PRACTICAL CONTROL LOOP TUNING USING A MATLAB/SIMULINK TOOLBOX

A. Adgar*, C.S. Cox*

* University of Sunderland, UK.

Keywords: Computer aided control system design, Modelling and simulation, Non-linear systems, System identification and signal processing, Utilities

Abstract

This paper describes a suite of software which has been developed at the University of Sunderland. The software takes the form of a toolbox for the MATLAB and Simulink environment. It assists in the many stages of designing a feedback controller for a real process.

The software may be used for data collection, data pre-processing, process modelling, controller design and finally controller evaluation. The software has been demonstrated on a laboratory based process trainer and several controller architectures have been designed and evaluated.

The software presented here is thought to provide several advantages over some of the commercial loop–tuning and controller design software. These are discussed within the paper.

1 Introduction

It is well known that many complex processes are often satisfactorily controlled by means of simple three term (PID) controllers. In fact, many industrial solutions are realised using only the P and I terms but it has been widely reported, that even after careful instruction, plant operators still have difficulty installing and operating such systems. It is obvious that a feedback system is of little value if the tuning parameters are poorly selected. For systems with long time constants (of the order of several minutes) the semi–empirical methods favoured by some users can be both time consuming and tedious.

Consequently, any help offered to the process operator to ease this tedious task and at the same time produce consistent closed–loop behaviour on the process is clearly very desirable. In many process industries (e.g. the water industry) it is often found that the system dynamics display appreciable time delay. The use of PI(D) control in these situations is not recommended. Instead, some form of dead–time compensation is necessary. Whilst improved performance is achievable it is accomplished at the cost of a more complex control strategy. In all of the above scenarios, it is obviously desirable if some kind of advice could be generated automatically. This would be especially useful if the advice would also embrace the modelling phase that inevitably accompanies any design strategy. The software toolbox described in this paper has been specifically designed to produce this kind of assistance.

2 The Software

The authors have developed a comprehensive set of software tools running under the MATLAB and Simulink platform to assist in performing real–time modelling and control experiments. MATLAB and Simulink are closely related computer packages which provide an environment that allows the integration of computation, visualisation, and programming in an easy–to–use environment. Typical uses include: mathematics and computation; algorithm development; modelling, simulation, and prototyping; data analysis, exploration, and visualisation; scientific and engineering graphics; and application development, including graphical user interface building.

The name MATLAB stands for matrix laboratory, since most of the functionality of the software comes from matrix manipulation. It has many uses therefore in mathematics, engineering, and science. In this case the software has been utilised for teaching process modelling and control systems design. The Simulink environment provides a drag–and–drop interface for building mathematical models of systems in the conventional block diagram form.

The toolbox which has been developed contains numerous MATLAB functions and Simulink blocks to assist many of the tasks described above. There are many types of process input signals, process controller structures and interfaces for typical laboratory process training equipment. The standard Simulink block libraries may also be included in any activities being performed.

The typical activities of data collection, data pre–processing, process modelling, controller design, implementation and evaluation can be performed very rapidly. The simple drag–and–drop interface allows for easy adjustment of the various parameters, the display of process variables, the storage of data, etc.
3 The Test System.

The test system used to demonstrate the application of the toolbox was the process trainer as shown in Figure 1. This apparatus is known informally as the ‘hairdryer’ for obvious reasons. The system consists of a fixed speed blower which propels a current of air through the plastic tube. The air flow rate may be modified by the use of a damper. The air is warmed using a simple heating device consisting of a mesh of wires just inside the entrance to the tube. The heating effect is controlled by the voltage supplied to the mesh. The air temperature may be measured at one of three locations along the tube with a thermistor. Hence the test equipment forms a simple single–input single–output system.

![Figure 1: Process trainer test system used for the data collection, modelling, controller design and controller evaluation studies.](image)

4 The Computer Hardware/Software

The software toolbox runs on a PC which already has MATLAB and Simulink installed. The PC is also fitted with an Amplicon data acquisition card. Driver software has been written for several types of card so that the software can be used on any of the computers in our laboratories. This allows the reading and writing of analogue signals to and from the test equipment.

5 The Controller Design Procedure

The main stages involved in the design of a feedback controller are described in the subsections below. Each subsection will show how the software is used and the typical results achieved at each stage.

5.1 Data Collection

When using the software toolbox, data may be received from and sent to the outside world via the interface card in the PC. These features may be used to collect data from a laboratory test rig or an industrial process. Signal conditioning units have been built to allow the sampling of voltages or currents. Typical scales which are used include -10 to +10V, 0 to 10V, 4 to 20mA.

To read or measure signals from a sensor (process output) via the ADC we use the function:

```matlab
y = changet(chan_no);
```

To write or send signals to an actuator (process input) via the DAC we use the function:

```matlab
chanput(chan_no,sig_lev);
```

There are some other functions available which assist the collection of data from processes in real–time. These are very flexible but require some simple programming knowledge. It is generally much easier to use the Simulink drag–and–drop environment and this is the route which is described here.

The Simulink model is shown in Figure 2. There are several points to note about the blocks used here. Firstly, the real–time block is important as this forces the simulation to run in real–time. Hence the data collection is triggered at the correct frequency. In this case the sample time was chosen to be 100ms. The sample rate which can be achieved for a given experiment is dependent on many factors: the windows operating system, the Simulink software and model complexity and the PC processor speed. On a typical lab PC the system can achieve 100ms sample times for the standard data collection and controller implementation tasks. When working with processes which require faster sampling than this, then some kind of commercial system is needed. Examples being LabVIEW, xPC target for windows or dSPACE.

A simple step test is to be performed so a step block is used as the input (parameters specified are the step time, the initial value and the final value). There are other signals that may be used such as a Pseudo Random Binary Sequence, a sine wave, an impulse, etc., but the step is probably the simplest. The PT326 block is used to specify the DAC and ADC channels to be utilised for communication to the process. There is no actual model of the process within this block. The process inputs and outputs are collected and stored in variables \( u \) and \( y \) respectively. Both are also sent to the scope block which reveals a real–time plot as the data collection is progressing.

Note that a variable, \( tv \), is also recorded. This variable stores information about the actual time a sample was collected. Hence, it can be compared to the variable \( t \) to ensure that the simulation is running in real–time and that there are no problems in achieving the sample rate specified on the PC being used.

The user defines a fixed sample time and then the data collection experiment is controlled at this fixed sampling rate. The user also defines starting and stopping times.
5.2 Data Pre–Processing

As can be seen from Figure 3, the step response covers only a small time period of the graph. Also there is some transient response information at the start of the plot relating to the process reaching its initial test level. This should be removed before any analysis is done on the data. This is achieved by using the MATLAB function:

```matlab
» [t,y,u] = aacrop(t,y,u);
```

The user simply uses the mouse to select the desired start and stop times for the data. In this case the data between 7 and 15s was selected.

5.3 Model Estimation

Once the data has been suitably pre–processed a process model may be estimated from the data. In this initial example the model structure chosen is that of a first order plus dead–time (FOPDT) equation. This model structure is relatively simple and is used by many controller design techniques as described comprehensively by O’Dwyer [2].

The model structure is shown in transfer function form in Equation 1. It is very simple and requires only three parameters (K, T and D) to be specified.

\[ G(s) = \frac{Ke^{-Ds}}{1+sT} \]  

(1)

The function required to calculate the three parameters is shown below:

```matlab
» [K,T,D] = fopdt(t,y,u);
```

The software uses the method of Nishikawa et al [3]. In this particular case the software calculates the parameters to be as shown in Table 1. Other methods of model parameter estimation based upon optimisation techniques may also be used. As can be seen the process dead–time is slightly smaller than the process time constant.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>Process Gain</td>
<td>1.13</td>
</tr>
<tr>
<td>T</td>
<td>Process Time Constant</td>
<td>0.43</td>
</tr>
<tr>
<td>D</td>
<td>Process Time Delay</td>
<td>0.36</td>
</tr>
</tbody>
</table>

Table 1: Parameters of the FOPDT Model estimated using the data collected during the process step test.

Graphical output is also generated automatically by the software. The plots in Figure 4 show the input step as well as the actual collected sampled data (the dots) plus the continuous line which represents the output from the FOPDT model to the same step input. This kind of information is very useful for the user because it provides knowledge about the accuracy of the characterisation.

5.4 Controller Design

The actual design of the controller is relatively straight forward once the FOPDT model parameters have been estimated. Another function may be used to design a simple feedback PI or PID controller for the above process. The function inputs are the model gain, K, time constant, T, and delay, D, (all evaluated above) to calculate the controller parameters. The actual function call is:

```matlab
» [Kc,Ti,Td] = despid(K,T,D,’ZN’,2)
```
The additional inputs define which controller design method to use (in this case Ziegler–Nichols has been used) and which controller structure to use (1 = P, 2 = PI, and 3 = PID). The controller is defined by the controller gain, $K_c$, the controller integral action time, $T_i$, and the controller derivative time, $T_d$. The function call shown performs a Ziegler–Nichols controller design, but several other design methods are available.

The results for a PI controller designed using the method of Daniel & Cox [1] yields the following controller parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_c$</td>
<td>Controller Gain</td>
<td>0.52</td>
</tr>
<tr>
<td>$T_i$</td>
<td>Controller Integral Time</td>
<td>0.43</td>
</tr>
<tr>
<td>$T_d$</td>
<td>Controller Derivative Time</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 2: Parameters of the FOPDT Model estimated from the step test data collected.

5.5 Controller Evaluation

The controller may then be evaluated on either the model (in simulation) or on the actual process itself (in real–time). Here, the controller will be evaluated in real–time on the process. The Simulink model is set up as shown in Figure 5 and, as can be seen, the set point input is a sequence of multi–level steps. The process set–point ($r$), the controller output ($u$) and the process variable ($y$) are all logged for the duration of the trail. The controller is evaluated over a total time period of 1 minute with the sample time set to 100ms. The response of the closed–loop control system can be seen in Figure 6.

The advantages of performing a controller design in this way are numerous. The design process is flexible, simple to implement and the software provides easy access to any of the signals around the loop for analysis purposes.
5.6 Advanced Control

As a final exercise, an advanced controller was designed and implemented using the software. This only requires the replacement of the PID block with the new controller block. In this case a pPI (predictive PI) controller was designed. The same trial as was done with the PI controller was then performed using this more advanced controller.

As can be seen from Figure 7, the pPI controller provides much tighter control than the PI controller. The responses to the set-point changes are less oscillatory and there is less overshoot. This is as expected and the difference would be more marked if the process time delay was larger in comparison to the time constant.

6. Conclusions

The MATLAB/Simulink Toolbox software described here provides an ideal environment for research and development studies. All the stages of controller design such as data collection, process modelling and controller testing may be performed using the software.

It is important to emphasise that the design carried out above hardly exploits the wide range of design features available with the Toolbox. Model development for continuous–time situations also includes SOPDT (second order plus dead time) and SOZPDT (second order with zero plus dead time) characterisations. In addition three different forms of time delay compensation are available and may be immediately implemented using individual Simulink blocks. Advanced discrete–time modelling is also facilitated. The discrete–time models derived are used for model predictive control designs. Two methodologies are available. These are GPC (generalised predictive control) and PIP (proportional–integral–plus).

The software described here has the advantage over other loop tuning software primarily in terms of its flexibility. New modelling algorithms and controller structures may be easily incorporated into the library. Users may develop their own tailor–made routines to suit the particular application being studied. The software can also be extended by taking advantage of the wide range of other MATLAB and Simulink libraries already available. This provides the opportunity of using statistical analysis and performance evaluation, neural network modelling, fuzzy logic control, and many other ready–made solutions.

The main problem faced with using this software is the final implementation of more complex control strategies within off–the–shelf controllers or PLCs. This is possible in some devices that have specific configuration software. It may also be possible to implement coded versions of the control strategies in PLCs using the IEC1131 standard.

Acknowledgements

The authors would like to thank the staff and students (both past and present) of the Control Systems Centre at the University of Sunderland for their contributions to this work.

References

