EEE482 PROJECT

Contents

Abstract, Introduction, Software, Installation, Assignment, Software Manual

Abstract

The objective of this project is to illustrate the use of state-space concepts and computational tools in a fairly complete controller design problem.

Introduction

The application of interest is the temperature control problem for Diffusion/CVD furnaces used in semiconductor manufacturing. Diffusion processes are an important component of semiconductor manufacturing. In these, the temperature of the silicon substrate is elevated to a sufficiently high level to enable the diffusion of a dopant or oxidation of the surface. There is a variety of Diffusion/CVD furnaces in terms of construction materials, size and orientation. Typical furnaces are divided to 3-5 heating zones, determined by the number and position of distinct heating elements as well the position of the thermocouples. In particular, two sets of thermocouples are used to measure temperature at different locations along the furnace, near the heating element (spike) and near the wafer load (profile). Repeatable and accurate temperature control of such processes is essential to achieve consistent process results. Typical control objectives include disturbance attenuation as well as minimal overshoot to ramp-up operations and temperature uniformity across the load, often referred to as zone matching. Naturally, as the industry moves towards smaller device geometries, these control specifications become more stringent.

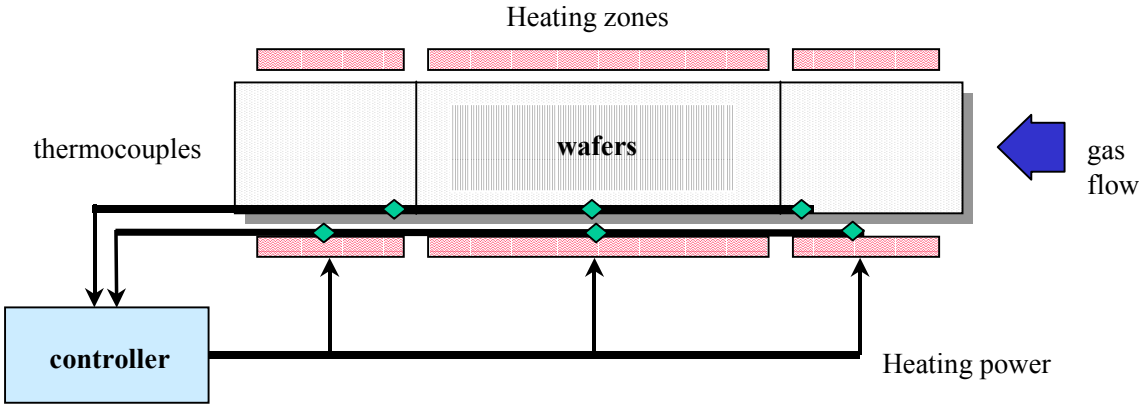

Figure 1. **Simplified schematic of a horizontal CVD furnace**

With the retrofit of older furnaces being of special interest, additional considerations are imposed on the controller design approach, such as quick design turnaround time and reliable first-time designs. A high level of software integration is also desirable (few intuitive design parameters) to enable its use by non-experts and, potentially, by field service personnel. The approach taken to solve this problem consists of a multivariable controller that is designed based on a model identified from input-output data. A linear system is used to describe the power-to-temperature dynamics around an operating condition. This is motivated by an approximation of the underlying physics of the process and has been validated by actual data as a local model.

It goes without saying that the process is (mildly) nonlinear and, because of its importance, accurate models have been developed (numerical solutions of nonlinear PDEs). The codes are commercially available and rather expensive. Their primary use is in the design of the furnaces, helping to decide the size of the heating elements, furnace geometry and configuration etc. On the other hand, their usefulness in control systems design is limited by complexity and availability issues. Significant effort needs to be expended in determining the required physical properties of the furnace and achieving the detail necessary for accurate prediction of fairly fast dynamics (~1min time scale). This may be impractical, especially for retrofitting

older furnaces with high-performance control systems. With this in mind, modeling via system identification is a viable and perhaps more appropriate alternative. Its disadvantage is that the obtained model is valid only locally, around the identified steady-state. This, however, is not very serious since the process nonlinearities are mild and high-performance is only required around the operating steady-state.

In terms of difficulty, this controller design problem is mild. The main problems are associated with the near-integral action of the process (i.e., slow dynamics relative to the intended closed-loop bandwidth). Zone-matching, although not trivial, can be achieved since any right-half plane zeros are usually well-above the intended closed-loop bandwidth. On the other hand, such problems become interesting as they bring up performance optimization issues. While closely related to the achievable bandwidth, this is a largely unresolved question, especially since from a processing point of view the specifications are rather vague. It is often left up to the designer to define suitable control objectives that may (and will) be case dependent. For this reason, it is appropriate to give a brief discussion on the motivation behind the control objectives and their relation to the process.

- The typical process begins with the furnace kept at a low temperature (500-600 deg.) when the room-temperature wafers are loaded. After the furnace temperature is stabilized, it is ramped-up to the processing conditions, specified by the so-called recipe (e.g., 800 deg.). Again, the temperature is left to stabilize and then processing begins by introducing the reactant gases. During the processing interval the temperature should be kept constant. At the end of the processing interval the furnace is cooled down and the wafers are unloaded.
- The timing of each step is established by initial experimentation by the process engineer (furnace qualification). Even though this step will account for any deviations in the settling time, it is clearly beneficial to achieve stabilization as quickly as possible. Quick stabilization translates into shorter processing times and better utilization of the tool (furnace).
- Temperature uniformity across the load is important so that all wafers undergo identical operations. This is most crucial during the processing interval. But it is also desirable at all times since raising the wafer temperature induces an annealing effect on the existing dopant concentrations in the wafer and even on the pattern geometry itself. The annealing effect is cumulative and its rate increases sharply with temperature (Arrhenius dependence, ~exp(-k/T)). Ultimately, variations in the temperature across the load translate in variations in the device properties and can cause yield reduction. A consequence of this is that overshoots must be kept small and all temperatures must be kept close to each other. In addition, the controller performance at the loading temperature can be compromised in favor of optimizing performance at the processing conditions.
- In terms of priority, zone-matching and minimal overshoot are more important than settling time, within reason. Hard limits are furnace-dependent but some loose target values are: zone separation < 1deg, overshoot < ½ deg, settling time (line-out) < 15 min from end-of-ramp.
- The ramp rate may be adjusted to minimize the transition time between steady-states. Such a procedure should avoid control input saturation either on the way up or during stabilization. Often, the limiting factor is the ability of the controller to “turn the corner” where the control inputs may saturate low and an overshoot appears. Other limiting factors are associated with stress on the wafers, caused by quick temperature changes (radial non-uniformity). In our simulation example, 20 deg./min is achievable.
- Semiconductor manufacturing processes are typically occurring in a controlled environment and are very repeatable. Still, there are several types of disturbances that affect the furnace temperature. One is the interaction with other furnaces that may be heated or cooled. This type of disturbance appears in stacks of horizontal furnaces only and has very low frequency content. It can cause problems in identification experiments but its impact on the closed-loop system is usually negligible. Another type of disturbance is the appearance of a heating/cooling source inside the furnace. For example, gases are turned on or off according to the recipe and have a cooling effect on the furnace (a few degrees if left uncontrolled). Also, certain processes require the presence of water vapor as one of the gases. This is created by introducing a mixture of hydrogen and oxygen in the furnace and igniting it.

Software

A Simulink file that contains the basic components for the simulation of the furnace temperature control problem is given in **sim_pla.mdl**. (MATLAB V5.x compatible; a V4.x compatible model is available upon

request). It can be executed by opening it with MATLAB (type its name) and selecting Start from the Simulation menu. It contains the following components:

- Step/Sequence sources (target temperatures from, e.g., a recipe).
- An excitation generator. This requires the sequences TX and RX in the workspace. The block is self-initialized by loading data from the file **excit.mat** that should be in the same folder. The default excitation sequence begins at time 150 min. and lasts approximately 18 min. (2048 samples at 0.5 sec sampling interval). If you want to change the excitation, you need to create a vector TX (n x 1) of time instants and a vector RX (n x 3) of deviations from the steady-state and save them under the same name.
- A generic PID controller that can control the furnace temperature; its tuning is not optimized in any sense. You may re-tune it as you see fit. The PID block contains three single-loop PID controllers with a filter. You can change their parameters by double-clicking on them and editing the appropriate fields. The default PID parameters are: proportional gain 0.02, integral time 2.5, derivative time 0.5.
- A block named FURNACE that contains the description of the plant model. It has no adjustable parameters and it is self-initialized with the file **plant_da.mat**, which should also be in the same folder. The input to this block is power fraction (3 element vector, each between 0-1) and the outputs are spike temperatures (unused here) and profile temperatures (3 element vector). The FURNACE will "shut-down" automatically by zeroing its internal inputs if the temperature exceeds 950 deg.

For convenience, some basic controller blocks have been included in the Simulink file **cblocks.mdl**. This contains blocks for single-loop PIDs with and without anti-windup and a multivariable (3 x 3) controller with an observer-based anti-windup. The multivariable controller has a discrete-time state-space description [Aci, Bci, Cci, Dci] and ant-windup gain [Lci]. It also contains an output filter (feedback compensator) with a discrete-time state-space description [Afo, Bfo, Cfo, Dfo]. Their sampling time is specified by the parameter TSAMPLE. All these parameters must exist in the workspace. They can be initialized by running the m-file **h_preh.m**.

The basic software for identification and controller design is given in the form of several (43) m- or p-files. All of these files must be in a folder in the MATLAB path. For convenience, these files are zip-compressed and contained in the file **mpfiles.zip**.

Installation

Download the files sim_pla.mdl, cblocks.mdl, excit.mat, plant_da.mat and save them in a folder. Download the zip file mpfiles.zip and extract the contents in the same folder. (mpfiles.exe is a self-extracting version of that file, in case you do not have the Winzip software.) If you open MATLAB by double-clicking an m-file (or any other MATLAB-associated file) in the same folder, the folder is automatically included in the MATLAB path. Otherwise, you need to add the folder in the path. You can save your work in the same folder. The entire folder should fit in one floppy disk for back-up purposes.

Assignment

Design a temperature controller for the furnace.

- *Deliverables* (e-mail or hard-copy):
    1. A file containing the controller parameters Aci, Bci,… suitable to initialize the given multivariable controller block. Or, a simulink controller block with the same I/O structure, accompanied by the necessary initialization m- and mat-files.
    2. Appropriate documentation explaining the design steps and discussing the controller validation results. Include graphs of the controller frequency response (singular values) and any pertinent graphs of time responses.
- *Performance Objectives:*

  Optimize the controller performance, as you see fit. Justify any trade-off decisions. The controller performance will be judged based primarily on the two recipes included in the **sim_pla** model:
    1. Stabilization at 550 deg., ramp-up to 800/850 deg. at ~20 deg/min, stabilization, and processing until time=95min, ramp-down. During processing, two disturbances enter the system. The first emulates a hydrogen torch effect and the second emulates the effect of an additional cold gas flow. (Note: The following performance measures are achievable: 0.5

deg. overshoot, stabilization within 0.5 deg (line-out) in 15 min from end-of-ramp, 12 deg peak-error after the 1st disturbance, 2 deg. undershoot, 15 min line-out, 2 deg. peak-error after the 2nd disturbance.)

2. Stabilization at 550 deg., step-up to 800 deg., stabilization, and processing until time=95min, ramp-down. (Note: The following performance measure is achievable: 15 deg. overshoot, settling within 1 deg. in 20 min, line-out in 30 min.)

- *Constraints:*

  The controller sampling time cannot be smaller than 0.5 sec (0.5/60 min; use min as the time-measurement unit). Due to "hardware" limitations, the data collection software cannot handle more than 4096 sample points. (You may try longer excitation sequences but your final controller should be based on a sequence of 4096 samples or less.)

- *Suggested Procedure:*
  1. Familiarize yourselves with the system; try different PID values and reference inputs.
  2. Perform an identification experiment with a suitable excitation sequence at the desired steady-state. You may use open-loop or closed-loop identification, and excitation at the reference or the plant input. For this, you may modify the Simulink block diagram as necessary. However, you may not alter the Furnace block in any way. Currently, there is no emulation of a data collection procedure. You need to extract the identification data from the workspace variables T,U,YP. For example, for a 2048-sample data sequence starting at time=150, you may use the commands:

     >> index = max(find(T<=150))

     >> t = T(index:index+2047); y = YP(index:index+2047,:); u = U(index:index+2047,:);
  3. Use the function **mvidi** to perform a continuous-time multivariable system identification. The program prompts for a file-name where all the pertinent information is saved.
  4. Use the function **h_hinf** to perform an H-infinity, loop-shaping controller design. The program prompts for a file-name to save the continuous-time state-space representation of the controller.
  5. Use the procedure **h_preh** to discretize the controller, design the anti-windup gain and a simple feedback filter. This will define all the necessary variables for the controller initalization. Note: The feedback filter is defined in terms of the poles and zeros of an equivalent prefilter at the reference input. Also, it is interesting to see the effects of the anti-windup modification (or lack thereof) by setting the observer gain to zero (Lci = 0*Lci;) and running a simulation.

Software Manual

The syntax and usage of the three important functions is described below. The rest of the functions are called by the basic ones. In all user-input prompts hitting <return> will accept the default value. The functions require the Control Toolbox and Robust Control Toolbox.

1. mvidi: Multivariable, continuous-time system identification (for more details on the algorithm used, see Chapter 5.3 of the notes).

Usage: `>> mvidi(t,u,y);`

t: time vector, or samplng time interval (scalar)

u: system input

y: system output

Outputs: A system identification file (name specified by the user at the end of the program). The file contains the continuous-time state-space description of the plant in the variables [Apr,Bpr,Cpr,Dpr] and the initial conditions in X0. The model uncertainty (multiplicative plus feedback) is in MUNC with a corresponding frequency vector in frun.

User-input prompts: The function begins with a menu screen. The available options are:

```
------ 2. Auxiliary Filters (Order and Bandwidth)
------ 4. Fitting parameters: weighted LS
```

```
------ 5. Prefiltering
------ 6. Estimation Constraints
------ 8. Trace flag, estimation/validation flag
------ 0. Exit Menu and Begin
```

In 2, you specify the order of the transfer function used to model each output. The bandwidth has a loose interpretation as frequency weighting up to that frequency. Both can be vectors, if different order transfer functions with different bandwidth are desired for each output.

In 4, you specify the regularization method and threshold (similar to solving a LSMN problem). Normally, you should not need to make any changes to the defaults.

In 5, you can specify a prefilter in terms of poles and zeros. The prefilter acts on both inputs and outputs to effectively implement a frequency-weighted estimation. In the same item, you can specify the length of a window to be used for determining the initial conditions of the prefilter.

In 6, you can switch the estimation of a direct throughput (D) on or off. This may be helpful in avoiding problems from zeros that have high bandwidth but are close to the right-half plane.

In 8, the first switch selects estimation or validation. The latter can be used to obtain uncertainty estimates for a different set of data but with a previously derived model. Its calling format is mvidi(t,u,y,'filename'); the second switch specifies whether the mean should be extracted from the data.

Entering 0 will begin the estimation. The identified model poles and zeros and DC-gain matrix are displayed, and the singular values are plotted. Then the program prompts for a graphical comparison of the time responses. In this case, the identified model is simulated with the input data and the result is compared against the output data. The overall additive error is also plotted. (The figures are displayed in sequence with a 3sec delay).

Then the program prompts for the estimation of the uncertainty. The estimation error (equation error) is displayed and after whitening it is used to produce estimates of the multiplicative and feedback uncertainty. These are displayed as constraints on the sensitivity (S) and complementary sensitivity (T). Then the program prompts for the selection of a target loop shape. The first input is a 2-dimensional vector with the minimum and maximum corner frequencies in S and T (i.e., S-bandwidth and T-bandwidth). These numbers need not be accurate estimates. They are only used to set-up the zeros of the weights in the control problem.

The program will now prompt for specifying the roll-off rates of T and S for each channel. For our type of problems, 3 and 2 are appropriate. The program will attempt to guess reasonable T and S targets for each output channel from the uncertainty data. However, because of our channel-matching objective, the same target should be chosen for each channel. Reply with 0 to the "Done?" question and graphically adjust the target T and S. In a typical run, use the left button to adjust the T-corner frequency, click on the yellow box to automatically adjust the target S and the red box to exit. Proceed with the rest of the channels by answering "1" to the prompt
"Enter 1 to use the previous T/S [0]"

The next display is the estimated robust stability condition, which should be less than unity. If satisfactory proceed with replying <return> to the prompt
"Repeat with different weights? (1=yes) [0]"
and then provide a file name where the results will be saved.

2. h_hinf: Multivariable, continuous-time system controller design via frequency loop-shaping. (For more details, refer to Chapter 5.4 of the notes.)
Usage: `>> h_hinf('file_name');`
filename: name of an ID-file containing the plant model and plant uncertainty (e.g., as created by mvidi).

Outputs: A controller file (name specified by the user at the end of the program). The file contains the continuous-time state-space description of the controller in the variables [Acr,Bcr,Ccr,Dcr].

User-input prompts: The function begins with a prompt for the approximate closed loop bandwidth in order to set-up the frequency vectors. It then asks for a specification of the target T and S. The default specifications are those selected at the identification stage. Normally, the answer is 0 (no) unless you want to use the same identified model but design a different controller for it. (In that case, the selection of the T and S targets is similar to mvidi.)

The next prompt is the control weight (W_2 in 5.4.2 of the notes) where you normally select a small number. Then, the program enters the H-infinity computation of the controller, displaying the sequence of gamma-iterations. At the end you are prompted to reduce the compensator. Normally you reply with 1 (default value), unless the H-infinity solution was not successful/suitable in which case you need to select new S and T targets and repeat the design.

The program enters the reduction computations and, upon completion it prompts for a viewing of the result. If the response is affirmative (default), it plots the reduced controller singular values, closed-loop T and S, and step responses. If the reduction unsuccessful, it can be repeated and you will then be asked to manually select the number of states in the reduced order controller. If the reduction is successful, you may check the robust stability condition with the actual controller (instead of the target).

Finally, the program prompts for the file name where the continuous-time state-space representation of the controller is saved.

1. h_preh: Simulation initialization file. Performs controller discretization, prefiltering and anti-windup design. This is an viewable/editable m-file.
Usage: >> h_preh
Outputs: The discrete controller parameters needed to initialize the multivariable controller block are left in the workspace.
User-input prompts: The first prompt is the sampling time; normally, you should use the default value (0.5/60 min). Then you are asked to supply a value for the anti-windup observer penalty factor. Larger values will implement a slower observer; these allow for deeper excursion of the control signal into the saturation and higher overshoots but may be necessary to avoid directionality problems.

In the next prompt, you are asked to specify the controller file name, e.g., from h_hinf. If you would like to try a different controller design procedure, all you need is to save the continuous time controller in the variables [Acr, Bcr, Ccr, Dcr]. The final two prompts are the zeros and poles of an equivalent prefilter at the reference input, but implemented as a cascade and feedback compensator (see Chapter 5.5.2 in the notes).

After the program is done, you may start the Simulink simulation.