Overview

This document explains the current progress on the windkessel code. It saddens Koen that he wasn't able to finish it before he left, but he hopes it will be completed one day.

Additional resources

Need more help?

Check the resources, and then see Ken

Main content

**Part 1: Windkessel Hold**

The windkessel hold will replace the isotonic hold in the C++ code. What the windkessel basically does is it simulates the flow from the aorta into the left atrium, using a electric circuit analogy containing resistors, a capacitor and a inductor.

There are multiple windkessel systems that have been evaluated. The one that has been found to be closest to the actual pressure characteristics is the 4-element windkessel. For this reason this is the circuit that has been chosen to simulate the windkessel hold.

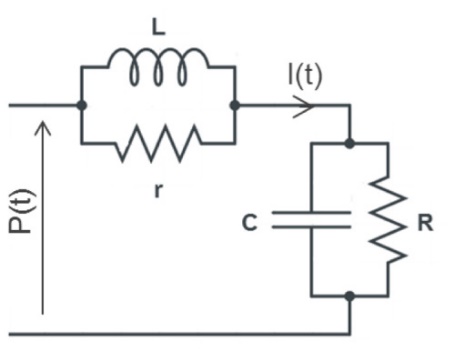
**4-element windkessel**

The 4-element windkessel contains, as the name suggests, four different elements.

- 2 Resistors

- 1 Inductor

- 1 Capacitor



From this setup, the transfer function between input (in this case rate of length change) and the output (force) can be determined. I won't go into detail about that here, but the solution can be found in the enclosed papers.

**State space equations**

From this transfer function, the differential equation between input and output can be determined. However, this solution would be a very long second order differential equation, which would be nigh impossible to solve analytically. A much easier method to solve this would be using state space equations (state space equations explained). This transforms the second order differential equation to a set of first order differential equations, which are much easier to solve. This gives the equations the shape of:



where we have input u(t), system state x(t) and output y(t).

And in this case matrices A, B, C and D look like figure (13) in the paper of Hauzer, *system analyze of the windkessel models*.

However this solution is for a continuous system, and the DAP uses a discrete system. This means that the matrices A and B need to be transformed into their discrete versions. For now suffice to say that these can be determined as is done in the code:

% Create discrete state space matrices Ad and Bd

Ad = expm(A\*dt);

Bd = A\(Ad-eye(length(A)))\*B;

Now for each time step in the DAP system, the input u (in this case the rate of length change dL/dt) is taken. From this, together with the old state x(t), the new state of x'(t) is determined for the next iteration. Secondly, the output (in this case the force level on which the trabec is held) is determined from the current state and the input.

**Initial state**

Since this calculation takes the previous state x of the system, a first state will need to be defined. This first state will contain the initial conditions of the output, so in this case dL/dt(t=0) and d2L/dt^2(t=0).  
x0 = [dL/dt(0)  
        dL/dt^2(0)]  
In the current estimation dL/dt(0) is assumed as 3.4V and dL/dt^2 is assumed starting at 0. If these values are off, it does not affect the end-results of the system, just the first few outputs will be off.

**Part 2: Matlab Code to Determine the Four Element Parameters**

In order to determine the four parameter values of the windkessel, there are three important parts in the code.

1. **Use the correct input characteristic**  
   The input in our case is the rate of length change of the trabec (dL/dt). Thus far the input has been simulated as part of a sinewave, which are currently only the second quarters of each wavelet. This was estimated to be roughly the same shape as the trabec length change, but might not be completely correct, so this needs to be verified.The input is in Volts per second [V/s]. The maximum value of the length change is currently estimated as 15 V/s.
2. **Chose the correct estimates for the four windkessel parameters**  
   The four parameter values of two resistors, one inductor and one capacitor will need to be estimated. It is a bit hard to explain here, so it might just be best to play with each of them a little and see how they influence the output. Since we are talking force here, the units of the resistors would be similar to a damper being N\*s\*/m. The mechanical equivalent of the inductor would be a spring, having units in N/m. And the mechanical equivalent of the capacitor would be a inerter (could be seen as a mass), having units of N\*s^2/m.
3. **Know what your output should look like**  
   Finally, to know if your estimated parameters are correct, you should know what kind of output you would like. Together with Charles was determined that the output should have the characteristic windkessel shape and should remain between 3 and 4 Volts.

**Part 3: C++ Code for the Windkessel Hold**

The major difference with the isotwitch\_PIcontrol C-code will be the change of isotonic\_hold to windkessel\_hold.

if (windkessel\_hold==1)

{

// Calculate rate of fl change in Volts

speed\_fl = ((((double)(last\_out-last\_out\_min1))/32767.0)\*5.0)/0.0002;

printf("%f\n",speed\_fl);

// Write new state to current state

state\_1 = state\_1\_new;

state\_2 = state\_2\_new;

// Calculate windkessel\_level in Volts

windkessel\_level = C1\*state\_1 + C2\*state\_2 + D\*speed\_fl;

windkessel\_int\_level = (short int)((windkessel\_level/5.0)\*32767.0);

// Update the state values

state\_1\_new = A1\*state\_1 + B1\*speed\_fl;

state\_2\_new = A2\*state\_2 + B2\*speed\_fl;

// Change windkessel level back to short int and calculate force error

error\_value = (in\_force\_value - windkessel\_int\_level);

// Update error\_buffer

for (i=0;i<(integral\_points-1);i++)

{

error\_buffer[i] = error\_buffer[i+1];

}

error\_buffer[integral\_points-1] = error\_value;

// Calculate the error\_sum

error\_holder=0;

for (i=0;i<integral\_points;i++)

{

error\_holder = error\_holder + (long int)error\_buffer[i];

}

error\_sum = (short int)(error\_holder/(long int)integral\_points);

if (out\_diff\_accumulator<0)

{

prop\_gain2 = prop\_gain + gain\_asymmetry;

}

else

{

prop\_gain2 = prop\_gain;

}

//Calculates out\_value with PIDcontrol, 100k is a constant that affrcts the gain params.

out\_value = last\_out -

(fl\_polarity \*

((prop\_gain \* error\_value) + (Ki \* (short int)(error\_holder / integral\_points)))

/ 100000);

// Test for breaking out of isotonic hold and into isometric hold

if (fl\_switch==0)

{

// Code breaks out of control loop if out\_diff\_accumulator

// exceeds fl\_threshold. Use this to stop isotonic control

// when the motor reverses direction

// Need to check for both fl\_polarity options

if (((fl\_polarity>0)&&(out\_diff\_accumulator>fl\_threshold)) ||

((fl\_polarity<0)&&(out\_diff\_accumulator<-fl\_threshold)))

{

windkessel\_hold=0;

inter\_ramp=1;

}

}

else

{

// Code breaks out of control loop if motor returns to original value

if (((fl\_polarity>0)&&(out\_value>fl\_threshold)) ||

((fl\_polarity<0)&&(out\_value<-fl\_threshold)))

{

windkessel\_hold=0;

inter\_ramp=1;

printf("%i\n",out\_value);

}

}

    // Update windkessel\_int\_level for next iteration by adding increment

    //windkessel\_int\_level = windkessel\_int\_level + (double)((double)isotonic\_increment/1000.0);

    double\_inter\_holder = last\_out;

}

**Calculate rate of length change**

Calculate the rate of length change and convert this from the binary number into the voltage value (/32767 and \*5 because a binary number of 32767 stand for +5V)

// Calculate rate of fl change in Volts

speed\_fl = ((((double)(last\_out-last\_out\_min1))/32767.0)\*5.0)/0.0002;

printf("%f\n",speed\_fl);

**Copy new state**

Write the new state from the previous loop to the current state.

// Write new state to current state

state\_1 = state\_1\_new;

state\_2 = state\_2\_new;

**Calculate the windkessel level**

The windkessel level is calulated from the current state and rate of length change. As this value is in Volts, this is also converted into integer values.

// Calculate windkessel\_level in Volts and integer values

windkessel\_level = C1\*state\_1 + C2\*state\_2 + D\*speed\_fl;

windkessel\_int\_level = (short int)((windkessel\_level/5.0)\*32767.0);

**Calculate the new states**

The new state values are determined from the previous state and rate of length change.

// Update the state values

state\_1\_new = A1\*state\_1 + B1\*speed\_fl;

state\_2\_new = A2\*state\_2 + B2\*speed\_fl;

**Calculate force error**

The error in force is calculated from the difference between the current force level and the windkessel level (where the force is supposed to be on).

// Calculate force error

error\_value = (in\_force\_value - windkessel\_int\_level);

**Some other important changes**

Except for the read in values from the DAPL code, every instance of isotonic\_level has been replaced with windkessel\_level. Isotonic\_hold has been replaced with windkessel\_hold.

Some parts of the code currently include a printf statement. These should be removed for the final code, but can be used for troubleshooting the system. For printing a value in the DAPL output code, the following statement is used:

printf("%f\n",speed\_fl);

printf("%i\n",out\_value);

%f is used for the Voltage values such as floats or doubles.

%i is used for the integer values such as int, short int and long int.

**Part 4: Current Issues**

* Estimated rate of length change might not be correct, because the speed is off the charts after a few loops. Because of this, if fl\_threshold is not increased, the system exits the windkessel\_hold loop after one loop.
* The C-code was written with the second last version of isotwitch\_PIcontrol. As of such the code for the out\_diff\_accumulator and for calculating the output with PI-control are not completely up to date (and perhaps some other parts...).
* The state space matrix values A1, A2, B1, B2, C1, C2 and D are currently still defined in the C-code. These should be changed into variables as soon as the raw code works (or sooner, should Ken desire so).