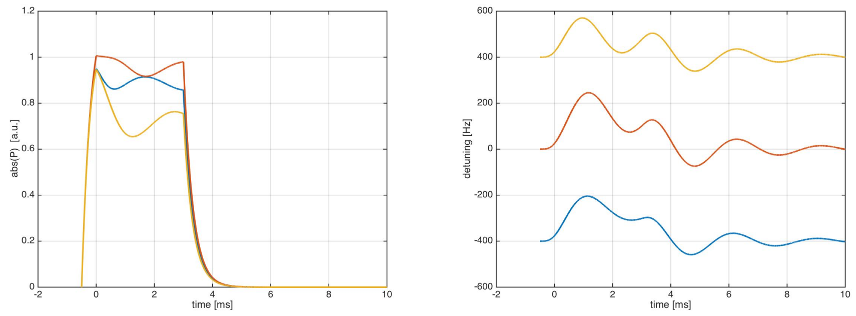

Figure 1: Cavity signal $|\boldsymbol{V}(t)|$ and detuning $\delta(t)$ for 3 different inital detuning offsets. No cavity resonance control is applied. Cavity bandwidth 500 Hz, mechanical resonance 350 Hz. The piezo compensator waveform needs to be flexible to compensate for the Lorentz force detuning when the pulse length is similar or longer than the mechanical time constants.

4 Offline Identification of $\omega_{1/2}$, $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$

- `cavityident.m`

Matlab code is available on github and in appendix. Since the code will be used offline and the implementation is not performance critical, the code was not translated to Julia.

The parameter estimation can be performed during system calibration. To identify $\omega_{1/2}$ and $\boldsymbol{C_1}$ a nonzero, varying, cavity field is needed. To identify $\boldsymbol{C_2}$ a beam is also needed. For proper identifiability, sufficiently exciting signals need to be used. Constant signals do not give full identifiability of all 5 real parameters ($\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ are complex and contain 2 real parameters each).

The suggested algorithm corresponds to standard least squares and is available in any linear algebra toolbox, e.g. matlab or numpy.

By multiplying (1) with the complex conjugate V^* we can decouple the problem of estimating $\delta(t)$ from determining the cavity bandwidth $\omega_{1/2}$, and the complex gain $\boldsymbol{C_1} = a_1 + ib_1$ of the LLRF control signal, and the beam $\boldsymbol{C_2} = a_2 + ib_2$.

We can find the constants $\omega_{1/2}, a_1, b_1$ without beam ($I_b = 0$) solving the least squares problem

$$\min_{\omega_{1/2}, a_1, b_1} \left| \mathrm{Re}\left(\boldsymbol{V}^* \frac{d\boldsymbol{V}}{dt} \right) + \omega_{1/2} \boldsymbol{V}^* \boldsymbol{V} - a_1 \, \mathrm{Re}(\boldsymbol{V}^* \boldsymbol{u}) + b_1 \, \mathrm{Im}(\boldsymbol{V}^* \boldsymbol{u}) \right|_2^2 . \tag{4}$$

For parameter identifiability, the signals $\boldsymbol{V}^*\boldsymbol{V}$, $\mathrm{Re}(\boldsymbol{V}^*\boldsymbol{u})$ and $\mathrm{Im}(\boldsymbol{V}^*\boldsymbol{u})$ need to span a three dimensional signal space.

When running with beam we instead need to solve

$$\min_{\omega_{1/2}, a_1, b_1, a_2, b_2} \left| \mathrm{Re}\left(\boldsymbol{V}^* \frac{d\boldsymbol{V}}{dt} \right) + \omega_{1/2} \boldsymbol{V}^* \boldsymbol{V} - a_1 \, \mathrm{Re}(\boldsymbol{V}^* \boldsymbol{u}) + b_1 \, \mathrm{Im}(\boldsymbol{V}^* \boldsymbol{u}) - a_2 \, \mathrm{Re}(\boldsymbol{V}^* I_b) + b_2 \, \mathrm{Im}(\boldsymbol{V}^* I_b) \right|_2^2 \tag{5}$$

Information from a sufficiently exciting trajectory $V(t), u(t), I_b(t)$ of order 5 must be available. Better estimation accuracy can be obtained by using longer data sets, as long as the parameters remain constant. A suitable LP filtering of signals could be performed to minimize impact of noise. A simple first order filter will probably suffice. The bandwidth of it should be an order of magnitude larger than the cavity bandwidth, not to influence the result too much.

To solve the LS problem we sample (4) using N equidistant time steps and find the LS-solution to $y = A\theta$ where

$$y = \begin{bmatrix} \mathrm{Re}(\boldsymbol{V}^* \frac{d\boldsymbol{V}}{dt})(t_1) \\ \vdots \\ \mathrm{Re}(\boldsymbol{V}^* \frac{d\boldsymbol{V}}{dt})(t_N) \end{bmatrix}$$

$$A = \begin{bmatrix} -\boldsymbol{V}^*\boldsymbol{V}(t_1) & \mathrm{Re}(\boldsymbol{V}^*\boldsymbol{u})(t_1) & -\mathrm{Im}(\boldsymbol{V}^*\boldsymbol{u})(t_1) & \mathrm{Re}(\boldsymbol{V}^*I_b)(t_1) & -\mathrm{Im}(\boldsymbol{V}^*I_b)(t_1) \\ \vdots & \vdots & \vdots & & \\ -\boldsymbol{V}^*\boldsymbol{V}(t_N) & \mathrm{Re}(\boldsymbol{V}^*\boldsymbol{u})(t_N) & -\mathrm{Im}(\boldsymbol{V}^*\boldsymbol{u})(t_N) & \mathrm{Re}(\boldsymbol{V}^*I_b)(t_N) & -\mathrm{Im}(\boldsymbol{V}^*I_b)(t_N) \end{bmatrix}$$

$$\theta = \begin{bmatrix} \omega_{1/2} \\ a_1 \\ b_1 \\ a_2 \\ b_2 \end{bmatrix}$$

The solution is give by $\hat{\theta} = (A^*A)^{-1}A^*y$. The matrix inverse is done on a real 3x3 matrix (without beam) or 5x5 matrix (with beam). The condition number `cond(A)` gives an indication of identifiability. A large condition number indicate bad identifiability.

Matlab code for both cases are given in the appendix. Interface details for this algorithm will not be discussed further in this report.

5 Offline Identification of the piezo control transfer function

- `piezoident.m`

Matlab code is available on github and in appendix. Since the code will be used offline and the implementation is not performance critical, the code was not translated to Julia.

We will assume that the transfer function from piezo control signal $V^{\mathrm{piezo}}(t)$ to detuning error $\delta(t)$ is approximately linear. For this to be true, we need to operate the piezo actuator within its linear region, and not use too aggressive control signal, running into saturation or rate limitation. If saturations are avoided, the iterative learning controller (ILC) should be able to handle modest deviations from nonlinearity.

In the simulations we have assumed the following dynamical model for the effect of Lorentz force detuning and Piezo actuator operation:

$$\frac{d^2\delta_k(t)}{dt^2} + \frac{\Omega_k}{Q_k}\frac{d\delta_k(t)}{dt} + \Omega_k^2\delta_k(t) = \Omega_k^2 K_k^{LFD}|\boldsymbol{V}(t)|^2 + \Omega_k^2 K_k^{\text{piezo}} V^{\text{piezo}}(t) \tag{6}$$

$$\delta(t) = \delta_0 + \sum_{k=1}^{N_{\text{modes}}} \delta_k(t) \tag{7}$$

The ILC algorithm that we describe will however work even if this model does not hold. The ILC algorithm is not based on identifiying the parameters in (7). It is instead based on obtaining the transfer function $H(i\omega)$ in the relation

$$\delta = Hv^{piezo} + \text{Lorentz force detuning due to } \boldsymbol{V} \tag{8}$$

In the code we use an impulse response experiment to identify $H(i\omega)$, but alternatives with step response or chirp experiments could be considered as alternatives. In (8)

$$v^{\text{piezo}} = V^{\text{piezo}} - V_{\text{offset}}$$

denotes the deviation from the level V_{offset} that would give zero detuning error δ with neglectable Lorentz force detuning i.e. for near-zero cavity voltages. In practice the level $V_{\text{offset}} = -\delta_0/H(0)$ is unknown and must also be determined, preferably through offline experiments (or tracked by the ILC). A simple way to estimate the rest value δ_0 of the detuning offset is to apply a small constant control signal u, measure the stationary value of the cavity field V and use the relation (valid when $I_b = 0$)

$$\delta_0 = \frac{1}{2\pi}\text{Im}\left(-\frac{\boldsymbol{C}_1\boldsymbol{u}}{\boldsymbol{V}}\right)$$

6 Offline calculation of ILC filter L

- `ilcfilterdesign.m`

Matlab code is available on github and in appendix. Since the code will be used offline and the implementation is not performance critical, the code was not translated to Julia.

The filter L in the ILC algorithm, see (10) is obtained as an approximate inverse of the estimated system dynamics $H(i\omega)$ from V^{piezo} to detuning error δ. A suitable approximative inverse L can be obtained in the frequency domain using the formula

$$L(i\omega) = \frac{H^*(i\omega)}{\sigma^2 + |H(i\omega)|^2}.$$

Here the robustification parameter σ limits the gain of L for frequencies where H is small. The parameter ensures that small uncertain values of H do not destroy the result through large incorrect gains in L. This formula for the approximate inverse can be motivated as being the optimally robust filter when H has additive uncertainty.

The filter L is transformed to the time-domain by an IFFT operation. (An alternative would be to implement the ILC algorithm in the frequency domain).

$$L(t) = \mathrm{IFFT}(L(i\omega))$$

The resulting filter is non-causal where the last half of the coefficients represent the non-causal part of the filter. The filter coefficient time-vector L is therefore circular shifted `par.Lshiftindex` time-steps, and cropped to give a shorter filter that ismore efficient to apply.

The transfer functions of $H(i\omega)$, $L(i\omega)$ and $L(i\omega)H(i\omega)$ are illustrated in Figure 2. The time domain filter $L(t) = \mathrm{IFFT}(L(i\omega))$ of length 1000 and its circular-shifted version of length 301 are illustrated in the next figures.

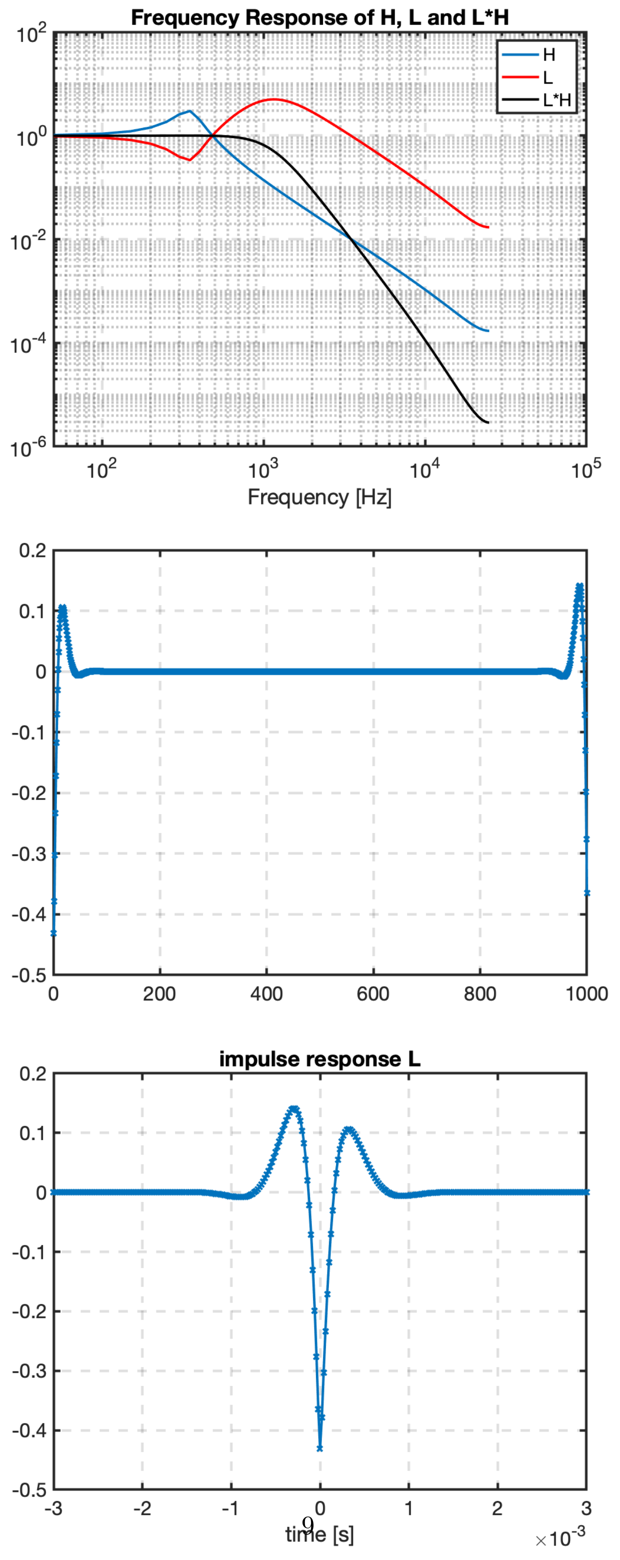

Figure 2: Upper figure: Transfer functions H, L and LH. Note that the L filter gives an approximate inverse, valid up to around 1kHz. Middle figure: IFFT(L) 1000 taps. Lower figure: circular-shifted time domain filter l with 301 taps and a delay of 150 time steps, i.e. 3ms. The inverse filter uses the error signal in the surrounding ± 1 milliseconds to calculate a compensation $L(q)e$ of the error e.

7 Online estimation of frequency offset $\delta(t)$

- `estimatedetuning.m`
- `estimatedetuning.jl`

Matlab and Julia-code is available on github and in appendix.

To find the time varying frequency offset $\delta(t)$ we multiply (1) by the conjugate $V^*(t)$, and use the estimate

$$\widehat{\delta}(t) = \frac{1}{2\pi}\frac{\mathrm{Im}(\boldsymbol{V}^*\frac{d\boldsymbol{V}}{dt} - \boldsymbol{V}^*\boldsymbol{C_1}\boldsymbol{u} - \boldsymbol{V}^*\boldsymbol{C_2}I_b)}{|\boldsymbol{V}(t)|^2 + V_{reg}^2}, \tag{9}$$

where V_{reg} is a real-valued regularization parameter that describes how large the cavity voltage $|V|$ needs to be to give reliable estimation of the frequency offset. A typical value could be 5 percent of the flat-top cavity voltage, i.e.

$$V_{reg} = 0.05 \max_t |\boldsymbol{V}(t)|$$

With too small value of V_{reg} the frequency offset estimation becomes noise sensitive for small cavity voltages.

To reduce impact of noise a 1st order filter with cutoff frequency around 2kHz is recommended on the signals $\boldsymbol{V}$, $\frac{d\boldsymbol{V}}{dt}$, u and I_b in (9). In matlab:

```
[B,A]=butter(1,2e3*20e-6*2)
   B =   0.1122    0.1122     % when dt = 20e-6, i.e. 50kHz
   A =   1.0000   -0.7757
Vf = filter(B,A,V)
```

An implementation is given by the recursion

```
Vf(k) = -A(2)*Vf(k-1) + B(1)*V(k) + B(2)*V(k-1)
```

Slightly better result could be obtained with a symmetric zero-phase filter, corresponding to the `filtfilt` command in matlab, but for simplicity we will stick with the standard IIR filter which is simpler to implement.

Using 50kHz sample rate and vectors of length 20ms would give signal vector lengths of 1000. This means that these filtering operations would correspond to 1000*4*3 = 12000 complex MAC operations for each pulse. To this comes around 6000 complex MAC operations and 1000 real divisions for (9). A final first order LP filter is finally applied to $\widehat{\delta}$ resulting in an additional 3000 real MAC operations. My estimate is that all these calculations would take less than 0.1ms running on a single core on my laptop.

8 Iterative Learning Control

- `ilcupdate.m`
- `arfilter.jl`
- `filtertest.jl`

Matlab and Julia-code is available on github and in appendix. The iterative learning control is based on an iteration of the form

$$V_{k+1}^{\text{piezo}}(t) = Q(q)(V_k^{\text{piezo}}(t) + \kappa_1 \cdot L(q)e_k(t)), \tag{10}$$

where k is iteration index. Here $e = \delta_{ref} - \widehat{\delta}_{ref}$ is the detuning error, the filter Q is one for low frequencies where ILC is active and perfect compensation is attempted, and Q is close to zero for higher frequencies where ILC should be inactive becuase model error and noise is large. Q was chosen as a 2nd order non-causal zero phase low pass filter (`filtfilt` in matlab), with bandwidth 3kHz.

The non-causal filter L is an approximate system inverse of the dynamics from input v^{piezo} to detuning error δ. The time critical part of the ILC update, calculation of $L(q)e_k(t)$, was implemented in Julia, see filtertest.jl.

The coefficient $\kappa_1 \in (0,1)$ controls the convergence rate. The value 1 corresponds to an aggressive one-step correction of error, whereas a lower value gives a more robust algorithm at the price of taking more iterations to converge. The value $\kappa_1 = 0.5$ has been used with good results in the simulations.

An extension to the ILC algorithm can be made where also the constant offset value is adjusted in the ILC algorithm

$$V_{k+1}^{\text{piezo}}(t) = Q(q)(V_k^{\text{piezo}}(t) + \kappa_1 \cdot L(q)e_k(t) + \kappa_2 b_k). \tag{11}$$

The last term b_k is a constant value which is an estimate of the piezo bias offset needed to give zero detuning error before the pulse starts. In simluations it was estimated from the initial detuning error, say from the 0.3ms of the pulse. It is unclear if this part will be needed.

9 Simulation Results

Figure 3 shows the result before ILC is turned on. There is a large detuning offset of several hundred Hz, resulting in inefficient cavity operation. Some results after successful iterative learning control of the Piezo control compensation pulse are shown in the following figures.

A filling period of 0.3ms was used. The pulse length was 2.86ms. The Lorentz force detuning coefficient K^{LFD} was chosen to give a detuning offset of a couple of hundreds of Hz. Different levels of initial frequency offsets in the range -1kHz to 1kHz were tested. The figures below use a -100Hz initial offset. A cavity bandwidth of 500Hz was assumed and signals were represented at 50kHz. Different noise levels were tested, the figures below use 60dB SNR. Reasonable performance was obtained also with significantly lower SNR.

In the simulations the true values of cavity bandwidth and complex calibration constant $\boldsymbol{C_1}, \boldsymbol{C_2}$ were used. Reasonable performance is obtained with slightly wrong estimations of $\boldsymbol{C_1}, \boldsymbol{C_2}$.

A mechanical resonance with one mode with frequency 350Hz and Q-value 3 was used in the simulations, but the code has been verified to work also with more modes.

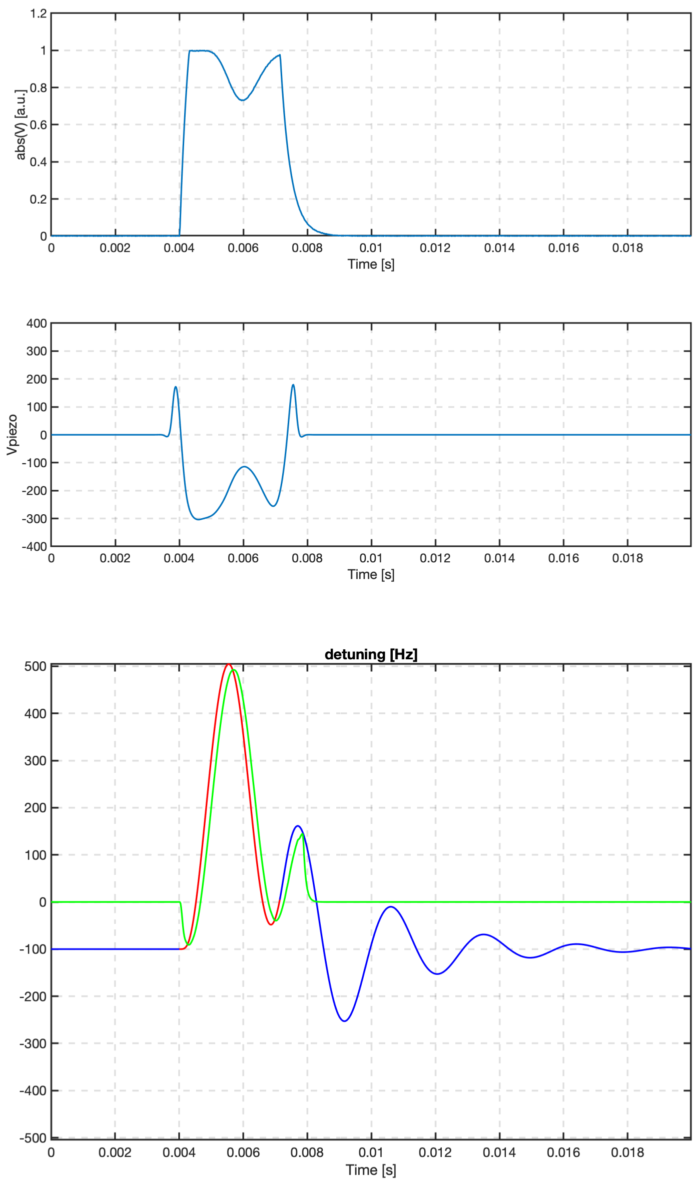

Figure 3: Before ILC, using $V^{\text{piezo}} = 0$. Upper plot shows cavity field magnitude $|\boldsymbol{V}(t)|$ and middle the learned piezo compensation signal $V^{\text{piezo}}(t)$ after one iteration, which will be applied in iteration 2. Lower plot shows true detuning (blue/red) and detuning estimation (green). Initial offset is -100Hz, and the Lorentz force detuning results in a large increase in detuning during the RF pulse. The red part corresponds to the pulse. Simulation with SNR=60dB

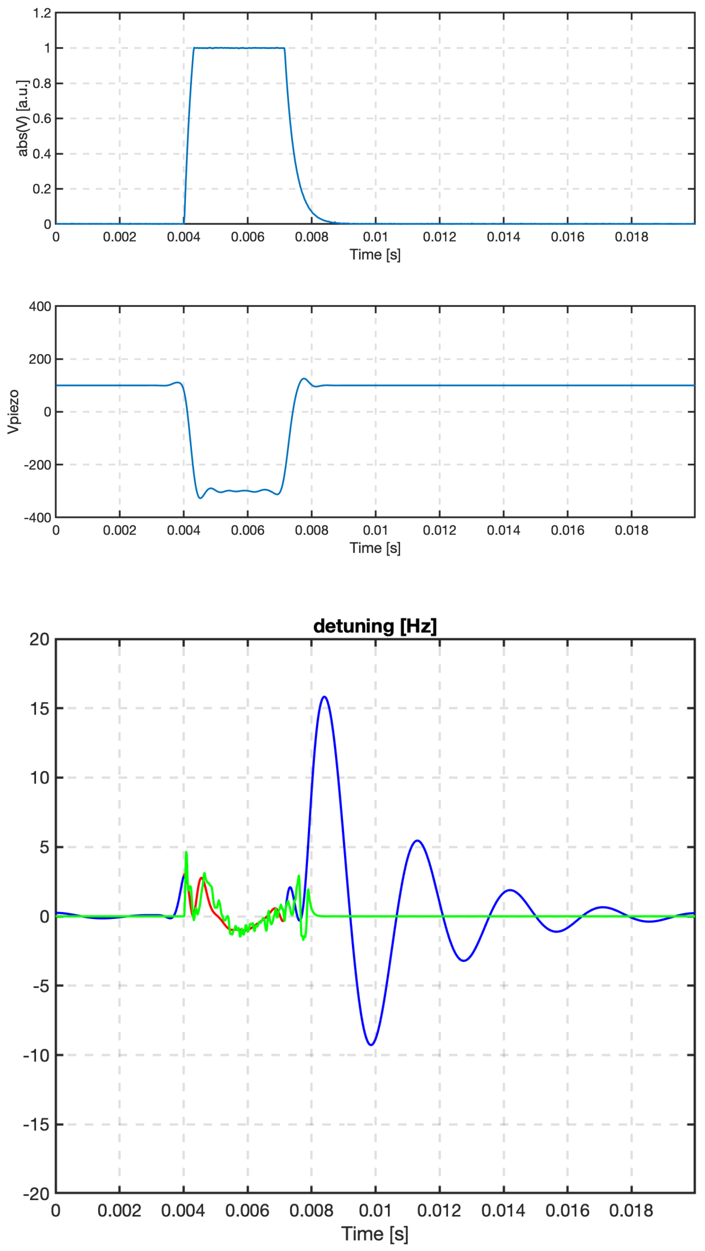

Figure 4: Same as previous figure, but after 20 ILC iterations. ILC has successfully resulted in a detuning error below 1Hz. Simulation with SNR 60dB.

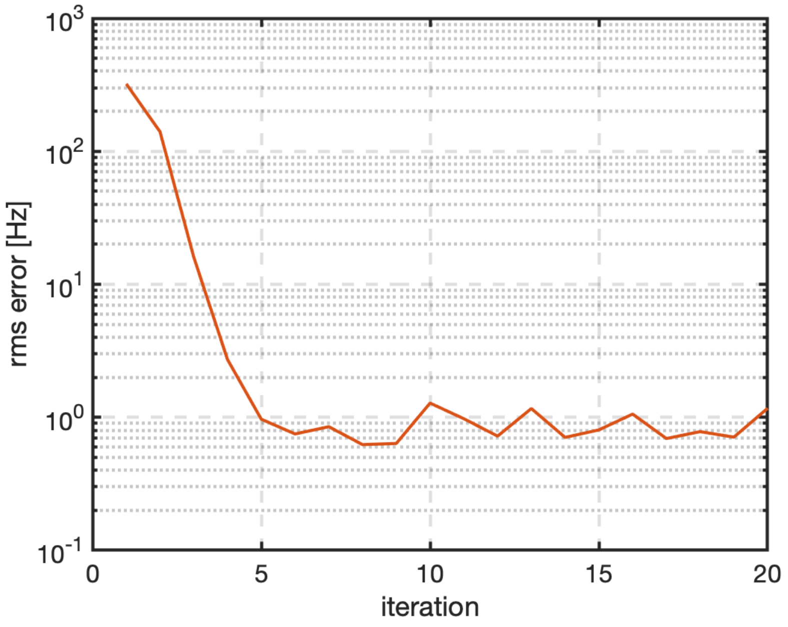

Figure 5: Rms detuning error versus iteration number. After 5 iterations the rms error is kept around 1Hz during the pulse. Simulation with SNR 60dB.

10 Data Shuffling and Computational Effort

It seems sufficient to run the piezo algorithm at a sampling rate of 50kHz, and to collect pulse data for 20ms near the applied pulse. The interval should start some milliseconds before the filling of the cavity, since the piezo controller preferably starts operation sligthly ahead of the filling of the cavity.

10.1 Interfacese LLRF - IOC/Epics

The scheme means that 14 times per second, one needs to send 1000 samples for each signal. In the following table "up" denotes the direction from LLRF and ”down” the direction to LLRF.

name	signal	type	rate	length	nr values	direction
V	cavity measurement	complex	50kHz	20ms	2*1000	up
u	LLRF control signal	complex	50kHz	20ms	2*1000	up
V^{piezo}	piezo control signal	real	50kHz	20ms	1000	down
CTRL-data	TBD	TBD	14Hz	TBD	small	up
CTRL-data	TBD	TBD	14Hz	TBD	small	down

10.2 Computation time

The most time critical operation is the execution of the ILC filter L. This operation was implemented in Julia, see the code `filtertest.jl`. In the simulations a filter with 301 taps on the real-valued detuning error $e(t)$ of length 1000 took around 0.24 ms to run on a MacBook Pro with a 2.7GHz Intel Core i7 (code was run on a single core). The rest of the filtering operations are of order 1 or 2 and are therefore much cheaper. This is true even if zero-phase forward backward filters (corresponding to matlab's `filtfilt.m`) would be chosen. A 2nd order AR filtfilt operation on 1000 real samples takes around 0.01ms in Julia.

The operations in estimatdetuning.m and ilcupdate.m should together be feasible to do within a total time of 2 ms.

If time should be squeezed even further, it could be enough with 10 ms of data, and a sample rate of 25kHz. These changes would reduce the signal lenghts to 250. The reduction of sample rate would also lead to a shorter L filter. The filtering run time would therefore be reduced by more than a factor 4.

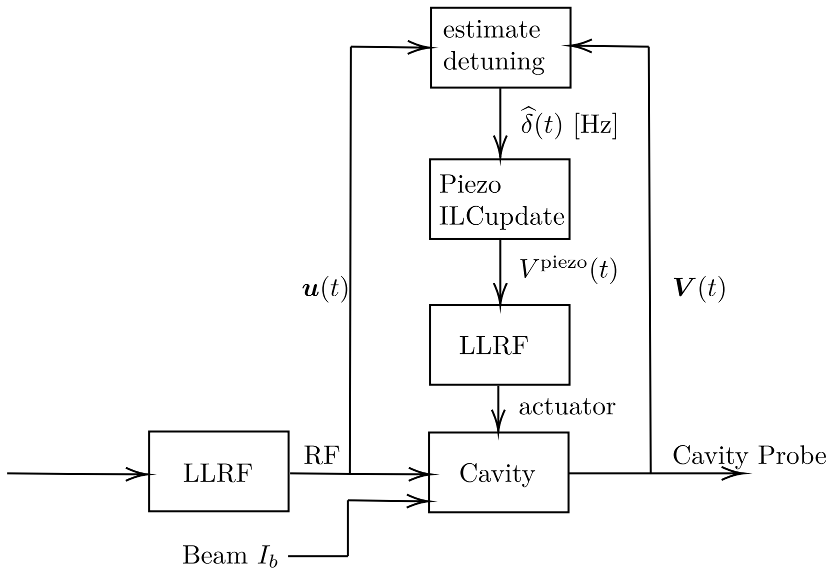

Figure 6: Blockdiagram.

11 Limitations and further work

To reduce the complexity of the investigation of the piezo ILC controller, the present simulator models the cavity field controller output signal as given by a fixed vector $\boldsymbol{u}(t)$. I.e. the cavity field LLRF PI controller and the LLRF cavity field ILC controller have not been modelled. If the Piezo-control ILC should be allowed to run concurrently with the LLRF cavity field ILC controller, a more extensive investigation should be performed to make sure they do not interact destructively. The present author thinks it is a good idea to only allow one of the ILC loops to be active at the same time, at least initially until further expericence of the true system has been gained.

12 Acknowledgement

I am grateful to Olof Troeng and Anders J Johansson for helpful comments on the manuscript.

References

[1] Bernhardsson, B., "Piezo Control Method and Initial Test Plan", *Internal Report*, June 2017.

[2] Troeng, O., "Cavity Field Control in Linear Particle Accelerators", *PhD Thesis*, 2019.

13 Appendix

13.1 initsystem.m

```
function par = initsystem(simulationfrequency)
% Initialize parameters for simulation
%
% Cavity Parameters
% The KLFD, Kpiezo, Omega and Q vectors describe the Piezo mechanical modes

par.cavity.whalf = 2*pi*500;           % Cavity half bandwidth in rad/sec
par.cavity.cavphase = 0;               % Cavity phase in radians
par.cavity.C1 = par.cavity.whalf*exp(i*par.cavity.cavphase); % A.u.
par.cavity.C2 = par.cavity.C1;         % Assumed equal to C1 for simplicity
par.cavity.KLFDvector = [400];         % Lorentz force detuning constants per mode
par.cavity.Kpiezovector = [1];         % Piezo force constants Hz/Volt. A.u.
par.cavity.Omegavector = 2*pi*[350];   % Mechanical modes in rad/sec
par.cavity.Qvector = [3];              % Q-factors of modes
par.cavity.offsetHz = (-100);          % Piezo offset in Hz for Vpiezo=0
par.cavity.nrmodes = length(par.cavity.KLFDvector); % Nr of mechanical modes
if length(par.cavity.Kpiezovector) ~= par.cavity.nrmodes
    error('wrong number of modes in Kpiezovector')
end
if length(par.cavity.Omegavector) ~= par.cavity.nrmodes
    error('wrong number of modes in Omegavector')
end
if length(par.cavity.Qvector) ~= par.cavity.nrmodes
    error('wrong number of modes in Qvector')
end

%
% Simulation Parameters
% We assume to fill the cavity to the (normalized) level V = 1
par.dt = 1/simulationfrequency;         % Piezo sample time, e.g. 20us
par.tmax = 20e-3 - par.dt;              % Piezo pulse length, here made shorter than 1/14 se
par.tvec = 0 : par.dt : par.tmax;
par.nt = length(par.tvec);
par.fmax = 1/par.dt;                    % Piezo sample rate, e.g 50kHz
par.df = 1/(par.nt*par.dt);
par.fvec = 0 : par.df : par.fmax-par.df;
par.tstart = 4e-3;                      % Start of filling, e.g. 5ms in
par.tfill = 0.3e-3;                     % Fill time, e.g 0.3ms
par.tpulse = 2.86e-3;                   % Beam pulse length, e.g 2.86ms
par.tend = par.tstart + par.tfill + par.tpulse;  % time for end of pulse
par.ufill = 1.638*exp(-i*par.cavity.cavphase);   % LLRF signal during filling of cavity, ass
par.beam = 0*(-0.3-0.2*i);              % Complex valued beam,  assumed constant
par.uduringbeam = exp(-i*par.cavity.cavphase) - par.beam; % LLRF signal during beam, perfect
par.options = odeset('RelTol',1e-6,'AbsTol',1e-5*ones(1,2*par.cavity.nrmodes+1)); % simu par
par.noiselevel = 0.001;                 % Noiselevel on u and V, 0.001 = -60dB
%
% Algorithm Parameters
par.initialmeasuretime = 0.2e-3;        % Initial part of pulse used for offset estimation
par.tdecay = 1e-3;                      % Pulse decay used for estimation
par.Lskip = 150;                        % Noncausal part of L-filter, nr of samples
par.Lfilterlength = 301;                % Length of L-filter, nr of samples
par.bw_ilc = 3e3;
par.ilckappa1 = 1;                      % ilc gain for detuning error e(t)
par.ilckappa2 = 0;                      % ilc gain for offset bias b
par.deltaref = 0;                       % setpoint for detuning [Hz]
```

13.2 cavitysimulator.m

```
function [x,delta] = cavitysimulator(xinit,par)
% Cavitysimulator, noise free. Add noise outside
% xinit: initial statevector [V(0);delta1(0); delta1dot(0);...]
% par: Initialize parameters with initsystem.m
global u vpiezo Ib
[t,x] = ode45(@(t,x) cavityprocess(t,x,par) ,par.tvec, xinit, par.options);
delta = par.cavity.offsetHz + sum(x(:,2:2:end),2);
end


function dx = cavityprocess(t,x,par)
global u vpiezo Ib
% function dx = process(t,x,par)
% t: time (scalar)
% x: state vector for cavity + detuning dynamics
% x(1) = V
% x(2k)= delta_k(t)
% x(2k+1) = deltadot_k(t)
% par.offset: Piezo voltage vpiezo = 0 gives detuning = offset in Hz in stationarity

% Dynamics
dx = zeros(par.cavity.nrmodes+1,1);    % a column vector
% Interpolate forward power u and piezo voltage vpiezo at time t
ut = lininterp1(par.tvec,u,t);
% Forward power, a.u.
vpiezot = lininterp1(par.tvec,vpiezo,t);  % Piezo voltage, a.u.
Ibt = lininterp1(par.tvec,Ib,t);          % Beam, a.u.
%
delta = par.cavity.offsetHz + sum(x(2:2:end));   % Offset in Hertz
dx(1) = -(par.cavity.whalf-1i*2*pi*delta)*x(1) + par.cavity.C1*ut + par.cavity.C2*Ibt;
% Mechanical dynamics, here 2nd order dynamics with nrmodes
for k = 1:par.cavity.nrmodes
    Omegak = par.cavity.Omegavector(k);
    Qk = par.cavity.Qvector(k);
    KLFDk = par.cavity.KLFDvector(k);
    Kpiezok = par.cavity.Kpiezovector(k);
    dx(2*k) = x(2*k+1);
    dx(2*k+1) = - Omegak/Qk*x(2*k+1) - Omegak^2*x(2*k) + Omegak^2*KLFDk*abs(x(1)^2) + Omegak
end
end


function v = lininterp1(X, V, x)
```

```
% linear interpolation, given set of X and V values, and an x query
% assumes X values are in strictly increasing order
%
% Differences from matlab built-in :
%       much, much faster
%       if coordinate is exactly on the spot, doesn't look at neighbors.  e.g. interpolate(
%       extends values off the ends instead of giving NaN
%
if length(X) ~= length(V), error('X and V sizes do not match'); end
pindex = find((x >= X), 1, 'last');
index = find((x <= X), 1, 'first');
if isempty(pindex)
    warning('interpolating before beginning');
    pindex = index;
    slope = 0;
elseif isempty(index)
    warning('interpolating after end');
    index = pindex;
    slope = 0;
elseif pindex == index
    slope = 0;
else
    Xp = X(pindex);
    slope = (x - Xp) / (X(index) - Xp);
end
v = V(pindex) * (1 - slope) + V(index) * slope;
end
```

13.3 cavityident.m

```
% Identifies Cavity parameters
% dV/dt = -whalf * V + 2*pi*delta*V + C1*u + C2*Ib
% Where whalf is real and C1 and C2 are complex numbers
% Sufficiently exciting signals u and Ib are needed.
% Constant signals give bad identifiability

global u vpiezo Ib

% Initialize system and simulation parameters
simulationfrequency = 1000e3;
par = initsystem(simulationfrequency);

% Initialize LLRF control , piezo control and beam
u = par.ufill*(par.tstart < par.tvec & par.tvec <= par.tstart + par.tfill) + par.uduringbea
```

```
u = u.';
% Better identifiability with more varying inpout
%u = 0.3*sin(2*pi*100*par.tvec) + i*0.3*sin(2*pi*180*par.tvec) + 0.3*sin(2*pi*250*par.tvec)
%u = u.';

vpiezo = 0*ones(size(par.tvec))';
Ib = par.beam*(par.tstart+par.tfill < par.tvec & par.tvec < par.tend);
Ib = Ib.';

% simulate cavity
xinit = zeros(2*par.cavity.nrmodes+1,1);
[x,delta] = cavitysimulator(xinit,par);
V = x(:,1);


bw = 100e3; % filter bandwidth in Hz
[b,a] = butter(1,bw*2*par.dt);      % Find LP-filter parameters
dVdt = (V(3:end)-V(1:end-2))/(2*par.dt); % Symmetric diff
dVdt = [0;dVdt;0];
dVdtfilt = filter(b,a,dVdt);    %
% Could possibly filter V, u and Ib also

% Solving least squares problem
if max(abs(Ib))> 1e-8
    % Estimating all parameters by least squares if beam present
    y=real(conj(V).*dVdtfilt);
    A=[-conj(V).*V real(conj(V).*u) -imag(conj(V).*u) real(conj(V).*Ib) -imag(conj(V).*Ib)]
    theta = inv(A'*A)*(A'*y);      % Inverting 5x5 matrix
    whalfhat = theta(1)
    C1hat =  theta(2) + i*theta(3)
    C2hat = theta(4) + i*theta(5)
else
    % Estimating cavity parameters by least squares if beam not present
    y = real(conj(V).*dVdtfilt);
    A = [-conj(V).*V real(conj(V).*u) -imag(conj(V).*u) ];
    theta = inv(A'*A)*(A'*y);      % Inverting 3x3 matrix
    whalfhat = theta(1)
    C1hat =  theta(2) + i*theta(3)
end
```

13.4 piezoident.m

```
function [h,H,Voffsethat] = piezoident(par)
%[h,H,Voffsethat] = function piezoident(simulationfrequency)
```

```
% Identify Piezo transfer function H(iw)
% Uses experiment with impulse response vpiezo(t)
% Alternative could be to use step or chirp experiments
% For real operation, delta1 and delta2 need to be estimated using detuningestim.m

global u vpiezo Ib

% First  experiment
smallamp = 0.1;      % small cavity field to be able to measure detuning
u = smallamp*ones(par.nt,1);
vpiezo = 0*ones(par.nt,1);
Ib = 0*ones(par.nt,1);
xinit = zeros(2*par.cavity.nrmodes+1,1);
[x,delta1] = cavitysimulator(xinit,par);

% Second experiment, impulse response
ind1 = 2;  % dont use ind1=1 since interpolation problem in cavitysimulator
impamp = 100;
vpiezo(ind1) = impamp;
[x,delta2] = cavitysimulator(xinit,par);

% Calculate transfer function
diff = delta2 - delta1;
H = fft(diff)./fft(vpiezo);
h = ifft(H);

% Estimate of Voffset, i.e. Vpiezo value giving zero initial detuning error
Voffsethat = (par.deltaref-mean(delta1))/H(1);   % Assumes error-free detuning estimation
end
```

13.5 ilcfilterdesign.m

```
function [l] = ilcfilterdesign(H,par)
% function [l] = ilcfilterdesign(H,par)
% Calculates timedomain ilcfilter l
% length of filter = par.Lfilterlength
% Filter is noncausal with delay = par.Lskip

sigma = 0.1*abs(H(1));  % lower value gives a more high frequency inverse
Hinv = conj(H)./(sigma^2+abs(H.^2));
l = ifft(Hinv);
l = circshift(l,par.Lskip);
l = l(1:par.Lfilterlength);
end
```

13.6 estimatedetuning.m

```
function [deltahat,delta0hat] = estimatedetuning(par,V,C1est,C2est,Vreg,B,A)
% function [deltahat,delta0hat] = estimatedetuning(V,u,Ib,C1,C2,Vreg)
% estimates the detuning offset deltahat(t), size(deltahat)=[par.nt,1]
% and the constant detuning offset delta0
global u Ib
if nargin <3
    C1est = par.cavity.C1;
    C2est = par.cavity.C2;
end
if  nargin<5
    Vreg = 0.05; % regularization parameter, larger value gives more robust estimates
end
if nargin<6
    bw = 2e3;    % bandwidth in Hz
    [B,A] = butter(1,bw*par.dt*2);
end

% filter V, dV/dt, u and Ib
Vf = filter(B,A,V);
dVf = filter(B,A,[0; V(3:end)-V(1:end-2); 0])/2/par.dt;
uf = filter(B,A,u);
Ibf = filter(B,A,Ib);

d=imag((conj(Vf).*dVf-C1est*conj(Vf).*uf-C2est*conj(Vf).*Ibf))./(Vreg^2+conj(Vf).*Vf)/2/pi;

% Only use deltahat when V is sufficiently large, else put deltahat=deltaref
% Filter to smooth out transients
largeV = (abs(V)>2*Vreg);
deltahat = filter(B,A,largeV.*d + (1-largeV)*par.deltaref);

indinitial = find(par.tstart < par.tvec & par.tvec < par.tstart+par.initialmeasuretime);
delta0hat = mean(deltahat(indinitial));
end
```

13.7 ilcupdate.m

```
function [newvpiezo] = ilcupdate(par,V,l,deltahat,delta0hat,skip)
% function [newpiezopulse,newvoffset] = ilcupdate(par,V,l,deltahat,delta0hat,skip)

global vpiezo
if nargin < 6
    skip = par.Lskip
```

```
end

e = par.deltaref - deltahat;
b = par.deltaref - delta0hat;
changev = filter(l,1,[e; zeros(skip,1)]);
changev = changev(skip+1:skip+par.nt);
changev(1:round((par.tstart-0.2e-3)/par.dt)) = 0;    % only change Vpiezo  near pulse
changev(round((par.tend+par.tdecay/2)/par.dt):end) = 0;

[B3,A3] = butter(2,par.bw_ilc*par.dt*2);    % Q-filter in ILC
newvpiezo = filtfilt(B3,A3,vpiezo  + par.ilckappa1 * changev + par.ilckappa2 * b);
end
```

13.8 runilc.m

```
% Run simulation
global u Ib vpiezo

% Initialize system and simulation parameters
simulationfrequency = 50e3;
par = initsystem(simulationfrequency);
[h,H,Voffsethat] = piezoident(par);
l=ilcfilterdesign(H,par);

% Initialize signals
u = par.ufill*(par.tstart < par.tvec & par.tvec <= par.tstart + par.tfill) +...
    par.uduringbeam*(par.tstart+par.tfill < par.tvec & par.tvec < par.tend);
u = u.';
u + par.noiselevel*max(abs(u))/sqrt(2)*(randn(par.nt,1)+1i*randn(par.nt,1));
Ib = par.beam*(par.tstart+par.tfill < par.tvec & par.tvec < par.tend);
Ib = Ib.';
indpulse = find(par.tstart <= par.tvec & par.tvec <= par.tend); % index for pulse


% Choose alternative for estimation of initial offset:
% 1) Oracle, assuming H correct estimated
 Voffset = (par.deltaref-par.cavity.offsetHz) / sum(par.cavity.Kpiezovector);
% 2) From piezoident
% Voffset = Voffsethat
vpiezo = Voffset * ones(size(par.tvec))';

% Initialize ILC simulation
xinit = zeros(2*par.cavity.nrmodes+1,1);
rmsvalue=[];
```

```
% ILC simulation
for iter = 1:20
    [x,delta] = cavitysimulator(xinit,par);
    rmsvalue(iter) = rms(delta(indpulse)-par.deltaref);
    V = x(:,1);
    V = V + par.noiselevel*max(abs(V))*(randn(par.nt,1)+1i*randn(par.nt,1));
    [deltahat,delta0hat] = estimatedetuning(par,V);

    % plot results
    figure(11);
    subplot(211)
    plot(par.tvec,abs(V));grid on;
    axis([0 par.tmax 0 1.2])
    xlabel('Time [s]');ylabel('abs(V) [a.u.]')
    subplot(212)
    plot(par.tvec,vpiezo);grid on;
    axis([0 par.tmax -400 400])
    xlabel('Time [s]');ylabel('Vpiezo')
    figure(12)
    plot(par.tvec,delta,'b',par.tvec(indpulse),delta(indpulse),'r',par.tvec,deltahat,'g')
    grid on; title('detuning [Hz]'); xlabel('Time [s]')
    yscale = max(20,max(abs(delta)));
    axis([0 par.tmax -yscale yscale]);
    figure(13)
    plot(par.tvec,abs(u))
    grid on; title('LLRF u'); shg

    vpiezo = ilcupdate(par,V,l, deltahat,delta0hat,par.Lskip+5);    % skip+5 even better !
    xinit = (x(end,:)).';      % for next pulse, needed for nonzero offsets
end
figure(20)
semilogy(rmsvalue)
grid on;  ylabel('rms error [Hz]'); hold on
xlabel('iteration')
axis([0 length(rmsvalue) 0.1 1000])
```

13.9 ARfilter.jl

```
const len = 1000
b1 = 0.1311
b2 = 0.2622
b3 = 0.1311
a1 = -0.7478
```

```
a2 = 0.2722
u = rand(Float64,len)
y = zeros(len)

function AR1filt!(u,y,b1,b2,a1)
  y[1] = b1*u[1]
  @inbounds for t = 2:len
    y[t] = -a1*y[t-1] + b1*u[t] + b2*u[t-1]
  end
end

function AR2filt!(u,y,b1,b2,b3,a1,a2)
  y[1] = b1*u[1]
  y[2] = -a1*y[1] + b1*u[2] + b2*u[1]
  @inbounds for t = 3:len
    y[t] = -a1*y[t-1] -a2*y[t-2] + b1*u[t] + b2*u[t-1] + b3*u[t-3]
  end
end

using BenchmarkTools
@btime AR1filt!(u,y,b1,b2,a1)
@btime AR2filt!(u,y,b1,b2,b3,a1,a2)
> 2.706 us (0 allocations: 0 bytes)
> 4.714 us (0 allocations: 0 bytes)
```

13.10 filtertest.jl

```
const len1 = 301
const len2 = 1000
const skip = 150
l = rand(Float64,len1)
u = rand(Float64,len2)
y = zeros(len2)

function bobfilt!(l,u,y)
  for t = 1:len2
    @inbounds for tau = max(1,t+skip+1-len2) : min(t+skip,len1)
         y[t] = y[t] + l[tau]*u[t+skip+1-tau]
     end
  end
end

using BenchmarkTools
@btime bobfilt!(l,u,y)
```

```
> 235.047 us (0 allocations: 0 bytes)
```