Digital filter implementation of the QTA model for Mandarin F0 modeling

Reiner Wilhelms-Tricarico @ Fonix

last update: 4/27/2006

0.1 Another attempt: long and short syllables

After mocking around for a while with the QTA model, we ran into the problem that the method has limitations for handling long durations. So I tried to put together a little bit of a "generalization" that in the limit of short syllable durations (80 ms minimal) basically the same type of control ramps are used as in the original model, but for increasing durations, more complexity is added. I finally arrived at a simple polynomial model with duration dependent parameters, to represent the input $\boldsymbol{u(t)}$ to the model. The method is explained in the following.

We use a third order polynomial in a normalized time $\boldsymbol{t}$,

$$\boldsymbol{v(t) = at^3 + bt^2 + ct + d} \tag{1}$$

The normalized time $\boldsymbol{t}$ is the actual time divided by the duration of the syllable, $\boldsymbol{\delta}$, therefore the normalized $\boldsymbol{t}$ is in the interval $\boldsymbol{[0, 1]}$.

To specify the polynomial coefficients we define values and slopes at the beginning and end of the syllable. Since we are using normalized time, the slopes in the calculation have to be first multiplied with the duration. Later the coefficients of the polynomial are scaled so they correspond to a polynomial for the interval $\boldsymbol{[0, \delta]}$

To obtain the coefficients we solve a simple system based on:

$$\begin{aligned} \boldsymbol{v(0)} &= \boldsymbol{d = v_0} \\ \boldsymbol{v'(0)} &= \boldsymbol{c = v'_0} \\ \boldsymbol{v(1)} &= \boldsymbol{a + b + c + d = v_1} \\ \boldsymbol{v'(1)} &= \boldsymbol{3a + 2b + c = v'_1} \end{aligned}$$

Hence:

$$\boldsymbol{a} = \boldsymbol{v'_1 + v'_0 + 2v_0 - 2v_1} \tag{2}$$

$$\boldsymbol{b} = \boldsymbol{v_1 - v_0 - v'_0 - a} \tag{3}$$

$$\boldsymbol{c} = \boldsymbol{v'_0} \tag{4}$$

$$\boldsymbol{d} = \boldsymbol{v_0} \tag{5}$$

The coefficients (except $\boldsymbol{d}$) need then be scaled, since they where calculated for the normalized time, by:

$$\boldsymbol{a \rightarrow a/\delta^3 \;\; b \rightarrow b/\delta^2 \;\; c \rightarrow c/\delta}$$

The next little feature is the dependence on duration. For this, we first define a scalar that is 0 for the minial duration and 1 for the maximal duration, and use it as an interpolation parameter that allows interpolation between specified parameters for the minimal duration and maximal duration of a syllable. I am using:

$$\boldsymbol{x(\delta) = \frac{2\bar{\delta}}{\bar{\delta} + 1}} \text{ with } \boldsymbol{\bar{\delta} = \frac{\delta - \delta_{min}}{\delta_{max} - \delta_{min}}} \tag{6}$$

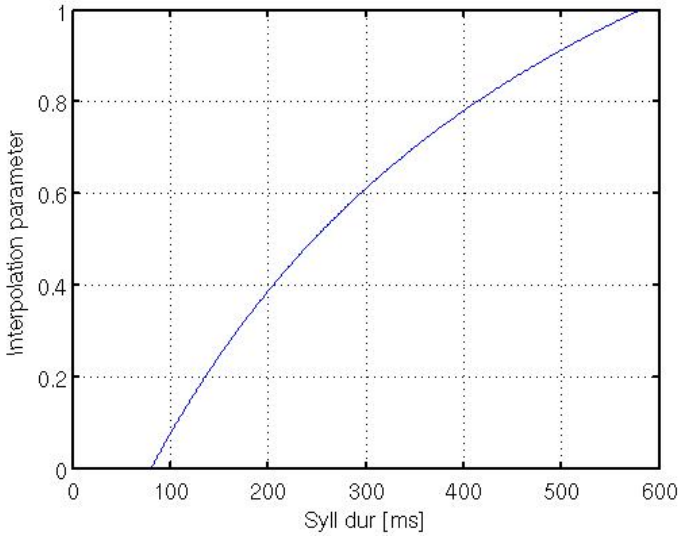

Figure 1: The interpolation parameter $\boldsymbol{x}$ that interpolates parameters for syllable control functions.

Using the specified transformation makes the parameter $\boldsymbol{x}$ move quicker for smaller syllable durations than for large ones, which in turn results in parameters changing quicker with duration for small durations than for large durations. The parameter is shown in Fig. 1 The paramter $\boldsymbol{x}$ is then used to interpolate, depending on the duration of the syllable, the values of the initial and final F0 of the control function and the initial and final slope of the control function.

So to define a range of control functions for one tone, the following parameters need to be specified:

(1) Syllable initial F0 for the shortest duration, $\boldsymbol{v(0)_{min}}$
(2) Syllable initial F0 for the longest duration, $\boldsymbol{v(0)_{max}}$
(3) Syllable initial F0-slope for the shortest duration, $\boldsymbol{v'(0)_{min}}$
(4) Syllable initial F0-slope for the longest duration, $\boldsymbol{v'(0)_{max}}$
- And the same for the final point, indicated with $\boldsymbol{v(1)_{min}}$ etc.

The following parameters were specified:

tone	$v(0)_{min}$	$v(0)_{max}$	$v'(0)_{min}$	$v'(0)_{max}$	$v(1)_{min}$	$v(1)_{max}$	$v'(1)_{min}$	$v'(1)_{max}$
1	**15**	**5**	**1**	**50**	**15**	**0**	**−1**	**−40**
2	**−20**	**−5**	**−10**	**−50**	**30**	**20**	**200**	**50**
3	**−25**	**−15**	**−10**	**−50**	**−25**	**−10**	**10**	**150**
4	**30**	**0**	**0**	**20**	**−50**	**−30**	**−300**	**−150**

(7)

This is taken from the following Matlab statement:

```
switch tone,
    case 1,
    parmsA = [15,5,1,50];
    parmsB = [15,0,-1,-40];
    case 2,
    parmsA =[-20, -5, -10, -50];
```

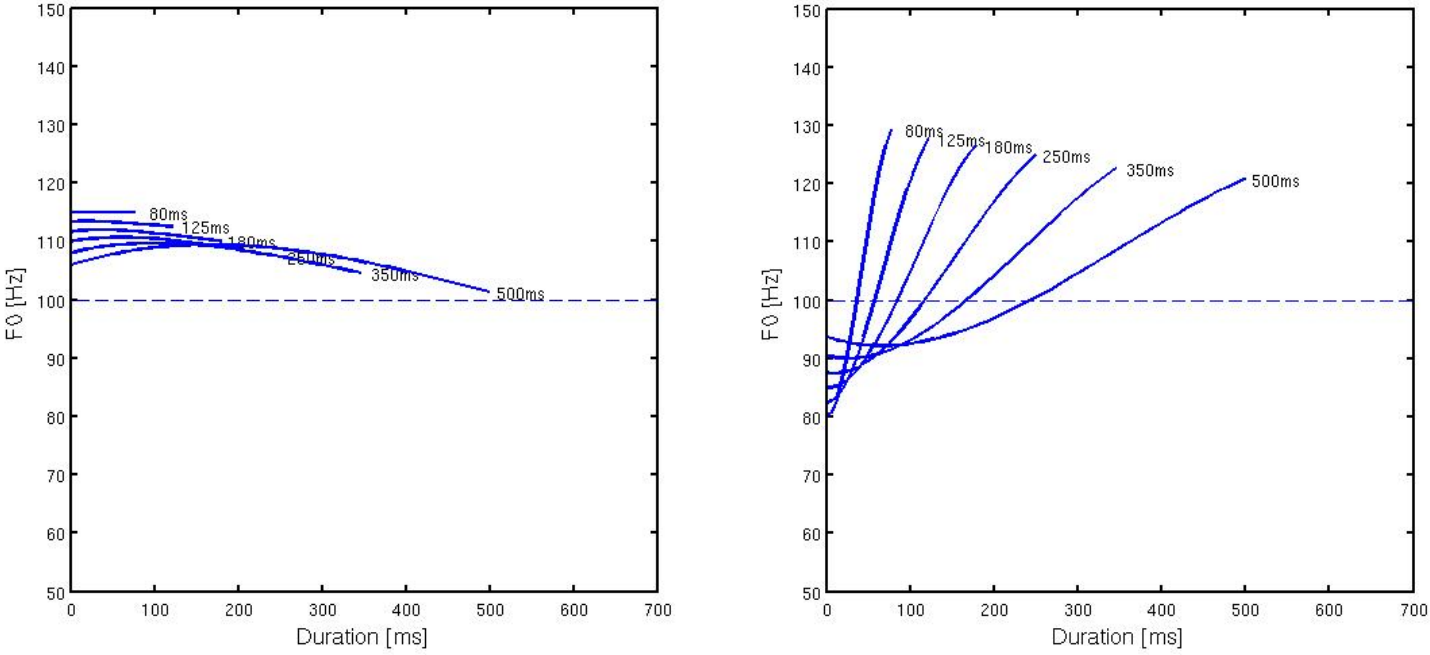

Figure 2: First tone 'H' and second tone 'R'

```
    parmsB =[ 30, 20, 200, 50];
    case 3,
    parmsA = [-25, -15, -10, -50];
    parmsB = [-25, -10,  10,  150];
    case 4,
    parmsA = [30, 0, 0, 20];
    parmsB = [-50, -30, -300,-150];
end;
```

In the figures Fig. 2 and 3, the calculated control functions are shown for a given set of durations (see figures) and using a reference F0 of 100Hz.

Using this type of synthesis, Fig. 4 shows an example.

```
/*
 Input and output: A pointer to a control interval.
   Input:  dur: duration in seconds (e.g., 0.125)
   f0:  offset F0
   a1,  a2:  F0 targets for minimal and maximal duration at the beginning of the syllable.
  da1, da2:  F0 slope targets for minimal and maximal duration at the beginning of the syllab
   b1,  b2:  F0 targets for minimal and maximal duration at the end of the syllable.
  db1, db2:  F0 slope targets for minimal and maximal duration at the end of the syllable.
  omega1, omega2: extreme values for of omega for shortest and longest
   syllable durations.
  damp1 damp2: extreme values for damping ratio for shortest and longests
```

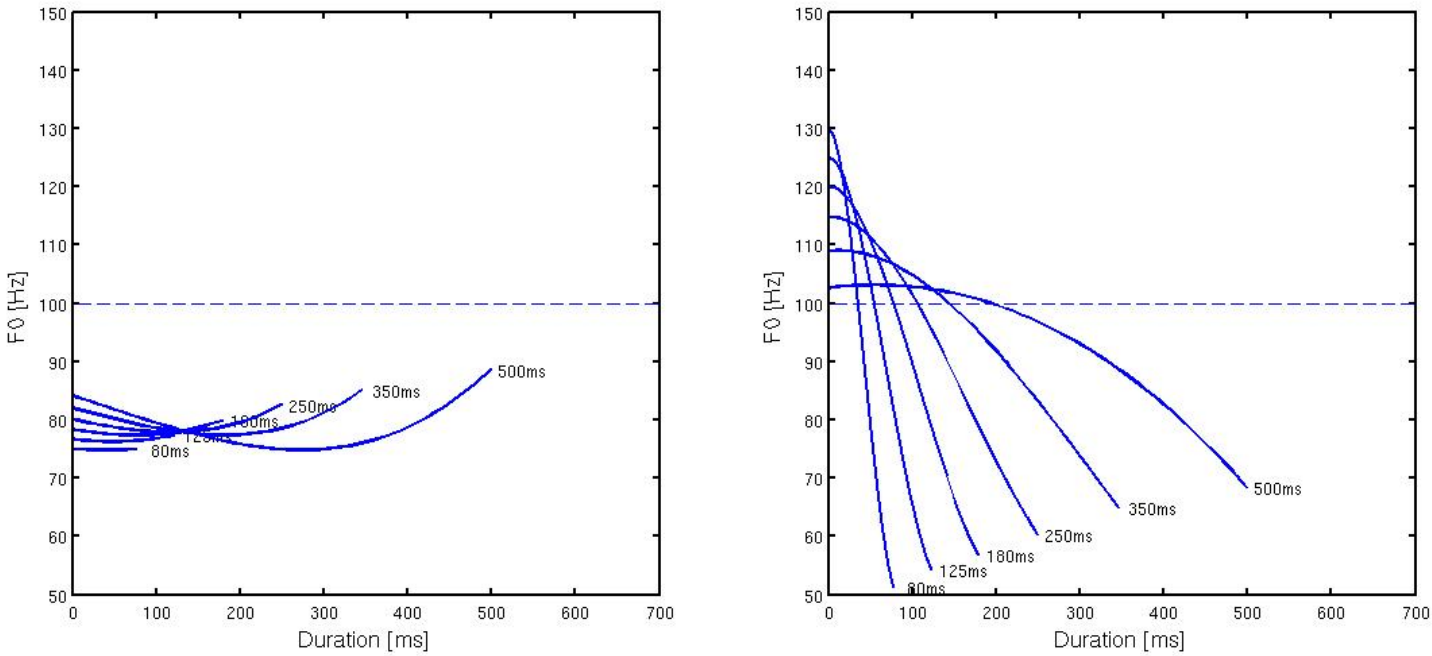

Figure 3: Third tone 'L' and fourth tone 'F'

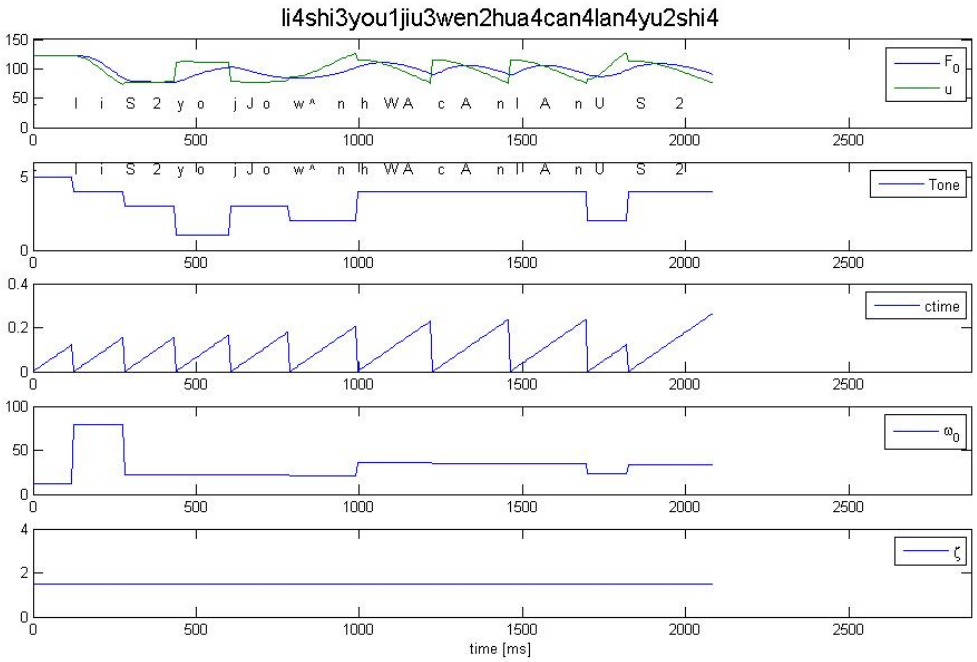

Figure 4: Sythesis example with variable control signals. Note also that there is now final lengthening.

```
   syllable durations.
*/
void variablecubictone(MandF0ControlInterval *I,
                   float dur, float f0,
   float a1, float a2, float da1, float da2,
   float b1, float b2, float db1, float db2,
   float omega1, float omega2, float damp1, float damp2)
{
float x, v0,v0d,v1, v1d;
float a,b,c,d;

x = (dur-0.08)*2.0;
if (x<0.0) x=0.0;
if (x>1.0) x=1.0;

x = 2*x/(x+1);
v0 = (1-x)*a1+x*a2;
v0d = (1-x)*da1+x*da2;
v1 = (1-x)*b1+x*b2;
v1d = (1-x)*db1+x*db2;
v0d = v0d*dur;
v1d = v1d*dur;

d=v0;
c=v0d;
a = v0d+v1d+2*v0-2*v1;
b = v1-v0-v0d-a;

I->a = a/(dur*dur*dur);
I->b = b/(dur*dur);
I->c = c/dur;
I->d = d+f0;
I->omega = (1-x)*omega1+x*omega2;
I->damp = (1-x)*damp1+x*damp2;
I->pdur = dur;
}

void  make_tonal_gestures(MandF0Machine *M, int tone, double duration)
{
    MandF0ControlInterval *I;

switch(tone)
{
case 0:                            // used for final lengthening
I = gimmy_a_new_interval(M);
I->omega = 12.0;        // wild guess.
```

```
I->damp = 2.5;
I->d = -2.0;
I->pdur = duration*MILLISECONDS;
break;
case 1:                          // High tone
I = gimmy_a_new_interval(M);
variablecubictone(I, duration*MILLISECONDS, 0.0,
   15.0 , 5.0 ,  1.0 ,  50.0,
               15.0 , 0.0 , -1.0 , -40.0,
   25.0 , 15.0, 1.5, 1.5 );

break;

case 2:                           // raising tone
I = gimmy_a_new_interval(M);

variablecubictone(I, duration*MILLISECONDS, 0.0,
  -20.0 , -5.0 , -10.0 , -50.0,
   30.0 , 20.0 , 200.0 ,  50.0,
   25.0 , 15.0, 1.5, 1.5 );
break;

case 3:                           // LOW falling raising tone
I = gimmy_a_new_interval(M);
variablecubictone(I, duration*MILLISECONDS, 0.0,
 -25.0, -15.0, -10.0,  -50.0,
 -25.0, -10.0,  10.0,  150.0,
  25.0 , 15.0, 1.5, 1.5 );
break;

case 4:                            // fast falling tone
I = gimmy_a_new_interval(M);
variablecubictone(I, duration*MILLISECONDS, 0.0,
                   30.0, 0.0, 0.0, 20.0,
                  -30.0, -20.0, -250.0,-150.0,
   45.0 , 25.0, 1.5, 1.5 );
break;

case 5:                            // NEUTRAL TONE
I = gimmy_a_new_interval(M);
I->omega = 12.0;        // wild guess.
I->damp = 1.5;
I->a =  0.0;
I->d = -5.0;
I->pdur = duration*MILLISECONDS;
break;
```

. . . .