Development of a Graphical Control Design and Tuning Platform

Aaron Radke

6/3/03
Abstract

This research focuses on building a real time cross-platform graphical tuning application framework for control systems. The objective is to create an interactive environment for controller design and tuning that reduces the difficulty of repetitive tuning with multiple parameters, and allows rapid insight into the effects the parameters in the control system. This is accomplished by creating a simulation tool kit called 'simtk' written in Java. The tool kit allows several benefits over conventional simulation packages. There are several main advantages of this new package over the existing ones. Primarily it provides a means to interactively vary parameters. Based in Java, it offers a cross-platform and web environment solution. It is also capable of presenting new control designs and algorithms.
List of Slides

7  Project focus
8  Motivation of research
9  Existing software tools
14 Proposed approach
16 Software requirements and specifications
17 Thesis organization
18 Focus on difference equations
19 Fundamental building blocks
25 GUI development
27 Developer’s viewpoint
31 User’s viewpoint
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Simple example problem for comparisons</td>
</tr>
<tr>
<td>38</td>
<td>Conclusion</td>
</tr>
<tr>
<td>39</td>
<td>Specific conversions from continuous form to discrete</td>
</tr>
<tr>
<td>45</td>
<td>General $s$ to $z$ transform matrices</td>
</tr>
<tr>
<td>49</td>
<td>General $z$ transfer function to difference equations</td>
</tr>
<tr>
<td>50</td>
<td>Parameterization</td>
</tr>
<tr>
<td>51</td>
<td>Approximate PID</td>
</tr>
<tr>
<td>54</td>
<td>Non-linear PID</td>
</tr>
<tr>
<td>57</td>
<td>Active Disturbance Rejection Control</td>
</tr>
<tr>
<td>60</td>
<td>Discrete Time Optimal Control</td>
</tr>
<tr>
<td>64</td>
<td>Filters</td>
</tr>
<tr>
<td>75</td>
<td>Classic system</td>
</tr>
<tr>
<td>76</td>
<td>Combined General System</td>
</tr>
</tbody>
</table>
Development of a Graphical Control Design and Tuning Platform

77  General Blocks
81  Introduction: A move towards active optimization
82  Threshold limit requirement
83  Optimization and threshold assumptions
84  Active noise quantification methods
93  Summary
94  Future research
95  Project link
96  Simtk API Documentation
97  Software Packages
99  Constructed tools
95  Obtain the Java Runtime Environment
96  Interface Structure
98  Example usage steps and tips
99  Troubleshooting
100 Inheritance structure
108 Containment structure
111 Screen shots
Project focus
Motivation of research

* Control theory history
Development of a Graphical Control Design and Tuning Platform

Existing software tools

* Matlab
  * Matlab is powerful

Figure 1: Matlab

* It is a controls standard
  * Simulink allows easy set up nonlinear systems
  * Shortcomings of Matlab
* Simulink is great but tuning is tedious
* Matlab has GUI but it takes a bit of programming to use
* Matlab license issues
* Version problems
* SysQuake

![SysQuake LE](image)

Figure 2: SysQuake

* Free package with syntax similar to Matlab
* Built around real time dragging
Figure 3: SysQuake icon

* Led to the first project with simplicity of NPID tuning
Development of a Graphical Control Design and Tuning Platform

Figure 4: SysQuake window

Aaron Radke
Development of a Graphical Control Design and Tuning Platform

* Shortcomings of SysQuake
  * Changing the system needed differential equation derivations
Proposed approach

* Problem formulation
  * Start new with difference equations
  * Real-time functionality
* Software development strategy
  * Development language

Figure 5: Java logo

* Free
* Accessible
* Powerful framework

* Structure
Software requirements and specifications

* A real time, interactive, dynamic tuning environment for control systems.
* Extensible framework
* Simple construction of nonlinear systems
* Cross-platform, web application deployment
* Capability of presenting new control designs and algorithms
* Useful and practical for implementation
Thesis organization
Focus on difference equations

* Start with the core difference equation solver method

* The GUI can be built later
Fundamental building blocks

* Object orientedness
  * Library of blocks
* SimBlocks
  * SimSource
  * SimFunction
  * SimSink
* Class inheritance structure diagrams
Figure 6: The class structure for the blocks (without implementation classes)
Figure 7: The entire hierarchical class structure of the project
* Class inheritance structure

\[ \text{SimPid} \rightarrow \text{SimStf} \rightarrow \text{SimZtf} \]
\[ \rightarrow \text{SimDiffEq} \rightarrow \text{SimFunction} \]
\[ \rightarrow \text{SimBlock} \rightarrow \text{SimBase} \]  

Table 1: Example hierarchy path of a PID controller

<table>
<thead>
<tr>
<th>Class name</th>
<th>provides</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimBase</td>
<td>simple title and description structures</td>
</tr>
<tr>
<td>SimBlock</td>
<td>the ability to interface with the rest of the simulation library</td>
</tr>
<tr>
<td>SimFunction</td>
<td>the functionality of inputs and outputs</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>SimDiffEq</td>
<td>the ability to create, display, and edit discrete difference equations</td>
</tr>
<tr>
<td>SimZtf</td>
<td>the ability to create, display, and edit z transfer functions and convert them to discrete difference equations</td>
</tr>
<tr>
<td>SimStf</td>
<td>the ability to create, display, and edit s transfer functions and convert them to z transfer functions</td>
</tr>
<tr>
<td>SimPid</td>
<td>the specific transfer function for proportional, integral and derivative control</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

* Block interconnection
* Small memory size
* Block interconnection method
GUI development
Figure 8: Screen shot for the sample simtk applet GUI
Developer’s viewpoint

* Top-level managing class
  * Hierarchy
  * SimFactory
  * Containment structure of example applets
Figure 9: Simplified example containment of classes for the GeneralControlLoop applet.
Figure 10: Example Containment of classes for the GeneralControl-Loop applet

6/3/03  Aaron Radke
* Setting up a simulation
  * Loading or instantiating blocks
  * Adding and connecting blocks
  * Simulating

* Building blocks
  * Extend a block
  * Add adjustable variables
  * Override iteration definition

* Developer’s documentation
User’s viewpoint

* Nature of the system

* Simulation accuracy

* Globals
Simple example problem for comparisons

* Applet Source

\[ y'(t) = f(t, y(t)) \]  \hspace{1cm} (2)
\[ y(t_0) = y_0 \]  \hspace{1cm} (3)
Figure 11: Comparison of ode solvers with 5 iterations
Figure 12: Comparison of ode solvers with 10 iterations
Figure 13: Comparison of ode solvers with 20 iterations.
Figure 14: Comparison of ode solvers with 200 iterations
Figure 15: Comparison of ode solvers with 3 iterations and Runge-Kutta
Conclusion

* Ten-fold increase of Tustin over Simple Euler
Specific conversions from continuous form to discrete

* Simple Euler

\[ s = \frac{-1 + z}{z} \]  

(4)

* Integrator

\[ \frac{1}{s} \rightarrow \frac{z}{-1 + z} \]  

(5)

* Double Integrator

\[ \frac{1}{s^2} \rightarrow \frac{z^2}{1 - 2z + z^2} \]  

(6)

* The problem with the Simple Euler method
* Tustin

\[ s = \frac{2 (-1 + z)}{T (1 + z)} \]  (7)

* Integrator

\[ \frac{1}{s} \rightarrow \frac{T + Tz}{-2 + 2z} \]  (8)

* Double Integrator

\[ \frac{1}{s^2} \rightarrow \frac{T^2 + 2T^2 z + T^2 z^2}{4 - 8z + 4z^2} \]  (9)

* First order plant

\[ \frac{1}{s + a} \rightarrow \frac{T + Tz}{-2 + aT + 2z + aTz} \]  (10)

* Second order plant
\[
\frac{\omega^2}{s^2 + 2\zeta\omega + \omega^2} \rightarrow \frac{T^2 \omega^2 + 2T^2 z \omega^2 + T^2 z^2 \omega^2}{4 - 4T \zeta \omega + T^2 \omega^2 + 2z (-4 + T^2 \omega^2) + z^2 (4 + 4T \zeta \omega + T^2 \omega^2)}
\]

* Second order plant with \(\omega = \zeta = 1\)

\[
\frac{1}{s^2 + 2 + \omega^2} \rightarrow \frac{T^2 + 2T^2 z + T^2 z^2}{4 - 4T + T^2 - 8z + 2T^2 z + 4z^2 + 4T z^2 + T^2 z^2}
\]

* PID controller continuous to discrete

\[
\frac{k_p s + k_d s^2 + k_i}{s} \rightarrow \frac{4k_d - 2k_p T + k_i T^2 - 8k_d z + 2k_i T^2 z + 4k_d z^2 + 2k_p T z^2 + k_i T^2 z^2}{-2T + 2T z^2}
\]
* Approximate differentiator

\[
\frac{s}{(\tau s + 1)^2} \rightarrow \frac{-2T + 2T \tau^2}{T^2 + 2T^2 z + T^2 z^2 - 4T \tau + 4T \tau^2 \tau + 4\tau^2 - 8z\tau^2 + 4z^2\tau^2}
\]  

(14)

* General transfer function derivations

\[G_p = n_0\]  

(15)

\[n_0.\]  

(16)

\[G_p = n_1 s + n_0\]  

(17)
\[-2n_1 + n_0 T + 2n_1 z + n_0 T z. \tag{18}\]

\[Gp = n_2 s^2 + n_1 s + n_0 \tag{19}\]

\[4n_2 - 2n_1 T + n_0 T^2 - 8n_2 z + 2n_0 T^2 z + 4n_2 z^2 + 2n_1 T z^2 + n_0 T^2 z^2. \tag{20}\]

\[Gp = n_3 s^3 + n_2 s^2 + n_1 s + n_0 \tag{21}\]
\[-8 n_3 + 4 n_2 T - 2 n_1 T^2 + n_0 T^3\]
\[+ 24 n_3 z - 4 n_2 T z - 2 n_1 T^2 z + 3 n_0 T^3 z\]
\[-24 n_3 z^2 - 4 n_2 T z^2 + 2 n_1 T^2 z^2 + 3 n_0 T^3 z^2 + 8 n_3 z^3\]
\[+ 4 n_2 T z^3 + 2 n_1 T^2 z^3 + n_0 T^3 z^3.\] (22)

\[Gp = n_4 s^4 + n_3 s^3 + n_2 s^2 + n_1 s + n_0\] (23)

\[16 n_4 - 8 n_3 T + 4 n_2 T^2 - 2 n_1 T^3 + n_0 T^4 - 64 n_4 z + 16 n_3 T z\]
\[-4 n_1 T^3 z + 4 n_0 T^4 z + 96 n_4 z^2 - 8 n_2 T^2 z^2 + 6 n_0 T^4 z^2 - 64 n_4 z^3\]
\[-16 n_3 T z^3 + 4 n_1 T^3 z^3 + 4 n_0 T^4 z^3 + 16 n_4 z^4\]
\[+ 8 n_3 T z^4 + 4 n_2 T^2 z^4 + 2 n_1 T^3 z^4 + n_0 T^4 z^4.\] (24)
General $s$ to $z$ transform matrices

\[ Z_n = M_n \cdot S_n \cdot T_n \]  \hspace{1cm} (25)

\[ S_n = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \]  \hspace{1cm} (26)
\[
T_n = \begin{pmatrix}
T^0 \\
T^1 \\
T^2 \\
\vdots \\
T^n
\end{pmatrix}
\]  \hspace{1cm} (27)

* Conversion matrixes

\[
M_0 = \begin{pmatrix}
1
\end{pmatrix}
\]  \hspace{1cm} (28)

\[
M_1 = \begin{pmatrix}
-2 & 1 \\
2 & 1
\end{pmatrix}
\]  \hspace{1cm} (29)
Development of a Graphical Control Design and Tuning Platform

\[
M_2 = \begin{pmatrix}
-4 & -2 & 1 \\
-8 & 0 & 2 \\
4 & 2 & 1
\end{pmatrix}
\]

\[(30)\]

\[
M_3 = \begin{pmatrix}
-8 & 4 & -2 & 1 \\
24 & -4 & -2 & 3 \\
-24 & -4 & 2 & 3 \\
8 & 4 & 2 & 1
\end{pmatrix}
\]

\[(31)\]

\[
M_4 = \begin{pmatrix}
-16 & -8 & 4 & -2 & 1 \\
-64 & 16 & 0 & -4 & 4 \\
-96 & 0 & -8 & 0 & 6 \\
-64 & -16 & 0 & 4 & 4 \\
16 & 8 & 4 & 2 & 1
\end{pmatrix}
\]

\[(32)\]
Listing 1: S transform to z transform algorithm

```c
for (int i = 0; i<=order;i++) {
    // loop through each z^i terms
    // loop through each coefficient (which is also T^j terms)
    for (int j = 0; j<=order;j++){
        // continuously add up terms for the coef of z^i
        // storing them into zpoly in the form of z^3+z^2+z^1
        // from MSZ to LSZ
        zpoly[order − i] += c2dmatrix[i][j]*Tz[j];
    }
}
```
General $z$ transfer function to difference equations

\[
\frac{y}{r} = \frac{n_0 z^4 + n_1 z^3 + n_2 z^2 + n_3 z^1 + n_4 z^0}{d_0 z^4 + d_1 z^3 + d_2 z^2 + d_3 z^1 + d_4 z^0} \tag{33}
\]

\[
\begin{align*}
\frac{z^{-4}}{z^{-4}} & \\
\end{align*} \tag{34}
\]

\[
y = \frac{1}{d_0} \left( (n_0 r + n_1 r z^{-1} + n_2 r z^{-2} + n_3 z^{-3} + n_4 z^{-4}) \right.
\]

\[
- \left( d_1 r z^{-1} + d_2 r z^{-2} + d_3 z^{-3} + d_4 z^{-4} \right) \bigg) \tag{35}
\]

\[
y = \frac{\sum_{i=1}^{q} (n_i r z^{-i} - d_i r z^{-i}) + n_0 r}{d_0} \tag{36}
\]

\[
z^{-i} \rightarrow y(k - i) \tag{37}
\]
Listing 2: $z$ transform to difference equation algorithm

```c
double tempsumterms = 0;
    for (int i = 1; i <= zorder; i++){
        tempsumterms += num[i] * in.val[i] - den[i] * out.val[i];
    }
    out.val[0] = (tempsumterms + num[0] * in.val[0]) / den[0];
```
Parameterization

\[ k_d = 2\omega_c \]  \hfill (38)

\[ k_d = \omega_c^2 \]  \hfill (39)
Approximate PID

\[ u = k_p e + k_i \int e + k_d \dot{e} \]  \hspace{1cm} (40)

\[ \dot{u} = k_p \dot{e} + k_i e + k_d \ddot{e} \] \hspace{1cm} (41)

\[ \frac{u}{e} = k_p \frac{s}{s} + k_d \dot{s} \] \hspace{1cm} (42)

\[ \dot{s} = \frac{s}{(\tau s + 1)^2} \] \hspace{1cm} (43)
Figure 16: 2nd order approximate derivative with $\tau=1,0.5,0.25$ on a step function

$$u = k_p + k_i \frac{s}{s} + k_d \frac{s}{(\tau s + 1)^2}$$  \hspace{1cm} (44)
\[ u = \frac{s^3(k_p \tau^2) + s^2(k_d + 2k_p \tau + k_i \tau^2) + s(k_p + 2k_i \tau) + k_i}{s^3(\tau^2) + s^2(2\tau) + s(1) + s(0)} \]  \hspace{1cm} (45)
Nonlinear PID

* Extension of classical PID

\[
u = G_{kp}(e)e + G_{ki}(e) \int e + G_{kd}(e)\dot{e}
\]  

(46)

* Gfunc

Figure 17: Nonlinear GFunction
\[ G_x(e, k_{p1}, k_{p2}, k_{n1}, k_{n2}, w_p, w_n) = \begin{cases} 
  e > 0 & \begin{cases} 
    e \cdot k_{p1} & |e| < w_p \\
    e \cdot k_{p2} & |e| \geq w_p 
  \end{cases} \\
  e \leq 0 & \begin{cases} 
    e \cdot k_{n1} & |e| < n_p \\
    e \cdot k_{n2} & |e| \geq n_p 
  \end{cases} 
\] 

(47)

Listing 3: GFunction example code in java (extracted from SimGFnc.java)

```java
// positive
if (error > 0) {
    // is in the first gain width
    if (error < pw.getValue()) {
        output = error * pk1;
    }
    else {
        output = error * pk2;
    }
}
```
}  
//negative  
else{
  //small region
  if(Math.abs(error) < nw){
    output = error*nk2;
  }
  //large region
  else{
    output = error*nk2;
  }
}
Active Disturbance Rejection Control

\[ \begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= x_3 + b_0 u \\
\dot{x}_3 &= h(t)
\end{align*} \] (48)

\[ \begin{align*}
\dot{z}_1 &= z_2 + B_1(y - z_1) \\
\dot{z}_2 &= z_3 + B_2(y - z_1) + b_0 u \\
\dot{z}_3 &= B_3(y - z_1)
\end{align*} \] (49)

\[ u = \frac{u_0 - z_3}{b_0} \] (50)
Figure 18: ADRC block diagram controller depicting the separation into two transfer functions

\[
\frac{U}{R} = \frac{\omega_c^2 \begin{bmatrix} 1 & 3\omega_o & 3\omega_o^2 & \omega_o^3 \end{bmatrix} \cdot S}{b_0 \begin{bmatrix} 1 & (3\omega_o + 2\omega_c) & (3\omega_o 2\omega_c + \omega_c^2 + 3\omega_o^2) & (0) \end{bmatrix} \cdot S}
\] (51)
\[
\frac{Y}{R} = \frac{\omega_c^2 \left[ (3\omega_o\omega_c^2 + 3\omega_o^22\omega_c + \omega_o^3) \right]}{b_0 \left[ (1) (3\omega_o + 2\omega_c) \right]} \cdot S
\]

\[
S = \begin{pmatrix}
s^3 \\
\vdots \\
1 \\
0
\end{pmatrix}
\]

\[
b_0 = \frac{\text{Largest plant numerator coefficient}}{\text{Highest order plant denominator coefficient}}
\]
Discrete Time Optimal Control

\[ u = -r \text{sign} \left( x_1 + \frac{x_2 |x_2|}{2r} \right) \] (55)
\[ u = \text{fst}(x_1, x_2, r, h) \]  
\[ d = rh \]  
\[ d_0 = hd \]  
\[ y = x_1 + hx_2 \]  
\[ a_0 = \sqrt{d^2 + 8r|y|} \]  
\[ a = \begin{cases} 
  x_2 + \frac{a_0 - d}{2} \text{sign}(y), & |y| > d_0 \\
  x_2 + \frac{y}{h}, & |y| \leq d_0 
\end{cases} \]  
\[ \text{fst} = - \begin{cases} 
  r\text{sign}(a), & |a| > d \\
  r\frac{a}{d}, & |a| \leq d 
\end{cases} \]  

Listing 4: DTOC control algorithm implemented in java

```java
/**DTOC, fst function.
 *from dr. Gao, 3/7/3
```
Development of a Graphical Control Design and Tuning Platform

```c
/* this is the magic stuff!
*/
double fst2(double x1, double x2, double u0, double r, double h)
{
    double d, d0, y, a0, a1;

    d=r*h;
    d0=d*h;
    y=x1-u0+h*x2;
    a0=Math.sqrt(d*d+8*r*Math.abs(y));
    if (y>d0) a1=x2+(a0-d)/2;
    else if (y<-d0) a1=x2-(a0-d)/2;
    else a1=x2+y/h;

    return(-r*usat(a1,d));
}

/** nonlinear unity saturation function.
* from dr. Gao, 3/7/3
*/
double usat(double x, double delta)
{
    if (x>delta) return(1.0);
```

6/3/03 Aaron Radke
else if (x<−delta) return (−1.0);
else return (x/delta);
}
Filters

* Pre-filters and Profiles

* The Trapezoidal Pre-filter

\[
y = \begin{cases} 
0 & x < d \\
\frac{3g}{w} x - \frac{3gd}{w} & d < x < d + \frac{w}{3} < x < d \\
g & d + \frac{w}{3} < x < d + \frac{2w}{3} \\
\frac{-3g}{w} x + \frac{3gd}{w} + 3g & d + \frac{2w}{3} < x < d + w \\
0 & x > d + w
\end{cases}
\]  

(57)
Figure 19: Trapezoidal source for profiles
Figure 20: Profile generated from the integration of a trapezoidal input

* The S-curve Pre-filter
Figure 21: Triangular source for S-curve profiles

\[ y = m_s mx + (-mm_s x_{int}) \]  \hspace{1cm} (58)
\[ m = \frac{6g}{w} \quad (59) \]

\[ m_s = 0; \quad x_{int} = 0 \quad x < d \]
\[ m_s = 1; \quad x_{int} = d \quad d < x < d + \frac{w}{6} \]
\[ m_s = -1; \quad x_{int} = d + \frac{w}{3} \quad d + \frac{w}{6} < x < d + \frac{w}{3} \quad (60) \]
\[ m_s = 0; \quad x_{int} = 0 \quad d + \frac{w}{3} < x < d + \frac{2w}{3} \]
\[ m_s = -1; \quad x_{int} = d + \frac{2w}{3} \quad d + \frac{2w}{3} < x < d + \frac{5w}{6} \]
\[ m_s = 1; \quad x_{int} = d + w \quad d + \frac{5w}{6} < x < d + w \]
\[ m_s = 0; \quad x_{int} = 0 \quad x > d + w \]

* Critically damped functions

\[ R(s) = \frac{\omega_r^2}{s^2 + 2\omega_r s + \omega_r^2} \quad (61) \]
Figure 22: Profile generated from a step and 2nd order input

\[ R(s) = \frac{\omega_r^2}{s^2 + 2\omega_r s + \omega_r^2} \frac{\omega_r^2}{s^2 + 2\omega_r s + \omega_r^2} \]  \hspace{1cm} (62)
* Polynomial pre-filter

\[
y(x) = g \begin{cases} 
\frac{10x^3}{T_r^3} - \frac{15x^4}{T_r^4} + \frac{6x^5}{T_r^5} & x < T_r \\
1 & x \geq T_r 
\end{cases}
\]  

(63)

Figure 23: Profile generated from the simple polynomial with \( T_r = 5 \)
* General Normalized Butterworth filters

\[ H(s) = \frac{1}{s + 1} \]  \hspace{1cm} (64)

\[ H(s) = \frac{1}{s^2 + \sqrt{2}s + 1} \]  \hspace{1cm} (65)

\[ H(s) = \frac{1}{s^3 + 2s^2 + 2s + 1} \]  \hspace{1cm} (66)

\[ H(s) = \frac{1}{\left(s^2 + \frac{3}{20}\sqrt{26}s + 1\right) \left(s^2 + \frac{181}{500}\sqrt{26}s + 1\right)} \]  \hspace{1cm} (67)

* Low-pass, High-pass, Bandpass, and Bandstop Extension

\[ \omega = 2\pi f \]  \hspace{1cm} (68)
Development of a Graphical Control Design and Tuning Platform

\[ s \rightarrow \frac{s}{\omega} \quad (69) \]

\[ s \rightarrow \frac{\omega}{s} \quad (70) \]

\[ s \rightarrow \frac{s^2 + \omega^2}{Bs} \quad (71) \]

\[ s \rightarrow \frac{Bs}{s^2 + \omega^2} \quad (72) \]

* Lowpass

\[ H(s) = \frac{1}{\frac{1}{\omega}s + 1} \quad (73) \]
Figure 24: 1st order low-pass Butterworth filter filtering noise

\[ H(s) = \frac{1}{\frac{1}{\omega^2} s^2 + \frac{\sqrt{2}}{\omega} s + 1} \]  \hspace{1cm} (74)

\[ H(s) = \frac{1}{\frac{1}{\omega^3} s^3 + \frac{2}{\omega^2} s^2 + \frac{2}{\omega} s + 1} \]  \hspace{1cm} (75)
* Highpass

\[ H(s) = \frac{1}{\frac{s}{\omega} + 1} \]  \hspace{1cm} (76)

\[ H(s) = \frac{s}{s + \omega} \]  \hspace{1cm} (77)

\[ H(s) = \frac{1}{\frac{1}{s^2} \omega^2 + \frac{\sqrt{2}}{s} \omega + 1} \]  \hspace{1cm} (78)

\[ H(s) = \frac{s^2}{s^2 + \sqrt{2} \omega s + \omega^2} \]  \hspace{1cm} (79)

\[ H(s) = \frac{1}{\frac{1}{s^3} \omega^3 + \frac{2}{s^2} \omega^2 + \frac{2}{s} \omega + 1} \]  \hspace{1cm} (80)

\[ H(s) = \frac{s^3}{s^3 + 2\omega s^2 + 2\omega^2 s + \omega^3} \]  \hspace{1cm} (81)
Development of a Graphical Control Design and Tuning Platform

Classic system

Figure 25: Simple classic controller without noise or disturbance

Figure 26: Classic controller model setup with noise and disturbance
Combined General System

Figure 27: New General controller model setup
General Blocks

general controller
2 inputs one output

Figure 28: New General controller block
Figure 29: Improved General controller to include pre-filter and observer before summing loop
Figure 30: New General controller block example using PID
Figure 31: ADRC Controller (which is a special case of the improved general controller where error controller is unity)
Introduction: A move towards active optimization

* Actively via parameterization
Threshold limit requirement

* Solution in terms of engineering requirements
  * Hardware implementation
* Noise level limit
  * Quantifying noise
* Rate of change limit
  * Quantifying rate of change
* Overshoot limit
  * Quantifying overshoot
* The Most Significant Bandwidth Threshold
Optimization and threshold assumptions

* Highest Bandwidth possible is best

* The threshold is easily defined
Active noise quantification methods

* Introduction

Figure 32: The human eye can simply ’get a feel’ for the level of noise
* Max and min

\[ \nu_{\text{maxmin}} = y_{\text{max}} - y_{\text{min}} \] (82)

Figure 33: Maximum value, with a buffer size of 15
Figure 34: Minimum value, with a buffer size of 15

* Moving average window

\[ \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \]  \hspace{1cm} (83)
Figure 35: Averaging filter with a buffer size of 20

* Variance and standard deviation
\[ s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2 \]  
(84)

\[ = \sum y^2 - n\bar{y}^2 \]  
(85)

Figure 36: Variance with a buffer size of 15 which hovers around 4
\[ s = \sqrt{s^2} \]

(86)

\[
\text{COV} = \frac{s}{\bar{y}} \]

(87)

Figure 37: Standard Deviation with a buffer size of 15
* Linear Regression

\[ y_i = b_0 + b_1 x_i \]  \hspace{1cm} (88)

\[ b_1 = \frac{(\sum xy) - n\bar{x}\bar{y}}{(\sum x^2) - n\bar{x}^2} \]  \hspace{1cm} (89)

\[ b_0 = \bar{y} - b_1 \bar{x} \]  \hspace{1cm} (90)

* Correlation coefficient to quantify the noise

\[ R^2 = \frac{SST - SSE}{SST} \]  \hspace{1cm} (91)

\[ R = \sqrt{R^2} \]  \hspace{1cm} (92)
$SST = \left( \sum y^2 \right) - n\bar{y}^2$  \hspace{1cm} (93)

$SSE = \left( \sum y^2 \right) - \left( b_0 \sum y \right) - \left( b_1 \sum xy \right)$  \hspace{1cm} (94)

* Relationship between bandwidth and linear model

\[ \alpha n \sim \beta b_1 \sim \omega_c \]  \hspace{1cm} (95)

* High-Pass Filter with Variance
* Wavelet for a new Variance definition

\[ s' = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y}')^2 \]  \hspace{1cm} (96)

\[ = \sum y^2 - n\bar{y}'^2 \]  \hspace{1cm} (97)
COWV = \frac{s'}{\bar{y}'} \quad (98)
Summary

* A real time, interactive, dynamic tuning environment for control systems.
  * Extensible framework
  * Simple construction of nonlinear systems
  * Cross-platform, web application deployment,
  * Useful and practical for implementation
* Simulation building blocks
* Simple GUI examples
  * Teaching tool
  * Capability of presenting new control designs and algorithms
Future research

* Additional library blocks

* Graphical setup of SimBlocks

* Extension to a hardware tuning and monitoring interface
  * Use the library to find new tuning parameters
  * Network protocols

* Extension to Automated Computer Assisted Design for Controls
References


[6] Zhiqiang Gao, Yi Huang and Jingqing Han, “An Alternative Paradigm for Control System Design,” *Proc. of the 40th IEEE Con-
ference on Decision and Control, Orlando, Florida USA, December 2001, pp. 4578-4585.


Development of a Graphical Control Design and Tuning Platform

IEEE conference on Decision and Control, Orlando, Florida, USA, December 2001, pp. ???.


Simtk API Documentation

* Javadoc
  * Simtk javadoc in HTML
  * Simtk javadoc in PDF
* Supporting documentation
  * PtPlot
  * Sun Microsystem’s Java API documentation
Software Packages

* Java
* Mathematica
* SysQuake
* Matlab
* Ptolemy
* Java math exploration
* Octave
* Perl
* Gnuplot
* Simple ode solvers
* Hartmath
* Vim
Development of a Graphical Control Design and Tuning Platform

* Graphviz

* \LaTeX
Constructed tools

* Aaindex
* Aao2tex
* Aao2pdf
* Aao.vim
* Java2dot
* Containment
* Aao2dot
* ExtractClassInfo
* Compressgif
* Csuthesis.sty
Table 2: Class list and description for the ’simtk’ package

<table>
<thead>
<tr>
<th>Class name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContainsFactory</td>
<td>Interface to simply make sure that the proper functions are implemented inorder to contain a factory.</td>
</tr>
<tr>
<td>HasInput</td>
<td>Interface for any Block that has an input; this way it is easy to test if a block has an input with instanceof.</td>
</tr>
<tr>
<td>HasOutput</td>
<td>Interface for any Block that has an output; this way it is easy to test if a block has an output with instanceof.</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SimAdjustable</td>
<td>Adjustable abstract type, (a new method based around the individual variables is now used for adjusting the blocks for getPanel()).</td>
</tr>
<tr>
<td>SimPanelable</td>
<td>If a class extends this class it has the ability to make an adjustment panel.</td>
</tr>
<tr>
<td>SimReLoadable</td>
<td>Interface ensures that a method has a calculateAndPlot() method.</td>
</tr>
<tr>
<td>Simulatable</td>
<td>Interface ensures that a method has a doSimulation() method and the getSimParams() method.</td>
</tr>
<tr>
<td><strong>ScrollViewblePicture</strong></td>
<td>Displays pictures in a scroll-able form, implemented from sun’s Java example on ScrollPanes, (not used in the simtk library).</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Sim1stOrder</strong></td>
<td>First order function with a single variable ’a’(the pole location).</td>
</tr>
<tr>
<td><strong>Sim2ndOrder</strong></td>
<td>Second order function with 2 variables ’w’(omega) and ’z’(zeta).</td>
</tr>
<tr>
<td><strong>SimAdd</strong></td>
<td>Addition block which adds multiple inputs together which results in a single output.</td>
</tr>
<tr>
<td><strong>SimADRC</strong></td>
<td>Compilation of SimBlocks to create the ADRC(Active Disturbance Rejection) block</td>
</tr>
</tbody>
</table>
### SimBase
- The main abstract block for many of the classes, providing reload mechanisms, as well as titles and description methods.

### SimBlock
- Main abstract block for all types of SimBlocks, sets defaults and sets up a fresh variable container and GUI panel.

### SimBuffer
- Buffer block, the glue or wires that connects the SimBlocks together.

### SimContinuousProfile
- Compilation of SimStep and Sim2ndOrder to create a profile.

### SimDialogManager
- An old class which at one point was used to create JDialogs for the tuning of SimBlocks, but was replaced with the OO variables.
<table>
<thead>
<tr>
<th><strong>SimDiff</strong></th>
<th>Second order approximate differentiator defined by a Stf and is a double pole.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SimDiffEq</strong></td>
<td>Difference equation which is provided for an intermediate class as well as display of the difference equation for implementation purposes.</td>
</tr>
<tr>
<td><strong>SimDisturbance</strong></td>
<td>Disturbance sources which include frequency and gain variable sine and square waves.</td>
</tr>
<tr>
<td><strong>SimDoubleIntegrator</strong></td>
<td>Directly calculated Tustin double integrator.</td>
</tr>
<tr>
<td><strong>SimDoubleTriangle</strong></td>
<td>Double triangle profile function, which after integration produces an S curve profile.</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SimDTOC</td>
<td>The DTOC (Discrete Time Optimal Control) Controller; uses two inputs, input1: error, input2: error differentiated.</td>
</tr>
<tr>
<td>SimDTOCDiff</td>
<td>The DTOC algorithm with an internal differentiator so a single input of error is required.</td>
</tr>
<tr>
<td>SimFactory</td>
<td>Factory class which makes and controls the simulation environment both for the entire system and hierarchal hybrid blocks.</td>
</tr>
<tr>
<td>SimFilter</td>
<td>Filter with a selection of multiple types of Butterworth filters.</td>
</tr>
<tr>
<td>SimFunction</td>
<td>Defines the bare bones of a single output Function block with multiple inputs.</td>
</tr>
<tr>
<td>SimFunction2Input</td>
<td>Defines the general class for 2 inputs and 1 output (this was replaced with the more general SimFunction).</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SimFunctionFactory</td>
<td>Compilation internalizes the SimFactory into a SimFunction for complex hybrid function blocks such as NPID composed from simple building blocks.</td>
</tr>
<tr>
<td>SimFunctionSelector</td>
<td>Provides dynamic selection of blocks by the user at runtime; it provides a list of simulation blocks which replaces the current block.</td>
</tr>
<tr>
<td><strong>SimFunctionTest</strong></td>
<td>A test block to test internal SimFactory block’s.</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td><strong>SimFunctionXInput</strong></td>
<td>Multiple input SimFunction (has been replaced with the more general SimFunction which can now have multiple inputs).</td>
</tr>
<tr>
<td><strong>SimGain</strong></td>
<td>Simple function applies a gain to the input.</td>
</tr>
<tr>
<td><strong>SimGeneralClasicController</strong></td>
<td>A hybrid block in the general controller form that allows a simple controller with the classic ”r-y = e” internally calculated to appear to a purely error based controller.</td>
</tr>
<tr>
<td>SimGFunc</td>
<td>A non-linear gain function for use in shaping PID gains for NPID.</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>SimImage</td>
<td>Allows a picture such as a block diagram to be loaded on a panel.</td>
</tr>
<tr>
<td>SimImageFactory</td>
<td>Common utilites for loading pictures from files.</td>
</tr>
<tr>
<td>SimInput</td>
<td>Defines the bare bones of a multi-input block for multi-input functions and sinks.</td>
</tr>
<tr>
<td>SimIntegrator</td>
<td>A directly derived Tustin integrator.</td>
</tr>
<tr>
<td>SimMultiply</td>
<td>Multiplication of two inputs with the result at the output.</td>
</tr>
<tr>
<td>SimNoise</td>
<td>Source which after each iteration outputs a random value from -1 to 1.</td>
</tr>
<tr>
<td><strong>SimNothing</strong></td>
<td>Outputs zeros, this can be used for test cases of zero noise instead of having some noise.</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SimNPID</strong></td>
<td>Compilation of SimBlocks with a couple SimGFunc to create the NPID(Non-linear Proportional Integral Derivative) Controller.</td>
</tr>
<tr>
<td><strong>SimNPIDwc</strong></td>
<td>Extends the SimNPID into a parameterized PID in terms of omega for a double integrator plant.</td>
</tr>
<tr>
<td><strong>SimParameters</strong></td>
<td>Stores and keeps all the important global type simulation parameters within this duplicated class which each block receives.</td>
</tr>
<tr>
<td><strong>SimPid</strong></td>
<td>Aproximate differentiator PID, this is a hybrid block composed of simpler blocks.</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SimPolyProfile</strong></td>
<td>Provides a smoother response for position, velocity, and acceleration and Jerk in place of a step function or trapezoidal function.</td>
</tr>
<tr>
<td><strong>SimProfile</strong></td>
<td>A dynamically create-able profile which has a selection of types of inputs(step, trapezoidal, triangle) and a selection of types of filters(integrators, 2nd order, butterworth).</td>
</tr>
</tbody>
</table>
### Development of a Graphical Control Design and Tuning Platform

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimPtPlot</td>
<td>The main plotting engine depends on the Ptolemy PtPlot package; this adds interfaces for plot exporting and connections.</td>
</tr>
<tr>
<td>SimSaturation</td>
<td>Simple saturation function, contains and upper and lower bound limit.</td>
</tr>
<tr>
<td>SimSink</td>
<td>Defines the bare bone necessities to a simBlock which only has inputs.</td>
</tr>
<tr>
<td>SimSource</td>
<td>A Bare-bones definition of a source which only has an output.</td>
</tr>
<tr>
<td>SimSourceFactory</td>
<td>Compilation internalizes the SimFactory into a SimSource for complex hybrid source blocks such as some profiles.</td>
</tr>
<tr>
<td>SimSourceSelector</td>
<td>Provides dynamic selection of blocks by the user at runtime; it provides a list of simulation sources which replaces the current block.</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SimStat</td>
<td>Provides statistical functions, such as average and standard deviation, on incoming data over a specified range of samples.</td>
</tr>
<tr>
<td>SimStdout</td>
<td>A sink that simply sends the current output to the standard output.</td>
</tr>
<tr>
<td>SimStep</td>
<td>A simple source Step function which reaches a specified gain value as fast as possible.</td>
</tr>
<tr>
<td>Block Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SimStf</td>
<td>Powerful block that simulates S domain transfer functions and converts them to discrete transfer functions!</td>
</tr>
<tr>
<td>SimSubtract</td>
<td>Subtracts the second input from the first input; ”out = in1 - in2”</td>
</tr>
<tr>
<td>SimTest</td>
<td>A test of extending the SimFunctionFactory.</td>
</tr>
<tr>
<td>SimTextBoxOutput</td>
<td>A sink block that simply send the current output to a formatted output in a swing text box.</td>
</tr>
<tr>
<td>SimTimeIndex</td>
<td>Outputs the current time index which is useful for testing blocks over a range of inputs.</td>
</tr>
<tr>
<td><strong>SimTrapezoid</strong></td>
<td>Trapezoidal profile which can be integrated to provide a smoother response for position, velocity, and acceleration in place of a step function.</td>
</tr>
<tr>
<td><strong>SimTrapezoidalProfile</strong></td>
<td>Trapezoidal profile combined with an integrator to form a smooth profile.</td>
</tr>
<tr>
<td><strong>SimZtf</strong></td>
<td>Z domain transfer function that automatically creates difference equations.</td>
</tr>
<tr>
<td><strong>SimZtfDirect</strong></td>
<td>A Z domain transfer function that calculates the output directly without handing the output down to the diffEq class.</td>
</tr>
</tbody>
</table>
Table 3: Class list and description for the ’simtk.applets’ package

<table>
<thead>
<tr>
<th>Class name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADRC</td>
<td>GUI ADRC example applet of the simtk library.</td>
</tr>
<tr>
<td>Console</td>
<td>Console example of the simtk library to allow plot exports in non-applet mode</td>
</tr>
<tr>
<td>GCADRC</td>
<td>GUI ADRC example applet using the General Controller model.</td>
</tr>
<tr>
<td>GeneralControlLoop</td>
<td>GUI example applet using the General Controller model for selectable controllers.</td>
</tr>
<tr>
<td>SimpleLoop</td>
<td>GUI example applet illustrating a super simple control loop.</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>SimpleLoopNN</td>
<td>GUI example applet illustrating a super simple control loop with no noise.</td>
</tr>
<tr>
<td>TestBlock</td>
<td>GUI example applet with no loop, simply meant to test block functions with an input and output.</td>
</tr>
</tbody>
</table>
Table 4: Class list and description for the ’simtk.variable’ package

<table>
<thead>
<tr>
<th>Class name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimCheckBox</td>
<td>A check box boolean variable with a title, which refreshes when the value changes.</td>
</tr>
<tr>
<td>SimComboBox</td>
<td>A combo box that can be used for selection of options in a block; when a new value is selected the simulation will refresh; it is mainly an upper level base object.</td>
</tr>
<tr>
<td><strong>SimComboBoxNumber</strong></td>
<td>A combo box that can be used for selection of SimNumbers.</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SimComboBoxString</strong></td>
<td>Holds a combo box that can be used for selection of strings.</td>
</tr>
<tr>
<td><strong>SimDoubleArray</strong></td>
<td>An array that for now holds variables of double type, the array is displayed as a comma separated list which can be used for polynomial coefficient setting.</td>
</tr>
<tr>
<td><strong>SimInteger</strong></td>
<td>A SimNumber that for now holds an integer rather than a double type.</td>
</tr>
<tr>
<td><strong>SimNumber</strong></td>
<td>A variable that holds a double type; this is the main variable used for real-time GUI parameters with a slider and animation button.</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SimStaticNumber</td>
<td>A SimNumber that has an overloaded getPanel() to simply display the value of the number and not be varied by the user.</td>
</tr>
<tr>
<td>SimString</td>
<td>A variable that holds a string, and refreshes the GUI when a new string is entered.</td>
</tr>
<tr>
<td>SimTest</td>
<td>Purely a test class to test the package and compiler paths</td>
</tr>
<tr>
<td>SimTextBox</td>
<td>A text box variable, for display of text material</td>
</tr>
<tr>
<td>SimVariable</td>
<td>The main abstract block for all types of SimVariables, which provides requirements for the reloading of the GUI mechanism.</td>
</tr>
<tr>
<td>VariableContainer</td>
<td>A container which is used within a block to hold each of the required SimVariables, from this the GUI panel is later dynamically constructed.</td>
</tr>
</tbody>
</table>