Teaching Control fundamentals for Mechatronics and Robotics - the use of JavaScript for simulation and animation.

John Billingsley
University of Southern Queensland
Toowoomba, Queensland 4350, Australia

Abstract

An opinion is expressed on the relevance to Mechatronics and Robotics students of control theory in the form in which it is at present taught. A scheme is proposed which will guide the student first through analysing a 'real' problem and simulating it, then through attempting to design a controller and finally to learning and applying those mathematical tools which are necessary to achieve success. The student is encouraged to look at the whole gamut of nonlinear and discrete time controllers, not just those which fit the theory as taught. Software platforms are discussed, with an emphasis on JavaScript, which will allow easy generation, modification and testing of algorithms by the student.

1 Introduction

As taught in standard degree courses, control theory is hard to apply to real cases. The taught material gives a mathematical solution to linear problems which have already been expressed in a mathematical form.

The student is presented with a diet of Laplace transforms, the complex frequency plane and mappings from this plane to the complex gain plane in the shape of the Nyquist plot. Nonlinearities are patched over with describing functions and discrete-time control is a morass of zero-order holds and tables of equivalent z-transforms.

To be of use to mechatronics and robotics students, the introduction to control must start from the ability to analyse 'real' problems and express them in a form which can be investigated computationally. This must allow nonlinear factors to be taken into consideration and must allow an arbitrarily determined input signal to be applied and tested.

2 Nonlinear control strategies.

Once the student has the ability to model the system, consideration can be given to determining the appropriate input signal to apply. A much wider range of control schemes can be explored than 'proportional feedback', in which the input is determined by the simple sum of weighted system variables. Many mechatronic systems require the input to be driven into saturation for a relatively small error, while others can work acceptably 'open loop'. Most will benefit from 'feed forward'.

Mechanical systems often have significant nonlinearities which have to play a dominant part in the design process. Servomotors have limitations on their driving force and linkages have hard constraints on their travel. To achieve high performance, such nonlinear systems will generally call for nonlinear controllers.

At the empirical level, it is relatively easy to incorporate all such factors into a computer model simple enough to be written 'from scratch'. Control schemes can be attempted by assigning values to the inputs which are general functions of the system variables and of external commands or disturbances.

At this point the student will become aware of problems such as stability, response time and overshoot. By 'owning' the problem, the student will be receptive to the theory which offers a solution which can bow be understood.

When practical control must be implemented, industry will most often turn to a three-term PID which is then tuned by trial and error. What is too often overlooked by those who seek to apply 'undergraduate' theory to tuning a PID controller is that the problem is essentially nonlinear. The linear theory usually only defines a small part of the overall response, that part which lies inside the 'proportional band', and its application can lead to an unacceptable controller..

Through modelling, the student can appreciate alternative forms of representing the system response which will allow nonlinear control strategies to be assessed in ways far beyond the capabilities of the conventional theory. Such general schemes can embrace all the essential aspects of fuzzy, neural and variable-structure control while taking a pragmatic approach.

3 Discrete time control.

Frequently the controller is embodied in digital form. This introduces two further complications. The first is minor, a result of the additional 'noise' implicit in signals which are digitised to a limited number of bits. More important is the restriction that the digital control can introduce correction inputs only at discrete intervals of time. In itself this is very easy to embody in the computer model, but its theoretical analysis is usually embroidered with the relationships between tables of Laplace and z-transforms and the added complication of the zero-order hold.

The discrete time state equations can be evaluated more simply by simulating the continuous system over one sampling interval, starting from a variety of initial conditions. These are equal in number to the

columns of the A and B matrices. The method is easy to follow intuitively. By using the operator 'z' to signify 'next', applied as a multiplier to the variables on the left-hand side, transfer functions in z can be derived . These become the tools by which a controller can be designed, which can then be tested on the model in full detail.

It can be shown that there are excellent control schemes in which a mixture of discrete and continuous feedback does not allow an analytical solution. That is not a good reason to discard them. Discrete control schemes in which corrections are not 'clocked' at regular intervals are also outside the capability of z-transform analysis, but there may be practical reasons why they have to be applied.

The rounded control graduate should have the analytic 'toolkit' available to apply when it is appropriate, but must be able to generalise the control design process to solve the practical problem which is presented in the way which is really needed.

4 Nonlinear examples

A simple position controller makes a good worked example. The student is instructed to design a system which can move a one kilogram load over a distance of one metre, reaching a final accuracy of one millimetre in one second and resisting disturbing forces up to five Newtons.

Pole assignment is misleading in this case. The phase-plane is found to be an appropriate design environment and an acceptable solution is found with poles in the proportional band which represent time constants of 0.16 second and 0.6 milliseconds - a damping factor of eight!

A photographic paper-coating process included a magazine holding tens of metres of paper, an 'unwinder' motor driving the massive paper reel, a passive 'float roll' to take up minor variations and a drive roll servo for filling the magazine. As each reel comes to an end, the unwinder is stopped while paper continues to be delivered from the magazine. When the new roll has been pasted onto the paper web, the unwinder starts up again. It was found that all too often the float roll would hit its stop and the whole process would be brought to a halt, with the loss of thousands of dollars worth of photographic paper drying in the plant.

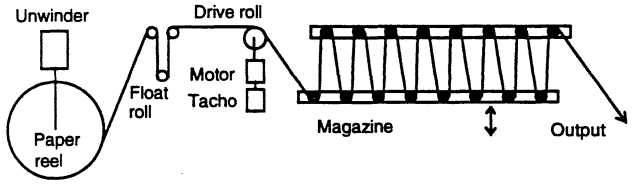

Fig.1 Schematic of a photographic paper coating process

The faulty controller involved feedback from the float-roll, having less than a meter of travel, to the unwinder which has a moment of inertia which changes by a factor of sixty - not an easy loop to close even with the most sophisticated of strategies. The level of the magazine, holding up to a hundred metres of paper, was fed back to the drive roll motor-tacho system which fills it.

The solution is of course to use the drive roll servo to keep the float roll within close limits. A rapid, responsive system is used to maintain a the value of a variable which has only a small safe range of travel. The magazine sensor signal is fed to the unwinder motor, together with an added velocity signal from the drive roll tachometer. The magazine can accommodate tens of metres of unwinder variability.

Simulation is an ideal method of testing designs which start from the topology of the feedback scheme, rather than from poles in the frequency domain.

5 Simulation environments

The simplest form of simulation uses Euler integration of the state equations , where the value of x is updated to x + dx/dt * dt as the time advances by an interval dt. dx/dt is in turn a simple function of the state variables and the inputs, so the process requires little more than the assignment statement common to nearly all computer languages. The process of updating the variables each iteration is contained in a program loop, also common to nearly every language.

It is highly desirable to represent the output graphically, preferably in real time. It is also an advantage to be able to interact with the system in real time to inject disturbances.

If the students are to modify and extend the code with ease, an interpreted language has advantages over one which requires lengthy compilation and linking before it can be run.

When DOS systems were in more common use, QBASIC presented most of the answers to these requirements. Its syntax was straightforward, while it had most of the features of Pascal (without the semicolons). Keyboard or mouse inputs were easy to interface and graphic output was achieved with half a dozen simple functions. The software with extensive 'help' documentation was bundled free with the DOS operating system.

Although available as an option with Windows 95, its inclusion in the installation is no longer automatic. Its future is debatable.

Visual Basic provides a superset of QBASIC and has superior graphics, but must be purchased as a special package. A package such as MATLAB can be used as a simple programming environment, but lacks the ease of real-time operation. C and its dialects can be used, but the cycle of edit-compile-link-load-and-test is somewhat cumbersome in most popular versions.

With the exploding use of HTML web material, modern browsers support embedded software. Java applets can be downloaded to complete the page and are then executed. However these are not the real answer, requiring a compiler to modify them. JavaScript has much in common with Java, but is interpreted from source text which can be viewed within the page.

By storing the page locally, it can be edited with a simple text editor to vary the software. Code has similar syntax to C and real-time input can be derived from key-stroke events or by the use of 'form' controls. Achieving graphic output requires some ingenuity, however. The task is complicated by the present conflict between Netscape and Microsoft, since there are serious

differences between their implementations of what was supposed to be a standard language. At present Netscape Communicator 4.0 has a feature which tips the present balance strongly in its favour.

6 Animation with JavaScript

First a simple animation technique is presented which can be implemented in both Netscape and Internet Explorer - though details of the code are different. Only the Netscape syntax will be described here.

Navigator 4 supports LAYERs. A layer is a graphics plane which can be repositioned at will by a `MoveTo` instruction. Each plane can contain images or text and the page can hold many layers. These can be animated by JavaScript instructions, being moved in response to the changes of the state variables in the simulation.

Although the possible movement does not include rotation, alternative GIF images can be substituted to achieve a similar result if desired.

The following code moves a bold red 'O' along a damped second order response:

```
<HTML>
<HEAD>

<SCRIPT LANGUAGE="javaScript">
var xx=200.0;
var v=0.0;
var tt=0.0;
var tmax=500.0;
var dt=1.0;

function stepmodel(){
  if (tt<tmax) {
    v=v-(.005*xx+.02*v)*dt;
    xx=xx+v*dt;
    tt=tt+dt;
    document.ball.moveTo(tt+10,200-xx);
    setTimeout("stepmodel()",4);
  }
}
</SCRIPT>
</HEAD>
<BODY>

<DIV id="ball" style="position:absolute;
left:20; top:200">
<B><FONT COLOR="#FF0000"><FONT SIZE=+1>O</FONT>
</DIV>

<P>
<A HREF="#"
onclick="tt=0.0;xx=200.0;v=0.0;stepmodel()">
GO !</A>
</BODY>
</HTML>
```

The layer is defined by the 'DIV' tag, one of two alternative forms of syntax. It is moved by the `MoveTo` method in the function `'stepmodel'`. The state variables are updated with just the two `'v='` and `'xx='` lines, while tt is also updated to give movement along a horizontal time axis.

The line `'setTimeout("stepmodel",4);'` causes the stepmodel function to be restarted every four milliseconds; it is invoked for as long as the condition `(tt<tmax)` remains true.

With similar code, an undamped mass-spring-wheel suspension can be simulated with GIF images providing an animated picture.

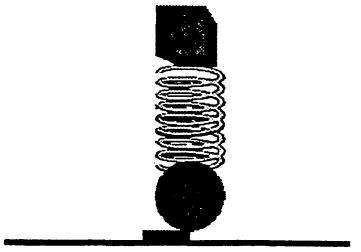

Fig.2. Animated mass-spring system.

Although entertaining, these do not give results which can be represented as a printed record. The second method enables graphs to be plotted and saved in a document.

7 Graph plotting in JavaScript

In itself, JavaScript has no provision for ceating graphical images. However Navigator 4 has a feature called LiveConnect which allows properties and methods to be employed from the Java library. In particular, Java.awt.Graphics has an extensive set of functions which fulfill all the standard graphic requirements.

Gaining access to these functions is still not trivial. The programming structure is bound up with the 'object oriented' ideology and necessary ceremonies have to be observed. While many objects have 'constructors' which enable them to be created, such as new browser windows, Graphics is resticted to operate on a component which already exists. Once such a component can be located, however,its graphics properties can be assigned to a variable, as in

```
var g=component.getGraphics()
```

and the way is open to apply any of the graphics methods.

The only way I have found to obtain such a component is to load it as an applet, Graph.class. This applet has virtually no content at all, serving merely to invoke the awt.Graphics library. Its source is

```
import java.awt.Graphics;
public class Graph extends java.applet.Applet {
}
```

The inclusion of the applet in the HTML page also defines a frame in which the graph is to be drawn.

Care must be taken to delay execution of any simulation code until the applet has completed loading, otherwise a 'no properties' error will be encountered. In the example which follows, a form is added which allows gain and damping ratio to be modified, then plots a response from an initial displacement when a button is clicked. An effective way to clear the screen when a number of plots have been drawn is to click the browser's 'reload' icon.

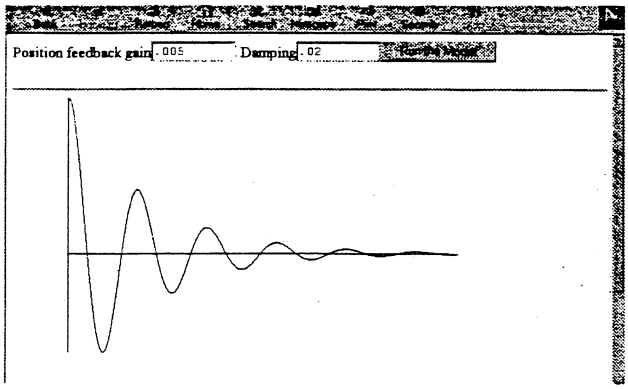

Fig. 3. Graphical plot using JavaScript.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="javaScript">
var a=.005;
var b=.02;

function runmodel()
{
var g=document.Graph.getGraphics();
var x=200.0;
var oldx=x
var v=0.0;
var oldv=v
var tt=0.0;
var tmax=500.0;
var dt=1;

while (tt<tmax) {
  v=v-(a*x+b*v)*dt;
  x=x+v*dt;
  tt=tt+dt;
  g.drawLine(tt+10,200-oldx,tt+10+dt,200-x);
  oldx=x;
  }
g.drawLine(10,200,500,200);
g.drawLine(10,0,10,400);
}
</SCRIPT>
</HEAD>
<BODY >
<FORM name=gains>
Position feedback gain
<INPUT TYPE="text" NAME="g1" VALUE=".005"
SIZE=10 >
Damping
<INPUT TYPE="text" NAME="g2" VALUE=".02"
SIZE=10 >
<INPUT TYPE="button" NAME="doit"
VALUE="Run the Model"
onClick="a=eval(gains.g1.value);
b=eval(gains.g2.value);
javascript:runmodel();" >
</FORM>
<CENTER>
<HR><APPLET
    code=Graph.class
    name=Graph
    width=640
    height=480 >
</APPLET></CENTER>
</BODY>
</HTML>
```

These examples are 'stripped down' to show that the essential content is quite small. Actual examples are more polished, but care must be taken that the 'gloss' does not get in the way of the student's understanding.

8 Conclusions

As web-based teaching material becomes ever more fashionable, we can expect the student to be sitting in front of a computer. If we can use the web material to deliver a simulation environment in which control problems can be 'felt and tasted' we might draw closer to the goal of making control courses relevant to the solution of real mechatronic problems.

I would not for a moment advocate the omission of frequency domain techniques and transform theory. They must be grasped by the student as the solution to a problem, however, not merely as an arcane ritual to be memorised for an examination.

References

[Billingsley 1989] John Billingsley. Controlling with Computers: Control theory and practical digital systems. McGraw-Hill, Jan 1989, ISBN 0-07-084193-4.

[Billingsley 1991] John Billingsley. On the design of position control systems. *Proceedings IEE Part D*, vol 138, no 4, pages 331-336, London, July 1991

[Billingsley 1995] John Billingsley. A mechatronic cynic's view of control theory. *IEE Computing and Control Engineering Journal*, vol 6, no 5, pages 243-244, London, October 1995.